Informative Path Planning with a Human Path Constraint

Daqign Yi, Michael A. Goodrich and Kevin D. Seppi
Department of Computer Science
Brigham Young University
Provo, Utah 84604
Email: daqing.yi@byu.edu, {mike, kseppi}@cs.byu.edu

***Abstract*—One way for a human and a robot to collaborate on a search task is for the human to specify constraints on the robot's path and then allow the robot to find an optimal path subject to these constraints. This paper presents an anytime solution to the robot's path-planning problem when the human specifies a path constraint and an acceptable amount of deviation from this path. The robot's objective is to maximize information gathered during the search subject to this constraint. We first discretize the path constraint and then convert the resulting problem into a multi-partite graph. Information maximization becomes a submodular orienteering problem on this topology structure. Backtracking is used to generate an efficient heuristic for solving this problem, and an expanding tree is used to facilitate an anytime algorithm.**

I. Introduction

In problems that require searching for an object of interest, robots can make human efforts more effective because robots can be more robust to environmental contamination and can sense things beyond of human capabilities. Designing the interactions between a human and a robot for search is no longer constrained to Sheridan's levels of autonomy [1]. Alternatives to Sheridan's levels for a search task include interactions where the human manages the robot either by shaping the information used by the robot to make decisions [2] or by imposing constraints on robot and then allowing the robot to flexibly plan within those constraints [2, 3, 4].

In this paper, we consider constraint-based interactions between a human and a robot for a search task. Rather than consider constraints like no-fly zones or strict waypoints, we explore path constraints imposed by the human and then allow the robot to deviate from the path within some specified tolerance. The source of these constraints range from the robot operating as a "wingman", to a co-located human searcher, to a human telling the robot to approximate the shortest path to an object of interest while gathering maximal information. Furthermore, we assume that the robot's path-planner operates on a discretized representation of the environment and that the robot's sensor footprint covers multiple cells in the discrete representation. Finally, we assume that the robot's sensor becomes less accurate as the distance between the robot and an object of interest grows. These assumptions make the path-planning problem a constrained version of the submodular orienteering problem on a graph.

This paper presents an anytime approximate solution to this problem that uses backtracking to generate an efficient heuristic and an expanding tree. Section III shows how a multi-partite graph is generated using the human-path constraint and formulates the problem into a class of submodular orienteering on a multi-partite graph. Section IV describes the algorithm in the context of solving the submodular orienteering problem and presents a proof that the algorithm will always find the optimal solution, given enough time. Section V introduces a robot wingman problem to demonstrate the performance and efficiency of the algorithm.

II. Related Work

By modeling the objective of a search task using information measurement, previous work has focused on planning a path for a robot to maximize information gained in a reasonable time, especially in a large problem spaces. In a continuous space, a rapidly-exploring random tree can solve the information maximization path planning problem, and also shows good efficiency in an online optimization [5]. If there exists a temporal logical constraint, a receding horizon planning can be used [6].

If the robot's observation model is a coverage instead of a point, the objective of the path planning becomes *maximum coverage*. Maximum coverage is a classic NP-hard combinatorial optimization problem [7], which includes unignorable overlaps. The total information of a set of observation coverages is measured by mutual information, which implies a property of "nondecreasing submodularity" [8]. A greedy approximation with known performance bound can efficiently exploit the submodularity property of mutual information [8]. A branch and bound approach can also be used in informative path planning [9].

Maximizing the reward collected from a limited-length graph walk is usually known as an *orienteering problem* [10], in which the total reward is a summation of the rewards of visited vertices. If the reward function of a vertex has submodularity as in a *maximum coverage problem*, the problem is defined as a *submodular orienteering problem* [11]. Unfortunately in the submodular orienteering case the location of the robot at time t constrains the reachable locations at time $t + 1$. Thus, naïvely applying a greedy algorithm to the submodular orienteering case, that is, with a "teleport" assumption, yields poor results [12]. For a generalization of the submodular orienteering problem in which the neighboring constraint can be converted into a time budget, a recursive greedy algorithm can be applied [11].

III. Problem Statement

In this section we convert a human-path constraint into a multi-partite graph and formulate the informative path planning problem into a class of submodular orienteering on a multi-partite graph.

A. *Information maximization path planning*

Consider a discretized map of the world formed by a set of cells $\mathbf{S}$ and suppose that the robot moves with constant speed from a cell to its neighbors. In the search task, each cell in a discretized map is assigned an entropy value to represent the information distribution. Because the robot's path must be connected, the robot's motion is constrained by a graph topology determined by the cell neighborhood. In a period of time of length T, we denote the robot's path as $X = [x_1, x_2, \cdots, x_T]$, and note that this path must satisfy the connection constraint on the discretized map. We adopt an observation coverage model for the robot, which means that the robot can observe not only the cell it currently occupies but also neighboring cells within a given range. Let the observation at time step t be O_t^X, which describes both the observed cells and how well they are observed. Thus the robot's path, X, induces a sequence of observations $\mathbf{O}^X = \{O_1^X, \cdots, O_{T-1}^X, O_T^X\}$.

