Cooperative Graph Search Using Fractal Decomposition*

James R. Riehl and João P. Hespanha†

***Abstract*— We present an algorithm based on hierarchical decomposition that finds close-to-optimal search paths for a cooperative team of agents searching for one or more targets on a graph. The method partitions the graph to create a high-level problem and several lower-level problems. Since the computations on each level are identical, the lower-level problems can be further decomposed. In this way, the problem becomes fractal in nature. We use best-case and worst-case instances of the decomposed problem to establish upper and lower bounds on the optimal search reward, and the bounds are determined with much less computation than what is required to solve the full problem. We show that as the number of decomposition levels increases, the computational complexity approaches $O(n)$ at the expense of looser bounds on the optimal reward. A large-scale test case shows that this method is computationally fast, produces good results, and achieves true cooperation between agents.**

I. Introduction

We consider the problem of using a team of autonomous agents to search a region for an object whose location is unknown or uncertain. Their objective is to find search paths that maximize a reward function, *e.g.* the probability of finding a target, subject to a cost constraint limiting a quantity such as time or fuel available. The key difficulty of this problem is the computational complexity required to find optimal search paths for the agents. Here, we take the approach of finding approximately optimal search paths by recursively decomposing the problem into several smaller problems, and then bounding the reward of the optimal search path by solving optimistic and pessimistic versions of the decomposed problem.

The theory of search, beginning with work by Koopman [6] [7], Stone [12], and others, was initially motivated by the desire to develop efficient search methods to find enemy marine vessels. More recently, agencies such as the U.S. Coast Guard have applied search theory to search and rescue missions with great success, measured in saved lives [3]. Other interesting search applications include exploration, mining, medicine, and surveillance [4].

Early search theory focused on the allocation of search effort to areas within the search region, which is appropriate to special cases for which finding optimal search paths on these areas is intuitive or searcher motion is unconstrained. If this is not the case, we are presented with the more difficult problem of finding optimal paths for the searchers. There are some studies of the search problem as an optimal control problem in continuous time and space [8], but these generally apply to a very restrictive set of initial target distributions. A more practical approach is to discretize the search space and formulate the search problem on a graph. Then the search path describes a sequence of regions to search along with an amount of time that should be spent in each region. Although this discretization simplifies the problem to some extent, it is still computationally very difficult. Trummel and Weisinger showed that the single-agent search problem is NP-Hard even for a stationary target [13]. Eagle and Yee [2] and Stewart [11] formulated the moving target problem as a nonlinear integer program and proposed branch and bound algorithms to solve it. This provided a significant computational reduction over a total enumeration, but the problem still has exponential complexity. DasGupta *et al.* presented an approximate solution for the stationary target search based on an aggregation of the search space using a graph partition [1]. We used a similar approach in [10], but with a fractal formulation, that is, the partitioning process could be implemented on multiple levels. All of the results mentioned so far are for a single searcher.

The problem becomes even more complex when we consider a team of agents that are cooperatively searching for a target. For this reason, literature on the cooperative search problem typically tries to find good approximate solutions. Polycarpou *et al.* introduced a short-horizon look-ahead approach similar to model predictive control where the searchers share information with each other [9]. Some advantages of this method are that it is distributed, allowing for limited communication between pursuers, and it works for a moving or stationary target. A potential disadvantage is that because all possible search paths are evaluated at each time step, the prediction horizon must be kept short. This means that if there are regions of high target probability separated by a distance that is much greater than an agent can cover over this horizon, the algorithm's performance may suffer. In this paper, we treat the cooperative search problem with a hierarchical graph-based approach, allowing paths to be planned over the entire graph with significantly less computation than that of a full optimal search. Furthermore, the multiple-level graph partitioning process makes this algorithm especially well-suited to searches on regions that have some inherent hierarchical structure.

The remainder of the paper is organized as follows. Section II introduces notation and terminology related to graphs and graph partitioning that will be used throughout the paper. The cooperative search problem is formulated in section III. Section IV describes a method of decomposing the search problem and verifies the resulting upper and lower bounds on the optimal reward. The computational complexity

*This material is based upon work supported by the Institute for Collaborative Biotechnologies through grant DAAD19-03-D-0004 from the U.S. Army Research Office.

†{jriehl,hespanha}@ece.ucsb.edu, Center for Control, Dynamical Systems, and Computation, Electrical and Computer Engineering Department, University of California, Santa Barbara, CA 93106-9560.

results are given in section V. Finally, in section VI, we apply the method to a realistic large-scale problem to show that the method is both computationally fast and that good cooperation is indeed achieved.