We assume that the observation coverage model follows Bayes rule. Thus we can define the information gain of the robot using mutual information $I(\mathbf{S} \mid \mathbf{O}^X) = H(\mathbf{S}) - H(\mathbf{S}, \mathbf{O}^X)$. The entropy reduction over the problem space $\mathbf{S}$ by the observation $\mathbf{O}^X$ is the *information gain* to the robot.

B. *Human path constraint*

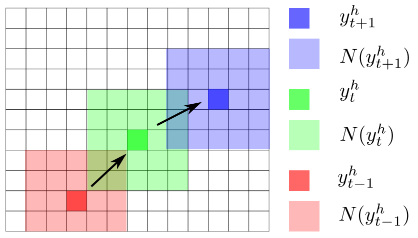

Fig. 1. How the multi-partite graph is obtained.

As discussed in the introduction, there are several ways that a human can constrain the robot's path. Without loss of generality, we adopt a "wingman" approach and assume we have a model that can predict the human's path. We denote the human's T-step path as $Y^h = [y_1^h, y_2^h, \cdots, y_T^h]$. We define a neighbor function $N()$ that represents the assumption from the introduction that the robot can deviate from a constrained path by no more than a given tolerance. At each time step, this neighborhood induces the set of *visitable cells* for the robot, which is denoted by $N(y_t^h)$. Figure 1 gives an example. By organizing the set of visitable cells at time t into a partition of vertices, we can construct a multi-partite graph $G = (V, E, T)$ from the constrained path. A partition $V(t) \in V$ is obtained from the cells in $N(y_t^h)$. The edge set E is determined by the neighborhood of each cell from the discretized map.

Imposing the path constraint Y^h, we define the multi-partite graph as follows. Figure 2 illustrates how the path constraint induces the multi-partite graph for a notional human path. Note that a cell in the discretized map might appear in multiple partitions due to overlaps between sets by $N(y_t^h)$ at different t.

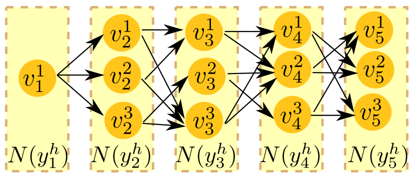

Fig. 2. A multi-partite graph from a human path constraint.

Definition 1 (Multi-Partite Graph). *The multi-partite graph $G = (V, E, T)$ is defined as a graph of T partitions. The vertex set V is defined as $V = \cup_{t=1}^T V(t)$. Each partition $V(t)$ is a set of vertices v_t^i, where t indicates which partition the vertex x is in and i indicates the index of this vertex. Edges are directed, originating from vertices in set $V(t)$ to vertices in set $V(t+1)$. Let $v_t^i \in V(t)$ and $v_{t+1}^j \in V(t+1)$. A directed edge (v_t^i, v_{t+1}^j) connects vertices v_t^i and v_{t+1}^j.*

In order to guarantee that the search process on a multi-partite graph $G = (V, E, T)$ always ends with a path of length T, we use a pruning process to ensure that each vertex can be reached from the previous partition and is connected to a vertex in the next partition. The pruning process includes a forward pruning and a backward pruning. The forward pruning traverses from partition $V(2)$ to partition $V(T)$ and removes any vertex that has no incoming edge; all edges incident to this vertex are also removed. The backward pruning traverses from partition $V(T-1)$ to partition $V(1)$, and removes any vertex that has no outgoing edge; all edges incident to this vertex are also removed.

C. *The Optimization Problem Model*

Without loss of generality, we assume all paths start from the same vertex. Thus, we have only one vertex in partition $V(1)$, as illustrated in Figure 2. Because the objective of the path-planning problem is to maximize mutual information, and because mutual information is a submodular function [8], we find it convenient to shift from the bulky notation for mutual information, $I(\mathbf{S}; \mathbf{O}^X)$, to the more concise notation of a general submodular function, $f(X)$. Because the mutual information $I(\mathbf{S}; \mathbf{O}^X)$ is independent of the sequence of the vertices in a path, we write X as a set in $f()$ for simplicity. $f()$ supports multiple vertices representing the same cell, which is like choosing same position multiple times in a sensor coverage placement problem.

The objective of the search task is to maximize information gain subject to the path constraint. Since the path constraint is encoded as the multi-partite graph, we can restate the objective as maximizing information gain on the multi-partite graph. This yields a submodular orienteering problem on a multi-partite graph $G = (V, E, T)$, which is given as

$$\begin{aligned} &Objective: X^* = \arg\max_X f(X); \\ &Constraint: |X| = T, x_t \in V(t), (x_t, x_{t+1}) \in E. \end{aligned} \tag{1}$$

Exhaustive search could find the optimal path but the time-complexity of such a search makes this unacceptable in all problems except in small problems. A greedy search is efficient but the performance of greedy search on a submodular problem is not guaranteed given a topological constraint [12]. Instead of a greedy heuristic, we develop an alternative heuristic based on the property of mutual information.

Mutual information has two desirable properties that we exploit. First, it is independent of the sequence of vertices on a path and, second, it follows a chain rule. This chain rule property can be written as $f(x_1, x_2, \cdots, x_T) = f(x_1) + f(x_2 \mid x_1) + \cdots + f(x_T \mid x_1, \cdots, x_{T-1})$, which yields a structured Bellman-like equation $\hat{x}_t = \arg\max_{X_t}[f(x_t \mid x_1, \cdots, x_{t-1}) + \max_{X_{t+1},\cdots,X_T} f(x_{t+1}, \cdots, x_T \mid x_1, \cdots, x_t)]$. These structures lead to two key terms: *maximum future reward* and *maximum total reward.*

Definition 2 (**Maximum Future Reward**). *Define the maximum future reward as*

$$h(x_1, \cdots, x_{t'}) = \max_{V(t'+1),\cdots,V(T)} f(x_{t'+1}, \cdots x_T \mid x_1, \cdots, x_{t'}),$$

given the topology constraint $\forall \tau \in \{t'+1, \cdots, T\}, x_\tau \in V(\tau)$ *and* $\forall \tau \in \{t'+2, \cdots, T-1\}, (x_{\tau-1}, x_\tau) \in E$.

Definition 3 (**Maximum Total Reward**). *Define the maximum total reward from choosing* x_t *after* $x_1 \cdots, x_{t'}$ *have been chosen as,* $\forall t > t'$,

$$u(x_t \mid x_1, \cdots, x_{t'}) = f(x_t \mid x_1, \cdots, x_{t'}) + h(x_1, \cdots, x_{t'}, x_t).$$

If we could obtain the values $u(x_t \mid x_1, \cdots, x_{t'})$, then we could greedily chose values for $\hat{x}_t$ as those that maximize $u()$; this would yield an optimal solution as $\hat{x}_t \to x_t$. Unfortunately, the calculation on $u(x_t \mid x_1, \cdots, x_{t'})$ is hard due to the submodularity of $f()$ and the topology constraint. In the next section, we present a heuristic for $u()$ that yields good performance in empirical studies.

IV. Approximate Anytime Solution

In this section, we use backtracking to estimate the maximum total reward and use this estimate as our search heuristic. We then use an expanding tree to create an anytime algorithm approximate solution to the submodular orienteering problem on the multi-partite graph.

A. Using the Search Heuristic from Backtracking

In a graph search process, if a sub-path $\{v_1, \cdots, v_{t'}\}$ has been visited, we can use the following property to approximate the maximum future reward.

Property 1.

$$h(x_1, \cdots x_{t'}) = \max_{x_{t'+1} \in V(t'+1) \wedge (v_{t'}, x_{t'+1}) \in E} u(x_{t'+1} \mid x_1, \cdots x_{t'}).$$

Property 1 implies that the maximum future reward at partition $V(t)$ can be estimated from the maximum total reward at partition $V(t+1)$. This means that the maximum total rewards could be estimated by using a backtracking process. We propose a backtracking process in Algorithm 1.

Algorithm 1 **BT**$(\{v_1, \cdots, v_{t'}\}, G)$ - Backtracking

```
Input: a sub-path $\{v_1, \cdots, v_{t'}\}$, and multi-partite graph $G = (V, E, T)$
Output: $\hat{u}(v_1, \cdots, v_{t'}, v_{t'+1}), \forall v_{t'+1} \in V(t'+1)$
 1: for $t = T : -1 : t'+1$ do
 2:     for $v_t \in V(t)$ do
 3:         if $t == T$ then
 4:             $\hat{u}(v_T \mid v_1, \cdots, v_{t'}) = f(v_T \mid v_1, \cdots, v_{t'})$
 5:         else
 6:             $\hat{h}(v_1, \cdots, v_{t'}, v_t) = \max_{x_{t+1} \in V(t+1) \wedge (v_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'})$
 7:             $\hat{u}(v_t \mid v_1, \cdots, v_{t'}) = f(v_t \mid v_1, \cdots, v_{t'}) + \hat{h}(v_1, \cdots, v_{t'}, v_t)$
 8:         end if
 9:     end for
10: end for
11: return $\hat{u}(v_1, \cdots, v_{t'}, v_{t'+1}), \forall v_{t'+1} \in V(t'+1)$
```

Algorithm 1 estimates the maximum total rewards of the vertices in $V(t'+1)$. The backtracking starts at partition $V(T)$ and goes back to $V(t'+1)$ in order to propagate the estimated maximum future rewards. For a vertex $v(t)$, the maximum future reward is estimated based on the estimated maximum total rewards of all the connected vertices in partition $V(t+1)$. The estimated total reward is then obtained by adding the estimated instant reward of $v(t)$ with the estimated maximum future reward of $v(t)$. The backtracking process in Algorithm 1 satisfies Lemma 1.