II. Graphs, Meta-graphs, and Subgraphs

This section introduces some notation and terminology that we will use in the remainder of this paper. Given a graph $G := (V, E)$ with vertex set V and edge set $E \subset V \times V$, a *partition* $\bar{V} := \{\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_k\}$ of G is a set of disjoint subsets of V such that $\bar{v}_1 \cup \bar{v}_2 \cup \ldots \cup \bar{v}_k = V$. We call these subsets $\bar{v}_i$ *meta-vertices*. For a given meta-vertex $\bar{v}_i \in \bar{V}$, we define the *subgraph of G induced by $\bar{v}_i$* to be the subgraph $G|\bar{v}_i := (\bar{v}_i, E \cap \bar{v}_i \times \bar{v}_i)$.

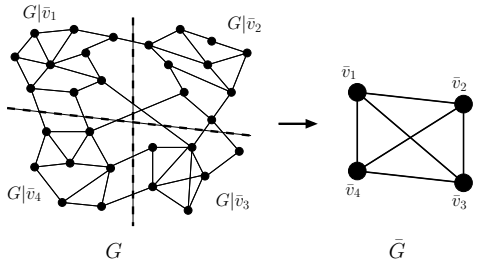

Fig. 1. Example of a graph partitioned into 4 *meta-vertices*.

Figure 1 shows an example of the graph partitioning process, where the dashed lines through G separate the partitioned subgraphs $G|\bar{v}_i$, which are represented by meta-vertices in $\bar{G}$.

Given a partition $\bar{V}$ of the vertex set V, we define the *meta-graph of G induced by the partition $\bar{V}$* to be the graph $\bar{G} := (\bar{V}, \bar{E})$ with an edge $\bar{e} \in \bar{E}$ between *meta-vertices* $\bar{v}_i, \bar{v}_j \in \bar{V}$ if and only if G has at least one edge between vertices v and v' for some $v \in \bar{v}_i$ and $v' \in \bar{v}_j$. In general, there may exist several such edges $e \in E$ and we call these the *edges associated with the meta-edge $\bar{e}$*.

III. Cooperative Graph Search Problem

Consider a team of s agents searching for one or more objects in a bounded region represented by a graph. Each vertex has a reward, generally relating to the probability of finding an object at that vertex, and a cost representing a quantity such as time or energy spent searching that vertex. Each edge also has a cost, representing the cost incurred in transit between vertices. The team's goal is to find paths on the graph for each searcher that maximize the total reward collected by the team subject to a cost constraint on the individual agents.

Data

$G := (V, E)$	directed location graph
$O := \{1, \ldots, o_{\max}\}$	vertex occupancy set
$r : V \times O \to [0, \infty)$	vertex reward function
$c_v : V \times O \to [0, \infty)$	vertex cost function
$c_e : E \to [0, \infty)$	edge cost function
$L \in [0, \infty)$	cost bound

In the above, $o_{\max} \in \{1, 2 \ldots, s\}$ is the maximum number of searchers allowed to occupy a single vertex. Note that the vertex cost and reward functions depend on the occupancy of the vertex. The reason for this will be made clear in section IV.

Search Path A *search path* in G is a sequence of vertices,

$$p := (v_1, v_2, \ldots, v_{f-1}, v_f), \qquad (v_i, v_{i+1}) \in E,$$

where f is the length of the path. The *path-cost* is given by

$$C(p) := \sum_{i=1}^{f-1} c_e(v_i, v_{i+1}) + \sum_{i=1}^{f} c_v(v_i),$$

and the *path-reward* is given by

$$R(p) := \sum_{v \in p} r(v), \tag{1}$$

where the sum in (1) is taken with no repetitions, that is, if a vertex appears in p more than once, it is only included in the summation once. This represents the fact that the reward of a vertex can only be collected once.

The search problem for a single agent is to find the path p that maximizes the reward $R(p)$ subject to the cost constraint $C(p) \leq L$.

Cooperative Framework As discussed in the introduction, there is significant existing literature on the single-agent search problem, but the cooperative search problem is inherently more complex. One way to approach the multiple-agent problem would be to set up s identical single-agent search problems on the location graph G and have the agents start in different strategic positions, but this is not a cooperative solution and could result in overlapping search paths. For the team to fully cooperate, we must consider the problem as a whole. We can do this by creating a graph in which a vertex represents the locations of all agents, *i.e.* the full graph will consist of up to n^s nodes. We call this new expanded graph the *team-graph induced by G* and denote it by $\mathbf{G} := (\mathbf{V}, \mathbf{E})$ (Bold face notation is used for all data and functions related to the team-graph).