Lemma 1. *"Backtracking" in Algorithm 1 never underestimates the maximum total reward, which means*

$$\forall t \geq t', \hat{u}(x_t \mid v_1, \cdots, v_{t'}) \geq u(x_t \mid v_1, \cdots, v_{t'}). \quad (2)$$

Proof: The proof is given in Appendix B. ■

Note that Lemma 1 holds even when there are multiple vertices in a multi-partite graph generated from same cell. This is because the submodularity of $f()$ is preserved and the proof depends primarily on submodularity. However, multiple vertices generated from same cell in a path increase the degree to which the reward is overestimated.

B. Expanding Tree Search

Since the heuristic is not guaranteed to produce an optimal solution, we create an anytime algorithm that allows us to continue the search process until a time limit is exceeded or the search is completed exhaustively. In order to track the anytime search process, the algorithm uses an expanding tree. The expanding tree is the tree produced by repeated depth-first traversals of the multi-partite graph [13].

Definition 4 (**Expanding Tree**). *An expanding tree* $G_T = (N, L, T)$ *obtained from a multi-partite graph* $G = (V, E, T)$ *is the tree produced by a depth first traversal of* G. T *is the depth of the tree, which is determined by the number of partitions in a multi-partite graph* G. N *is the node set. Each* $n_t^{i(j)} \in N$ *indicates the relevant vertex in the multi-partite graph* G, *in which* t *shows the index of the time partition,* i *shows the index of the corresponding vertex from within that* *partition and (j) shows the index of a vertex in $V(t-1)$ that has an out edge to vertex i. L is the directed link set. $(n_t^{i(k)}, n_{t+1}^{j(i)}) \in L$ is determined by $(v_t^i, v_{t+1}^j) \in E$.*

We assign the *type* to each node, which are *New* (a node has been created but not expanded), *Expanded* (a node that has all child nodes created) and *Frozen* (a node that has been created but will not be expanded). Each path in an expanding tree is derived from a unique depth-first traversal of the corresponding mutli-partite graph. We use $v(n_{t+1}^{j(i)})$ to denote a vertex mapped from a node. For a node $n_t^{i(j)}$, we use $path(n_t^{i(j)})$ to denote the implicit path from the start position to the corresponding vertex of the multi-partite graph, the cardinality of which is t.

We can now present Algorithm 2 for a single search iteration. It is used as one run in the anytime framework.

Algorithm 2 $\mathbf{NERB}(n_{t'}, G, G_T)$ - Node Expanding with Recursive Backtracking

```
Input: Expanding Node $n_{t'}$, Multi-partite graph $G = (V, E, T)$, Expanding tree $G_T = (N, L, T)$
Output: $solution$ of a complete path
 1: $solution = path(n_{t'})$
 2: for $t = t' : 1 : T-1$ do
 3:     Create all $child(n_{t'}) = \{n_{t'+1} \mid v(n_{t'+1}) \in V(t'+1) \wedge (v(n_{t'}), v(n_{t'+1})) \in E\}$
 4:     Add $child(n_{t'})$ as the child nodes of $n_{t'}$
 5:     $n_{t'}.state = Expanded$
 6:     $\hat{u}(v_{t'+1} \mid path(n_{t'})) = \mathbf{BT}(path(n_{t'}), G)$
 7:     $\hat{n}_{t'+1} = \arg\max_{n_{t'+1} \in child(n_{t'})} \hat{u}(n_{t'+1} \mid path(n_{t'}))$
 8:     $solution = solution \bigcup \{\hat{n}_{t'+1}\}$
 9: end for
10: return $solution$
```

C. Adding node freeze to the expanding tree

Because Lemma 1 tells us that the backtracking process never underestimates the maximum total reward of a node, we can use the estimated maximum total reward of a node to evaluate whether the node might lead to a path that returns a bigger reward than the current best one. A node is not in a path that has bigger reward if its estimated value is smaller than the current best solution. We can freeze this node, which means that we are not going to expand any of its descendant nodes. At each iteration we find a new solution, we can call a node freeze process to update the states of the nodes in the expanding tree. This process is given in Algorithm 3.

Algorithm 3 $\mathbf{NF}(G_T, \theta^*)$ - Node Freeze

```
Input: an expanding tree $G_T = (N, L, T)$, the reward of found maximum reward path $\theta^*$
1: for $n_t \in N$ and $n_t.state == New$ do
2:     if $f(path(n_t)) + \hat{h}(path(n_t)) \le \theta^*$ then
3:         $n_t.state = Frozen$
4:     end if
5: end for
```

Algorithm 4 combines Algorithm 1, Algorithm 2, and Algorithm 3 to yield the anytime algorithm. The expanding tree starts with just a root node, which is the start vertex. When a node is created, the state of the node is *New*. Expanding a node in Algorithm 4 means creating all of its children nodes and changing the state of this node to *Expanded*. When a child node is created, the estimated maximum total reward is calculated using Algorithm 1 and stored. Each run of Algorithm 2 returns a complete path as a solution. When a new complete path has been returned, the freeze process defined in Algorithm 3 is executed by checking estimated maximum total rewards stored in each nodes in the state of *New*. The next run of the search starts from the *N*ew node n_t that has the largest estimated reward $f(path(n_t)) + \hat{h}(path(n_t))$. Starting from this node, the next call to Algorithm 2 generates another complete path. This anytime algorithm stops at a pre-specified number of iterations or when there is no *New* node remaining.

Algorithm 4 Anytime Algorithm Framework

```
Input: Expanding Tree $G_T = (N, L, T)$, and multi-partite graph $G = (V, E, T)$;
 1: Initial expanding tree $G_t(N, L, T)$ with $v_1$ as root node
 2: $maxPath = NULL$, $newPath = NULL$
 3: $n' = G_T.root$
 4: while $n'! = NULL$ do
 5:     $newPath = \mathbf{NERB}(n', G, G_T)$
 6:     if $(f(newPath) > f(maxPath))$ then
 7:         $maxPath = newPath$
 8:         post $maxPath$
 9:     end if
10:     $\mathbf{NF}(G_T, f(maxPath))$
11:     $n' = \arg\max_{\{n \mid n \in N \wedge n.state == New\}}(f(path(n)) + \hat{h}(path(n)))$
12: end while
```

Algorithm 4 is optimal as shown in Theorem 1 given here.

Theorem 1. *The anytime algorithm framework in Algorithm 4 can always find an optimal solution given enough time.*

Proof: The proof is by contradiction and is similar in spirit to the proof of optimality for the well-known A* algorithm. Since Algorithm 4 keeps expanding until no *New* nodes remain, as long as any node in the optimal path will never be frozen, the search will reach the optimal terminal node.

Assume that one of the nodes n_t^* in the optimal path can be frozen. This means that $f(path(n_t^*) + \hat{h}(path(n_t^*))) \le f(path(n_T'))$, in which n_T' is another terminal node but not the terminal node for the optimal path. As a result, when n_T' has been reached, it will freeze node n_t^*.

However, since node n_t^* is in a path to an optimal terminal node, $f(path(n_t^*)) + h(path(n_t^*)) > f(path(n_T'))$. Also we have $f(path(n_t^*)) + \hat{h}(path(n_t^*)) \ge f(path(n_t^*)) + h(path(n_t^*))$ by Lemma 1. Thus we have $f(path(n_t^*)) + \hat{h}(path(n_t^*) > f(path(n_T'))$, which is a contradiction.

Therefore, a node in a path to an optimal terminal node will never be frozen by any non-optimal terminal node. ■

V. An Application to Robot Wingman

In this section, we apply Algorithm 4 to the robot wingman problem [14]. Let R_{flank} denote the *flank support range*, which determines the area that a robot wingman is expected to stay

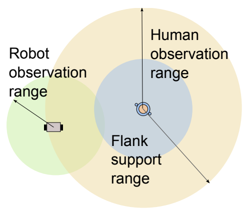

Fig. 3. A Robot Wingman Framework.

in when a human is moving; this is illustrated in Figure 3. The robot wingman constraint requires that $\forall t, ||x_t - y_t^h|| \leq R_{flank}$ or, equivalently, $\forall t \in T, x_t \in N(y_t^h)$.

Consider a two-dimension search space discretized into a world of hexagonal cells. This discretization gives a constant distance from the center of one cell to any of its immediate neighbors, which facilitates modeling the agent observation range. Moreover, a hexagonal tessellation is consistent with the assumption that we made that the robot would move at constant speed from one cell to another; in a hexagonal tessellation, the distances between the centers of all neighboring cells and the current cell is constant.

The observation model of an agent uses the likelihood of detecting an object of interest in cell i and follows Bayes rule in updating the posterior [14]. Since we know the human's path, Y^h, we can use the human's observation model to predict what the human could observe and update the prior distribution of information to reflect this. The simulation results we present assume that human observations have already been factored into the prior distribution of information.

A. Performance

We simulate the use of the algorithm in a search space in which the entropy of each cell is randomly generated. The simulation results are aggregated from 20 runs of each case. The parameters $R_{flank} = 2$ is used for the neighboring function of the human path constraint and $R_{obs}^{robot} = 2$ is used for the robot's observation range.

We use a "fully expanded tree size" to measure the *problem size*, which depends on the planning length and the vertex connectivities. Due to the human constraint, the planning length is determined by the human's path length. The performance of the heuristic is measured by the *percentage of optimal at first run*, that is a percentage computed from the value obtain in the first run of Algorithm 4 over the optimal value. High values of this metric indicate that the heuristic is useful. The greedy heuristic [8], which chooses the maximum next step, is imported to compare with the backtracking heuristic. Figure 4 shows the comparison between two types of heuristic on the percentages of the optimal as a function of different planning lengths. We can see that the performance of the backtracking heuristic significantly surpasses that of the greedy heuristic.