The *team-vertex* set $\mathbf{V}$ consists of s-length vectors whose entries are the vertex locations in V of each member of the team. We write the expanded team-vertex as $\mathbf{v} = (\mathbf{v}(1), \mathbf{v}(2), \ldots, \mathbf{v}(s))$, where $\mathbf{v}(a) \in V$ is the location of agent a when the team is at team-vertex $\mathbf{v}$. We construct the set $\mathbf{V}$ by including team-vertices for all possible configurations of searchers in V such that the vertex occupancy $o_{\max}$ is not exceeded. An edge connects vertices $\mathbf{v}$ and $\mathbf{v}'$ if there is an edge in E between $\mathbf{v}(a)$ and $\mathbf{v}'(a)$ for each agent a. We also allow members of the team to stay at a vertex. This is useful in the case that some searchers reach their cost limit before others. We can write the team-edge set as

$$\mathbf{E} := \{(\mathbf{v}, \mathbf{v}') : \forall a \ (\mathbf{v}(a), \mathbf{v}'(a)) \in E \cup (\mathbf{v}(a), \mathbf{v}(a))\},$$

where $\mathbf{v}, \mathbf{v}' \in \mathbf{V}$.

Team Search Path We now describe how to construct and evaluate paths in the team-graph based on data for the

location graph. A *team search path* in $\mathbf{G}$ is a sequence of vertices,

$$\mathbf{p} := (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{f-1}, \mathbf{v}_f), \qquad (\mathbf{v}_i, \mathbf{v}_{i+1}) \in \mathbf{E}, \tag{2}$$

where f is the length of the path. Let $\mathbf{p}_a$ denote the path of agent a, that is $\mathbf{p}_a := (\mathbf{v}_1(a), \ldots, \mathbf{v}_f(a))$. The *team path-cost* is the maximum path-cost of any agent in the team and is given by

$$\mathbf{C}(\mathbf{p}) := \max_a C(\mathbf{p}_a)$$

The *team path-reward* is the total reward collected by the search team on the path, which we can express as

$$\mathbf{R}(\mathbf{p}) := \sum_{i=1}^{f} \sum_{v \in \mathbf{v}_i} r(v, o^v_{\mathbf{v}_i}) \kappa(v, i), \tag{3}$$

where $o^v_{\mathbf{v}_i}$ is the occupancy of v when the team is at $\mathbf{v}_i$. The function

$$\kappa(v, i) := \begin{cases} 1, & v \notin \bigcup_{j=1}^{i-1} \mathbf{v}_j \\ 0, & \text{otherwise} \end{cases}.$$

encodes the property that the reward for a vertex in G may only be collected once by the search team.

Objective Given a cost bound L, denote the maximum reward that s searchers can collect on G by $R^*_G(s, L)$. We can write the objective for the cooperative search problem as follows:

$$R^*_G(s, L) := \max_{\mathbf{p}} \mathbf{R}(\mathbf{p}) \quad \text{s. t.} \quad \mathbf{C}(\mathbf{p}) \leq L \tag{4}$$

IV. Fractal Decomposition

We have now formulated the cooperative search problem, and although there are known methods to solve it, the computational complexity is very high (see Section V). In this section of the paper, we propose a method of decomposing the problem to generate lower and upper bounds on the optimal reward. Additionally, the method is designed to generate problems that are exactly in the form described above. Hence, it may be applied recursively on as many levels as the problem will allow.

A. Worst-case cooperative meta-graph search

Let $\bar{G}_{\text{worst}} := (\bar{V}, \bar{E})$ be a meta-graph of G. Our goal is to formulate the worst-case problem such that its solution will be a lower bound on the optimal reward of (4).

We first choose a meta-vertex maximum occupancy $\bar{o}_{\max}$, yielding the occupancy set $\bar{O} := \{1, \ldots, \bar{o}_{\max}\}$, and then choose a cost assignment $l : \bar{V} \times \bar{O} \to [0, \infty)$. These choices are a degree of freedom for the user, but they should be chosen carefully as they may significantly affect the tightness of the bounds on the optimal reward. The best choices will depend on the structure of the graph as well as the number of decomposition levels.

The meta-vertex cost and reward functions are defined by solving cooperative search problems on their corresponding subgraphs:

$$r_{\text{worst}}(\bar{v}, \bar{o}) := R^*_{G|\bar{v}}(\bar{o}, l), \tag{5}$$

$$c_{v_{\text{worst}}}(\bar{v}, \bar{o}) := \mathbf{C}(\mathbf{p}^*(\bar{v}, \bar{o})), \qquad \forall \bar{o} \in \bar{O}, \forall \bar{v} \in \bar{V}, \tag{6}$$