Naturally, as the size of the search space expands, the difficulty in finding the optimal solution using an exhaustive search grows. Since we want to understand how well our anytime algorithm performs for problems that are too big to search exhaustively, we bound the payoff for the optimal path by using a "teleport" search in which the robot can bounce from region to region without following a connected path. Figure 5 shows the rewards collected using the path produced by the backtracking heuristic normalized by the rewards collected by the"teleport" path for large search spaces. Again, the backtracking heuristic is much better than the greedy solution (similarly normalized).

We use *percentage of nodes explored* to indicate the efficiency of the anytime algorithm framework. In particular, we are interested in whether freezing nodes improves search efficiency. Figure 6 shows that the percentage of nodes explored decreases significantly when the problem size is expanded. Since the anytime algorithm becomes an exhaustive search in the absence of freezing nodes (and hence follows the size of the search space in the figure), this figure indicates that the percentage of nodes expanded is significantly decreased by freezing nodes. In the anytime algorithm, the exploration might not stop when the optimal is found, due to the existence of overestimation. If current best of a search can reach the optimal very quickly, it means that a best solution found in a fixed time has high probability of being optimal. We use *percentage of runs reaching the optimal* to measure this optimal search

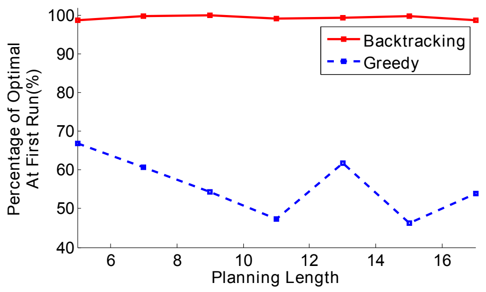

Fig. 4. The performance comparison between the backtracking heuristic and the greedy heuristic.

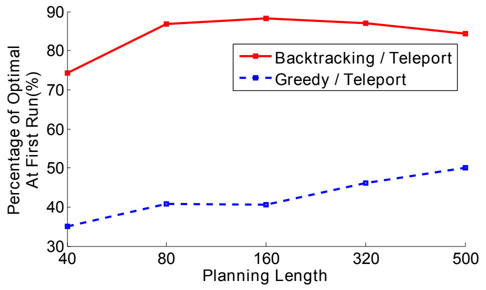

Fig. 5. The performance comparison between the backtracking heuristic and the greedy heuristic in a large problem space.

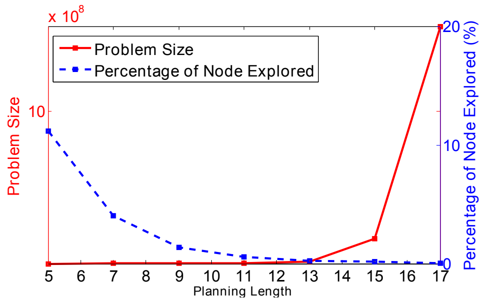

Fig. 6. Problem size and exploration ration with different planning lengths.

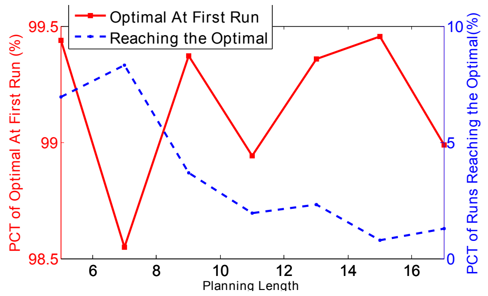

Fig. 7. Percentages of optimal at first iteration and Percentage of runs reaching the optimal with different planning lengths.

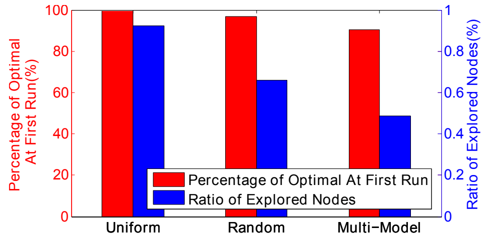

Fig. 8. Performance in different types of environments.

capability of Algorithm 4. Figure 7 illustrates that the anytime algorithm can find the optimal solution relatively quickly.

B. Robustness

We extend the search environment from *random* to *uniform* and *multimodal*. Uniform indicates that the entropies in different cells are identical, and multimodal indicates that the entropy distribution among cells is a multi-modal spatial distribution. Figure 8 shows that Algorithm 4 consistently performs well in different types of search environment.

In order to illustrate how well Algorithm 4 adapts to different human path constraints, we introduce five common patterns of paths executed by a human in a search task, which are *line*, *spiral*, *lawn-mower*, *arc* and *loitering*. Figure 9 shows examples on these five patterns. Due to the wingman constraint, different human paths lead to different problem sizes and different ratios of overlap in the coverage at two different time steps. For this comparison we hold the number of time steps fixed at 11 over different patterns as in Figure 9.

Figure 10 shows that the problem size varies significantly depending on the type of path, though the planning length is identical. Interestingly, Algorithm 4 shows better efficiency in larger problem size. In Figure 11, we can see that the ratios of explored nodes are relatively smaller in the patterns of "spiral", "lawn-mower" and "loitering", in which the problem sizes are relatively larger in Figure 10. We can see that the percentage

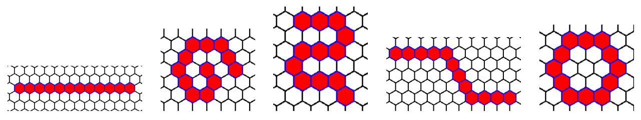
(a) Line (b) Spiral (c) Lawn-mower (d) Arc (e) Loitering

Fig. 9. Different patterns of human path.

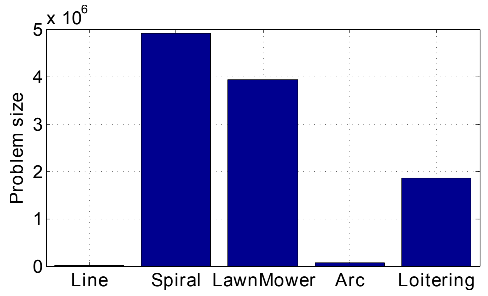

Fig. 10. Problem size in different patterns of human paths.

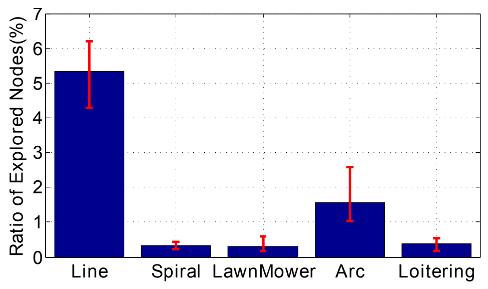

Fig. 11. Exploration ratios in different patterns of human paths.

of optimal at first run are all close to the optimum in all the patterns in Figure 12, which implies the goodness of the backtracking heuristic.

VI. Summary

In this paper, we use a human path to form a path constraint and seek to maximize the information gathered by a robot gathered in a search task. The resulting information maximization path planning is identified as a constrained submodular orienteering problem on a multi-partite graph. We present an anytime algorithm that used a planning heuristic based on backtracking to efficiently find a high quality path. We use a node freeze process to avoid an exhaustive search, yet we prove that this process always preserves the ability of the algorithm to find an optimal solution. We have also shown empirically that this approach substantially reduces the complexity of the resulting search.

References

[1] T. B. Sheridan and W. L. Verplank, "Human and computer control of undersea teleoperators," DTIC Document, Tech. Rep., 1978.

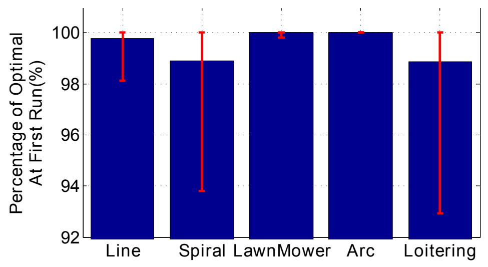

Fig. 12. Percentages of optimal at first iteration in different patterns of human paths.

[2] L. Lin and M. Goodrich, “A bayesian approach to modeling lost person behaviors based on terrain features in wilderness search andrescue,” *Computational and Mathematical Organization Theory*, vol. 16, no. 3, pp. 300–323, 2010.

[3] S. Clark and M. Goodrich, “A hierarchical flight planner for sensor-driven uav missions,” in *2013 IEEE RO-MAN*, Aug 2013, pp. 509–514.

[4] J. M. Bradshaw, M. Sierhuis, A. Acquisti, P. Feltovich, R. Hoffman, R. Jeffers, D. Prescott, N. Suri, A. Uszok, and R. V. Hoof, “Agent autonomy,” in *Adjustable Autonomy and Human-Agent Teamwork in Practice: An Interim Report on Space Applications*, H. Hexmoor, R. Falcone, and C. Castelfranchi, Eds. Kluwer, 2002.

[5] D. S. Levine, “Information-rich path planning under general constraints using rapidly-exploring random trees,” Ph.D. dissertation, MIT, 2010.

[6] A. Jones, M. Schwager, and C. Belta, “A receding horizon algorithm for informative path planning with temporal logic constraints,” in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, May 2013, pp. 5019–5024.

[7] N. Megiddo, E. Zemel, and S. L. Hakimi, “The maximum coverage location problem,” *SIAM Journal on Algebraic Discrete Methods*, vol. 4, no. 2, pp. 253–261, 1983.

[8] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, “Efficient informative sensing using multiple robots,” *J. Artif. Int. Res.*, vol. 34, no. 1, pp. 707–755, Apr. 2009.

[9] J. Binney and G. Sukhatme, “Branch and bound for informative path planning,” in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, May 2012, pp. 2147–2154.

[10] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, “The orienteering problem: A survey,” *European Journal of Operational Research*, vol. 209, no. 1, pp. 1 – 10, 2011.

[11] C. Chekuri and M. Pal, “A recursive greedy algorithm for walks in directed graphs,” in *46th Annual IEEE Symposium on Foundations of Computer Science, 2005. FOCS 2005.*, Oct 2005, pp. 245–253.

[12] A. Krause and D. Golovin, “Submodular function maximization,” *Tractability: Practical Approaches to Hard Problems*, vol. 3, 2012.

[13] K. Rosen, *Discrete Mathematics and Its Applications*. McGraw-Hill Science, 2011.

[14] M. Goodrich and D. Yi, “Toward task-based mental models of human-robot teaming: A bayesian approach,” in *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, ser. Lecture Notes in Computer Science, R. Shumaker, Ed. Springer, 2013, vol. 8021, pp. 267–276.

Appendix A
Proof of a Useful Property

The following property is necessary to prove Lemma 1.

Property 2.

$$u(x_t \mid x_1, \cdots, x_{t'}) = f(x_t \mid \tilde{X}(x_t), x_1, \cdots, x_{t'}) + \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid x_1, \cdots, x_{t'}), \quad (3)$$

in which

$$\tilde{X}(x_t) = \arg \max_{V(t+1) \cdots V(T)} f(x_{t+1} \cdots x_T \mid x_1, \cdots, x_{t'}) \quad (4)$$

subject to the constraint

$$\forall \tau \in \{t+1, \cdots, T\}, x_\tau \in V(\tau) \wedge (x_{\tau-1}, x_\tau) \in E. \quad (5)$$

Proof: By chain rule, we have $u(x_t \mid x_1, \cdots, x_{t'}) = f(\tilde{X}(x_t) \mid x_1, \cdots, x_{t'}) + f(x_t \mid x_1, \cdots, x_{t'}, \tilde{X}(x_t))$*. By decomposing the constraint in* (5) *into* $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$ *and* $\forall t'' \in [t+2, T], x(t'') \in V(t'') \wedge (x_{t''-1}, x_{t''}) \in E$*, equation* (4) *can be* $f(\tilde{X}(x_t) \mid x_1, \cdots, x_{t'}) = \max_{V_{t+1}} u(x_{t+1} \mid x_1, \cdots, x_{t'})$ *subject to the constraint* $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$*. Thus equation* (3) *can be obtained.*

■

Appendix B
Proof of Lemma 1

Proof: Equation (2) can be proven using induction as follows. We have following two propositions, corresponding to the *basis case* and *induction step*, which are

- **proposition 1** $\forall x_T \in V(T), \hat{u}(x_T \mid v_1, \cdots, v_{t'}) = u(x_T \mid v_1, \cdots, v_{t'})$;
- **proposition 2** If $\forall x_{t+1} \in V(t+1), \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x_{t+1} \mid v_1, \cdots, v_{t'})$, then $\forall x_t \in V(t), \hat{u}(x_t \mid v_1, \cdots, v_{t'}) \geq u(x_t \mid v_1, \cdots, v_{t'})$.

Basis: At time T, we have $u(x_T \mid v_1, \cdots, v_{t'}) = f(x_T \mid v_1, \cdots, v_{t'})$ and $\hat{u}(x_T \mid v_1, \cdots, v_{t'}) = f(x_T \mid v_1, \cdots, v_{t'})$. Thus proposition 1 is true.

Induction Step: The definition of $u(x_t \mid v_1, \cdots, v_{t'})$, Property 2, and the definition of $\hat{u}(x_t \mid v_1, \cdots, v_{t'})$ in Algorithm 1, imply $\hat{u}(x_t \mid v_1, \cdots, v_{t'}) - u(x_t \mid v_1, \cdots, v_{t'}) = [f(x_t \mid v_1, \cdots, v_{t'}) - f(x_t \mid v_1, \cdots, v_{t'}, \tilde{x}_{t+1}, \cdots \tilde{x}_T)] + [\max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) - \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'})]$. By submodularity, we know that $f(x_t \mid v_1, \cdots, v_{t'}) - f(x_t \mid v_1, \cdots, v_{t'}, \tilde{x}_{t+1}, \cdots \tilde{x}_T) \geq 0$.

Define the following two values

$$\begin{aligned} x^a_{t+1} &= \arg \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_{t'}, \cdots, v_1) \\ x^b_{t+1} &= \arg \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'}). \end{aligned}$$

Both x^a_{t+1} and x^b_{t+1} belong to the set of vertices that satisfy the constraint $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$. Since x^a_{t+1} is the answer to $\arg\max \hat{u}(\cdot)$, we have $\hat{u}(x^a_{t+1} \mid v_1, \cdots, v_{t'}) \geq \hat{u}(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By the induction hypothesis, $\hat{u}(x^b_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By transitivity, we have $\hat{u}(x^a_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By the definitions x^a_{t+1} and x^b_{t+1}, which equals to $\max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) - \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'}) \geq 0$. Thus proposition 2 is true.

Conclusion: Since the basis case and induction step are true, Equation (2) follows. ■