where $\mathbf{p}^*(\bar{v}, \bar{o})$ is the optimal team search path in $G|\bar{v}$ that generates the maximum reward $R^*_{G|\bar{v}}(\bar{o}, l)$. Let $p^*_a(\bar{v}, \bar{o}, i)$ denote the i^{th} vertex in agent a's optimal path on meta-vertex $\bar{v}$, having occupancy $\bar{o}$. We define the cost of an edge from $\bar{v}$ to $\bar{v}'$ by pairing all final vertices of paths computed in $\bar{v}$ with all starting vertices of paths computed in $\bar{v}'$, computing the costs of the shortest paths between them, and taking the maximum of these costs. Figure 2 diagrams this process for two meta-vertices for which $o_{\max} = 2$. Suppose that the dotted lines represent optimal paths computed on both meta-vertices for an occupancy of 1, and the dashed lines are the paths computed for an occupancy of 2. The highlighted vertices represent the pair of ({final vertices in $\bar{v}$}, {starting vertices in $\bar{v}'$}) that are farthest apart. The cost of the shortest path between these vertices is 7, hence $c_{e_{\text{worst}}}(\bar{v}, \bar{v}') = 7$ in this example. We can formally write the worst case meta-

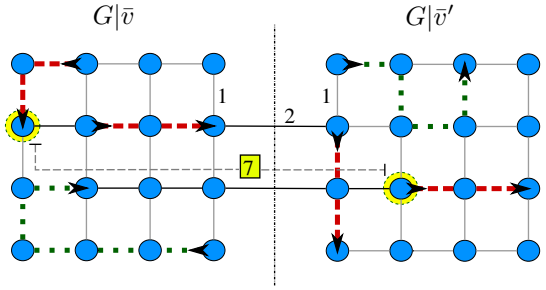

Fig. 2. Example showing the worst-case edge-cost between two meta-vertices. Edge-costs are 1 for edges within subgraphs, 2 for edges between subgraphs and all vertex costs are 0.

edge cost function for all $(\bar{v}, \bar{v}') \in \bar{E}$, as

$$c_{e_{\text{worst}}}(\bar{v}, \bar{v}') = \max_{\bar{o}, \bar{o}'} \max_{a, a'} d\left[p^*_a(\bar{v}, \bar{o}, f), p^*_{a'}(\bar{v}', \bar{o}', 1)\right], \tag{7}$$

where $\bar{o}, \bar{o}' \in \bar{O}$, $a \in \{1, \ldots, \bar{o}\}$, $a \in \{1, \ldots, \bar{o}'\}$, and $d[v, v']$ is the cost of the shortest path in G from v to v'. In implementation, there are ways to make this less conservative. For example, one could create a team edge-cost function that depends on the occupancies of adjacent vertices. However, for simplicity of notation, we choose an edge-cost on the location meta-graph that does not depend on the vertex occupancies.

Now let $\bar{\mathbf{G}}_{\text{worst}} := (\bar{\mathbf{V}}, \bar{\mathbf{E}})$ be the team meta-graph induced by $\bar{G}_{\text{worst}}$. The team path-cost and path-reward functions for the meta-graph are defined exactly the same as they were for the original graph in (2) and (3). The objective in the worst-case cooperative meta-graph search problem is to solve the cooperative search problem (4) on $\bar{G}_{\text{worst}}$ and thus find the reward $R^*_{\bar{G}_{\text{worst}}}(s, L)$.

B. Best-case cooperative meta-graph search

We construct the best-case problem such that its solution will be an upper bound on the optimal reward of (4). Let $\bar{G}_{\text{best}} := (\bar{V}, \bar{E})$ be a meta-graph of G with edge cost function defined by

$$c_{e_{\text{best}}}(\bar{v}, \bar{v}') = \min_{v \in \bar{v}, v' \in \bar{v}'} d[v, v'], \qquad \forall (\bar{v}, \bar{v}') \in \bar{E}. \tag{8}$$

where $d[v, v']$ is the cost of the shortest path in G from v to v' (for the example in Figure 2, $c_{e_{\text{best}}}(\bar{v}, \bar{v}') = 2$). Now, set $o_{\max} = 1$ and define the meta-vertex reward and cost functions as

$$r_{\text{best}}(\bar{v}, 1) := \sum_{v \in \bar{v}} \max_o r(v, o) \tag{9}$$

$$c_{v_{\text{best}}}(\bar{v}, 1) := \min_{v \in \bar{v}} c_v(v, 1). \tag{10}$$

Let $\bar{\mathbf{G}}_{\text{best}} := (\bar{\mathbf{V}}, \bar{\mathbf{E}})$ be the team meta-graph constructed from $\bar{G}_{\text{best}}$. The objective in the best-case cooperative meta-graph search problem is to solve the cooperative search problem (4) on $\bar{G}_{\text{best}}$ and thus find the reward $R^*_{\bar{G}_{\text{best}}}(s, L)$.

Theorem 1 *For every partition $\bar{V}$ of G*

$$R^*_{\bar{G}_{\text{worst}}}(s, L) \leq R^*_G(s, L) \leq R^*_{\bar{G}_{\text{best}}}(s, L) \tag{11}$$

The proof of the lower bound contains a procedure for generating an approximately optimal team search path on $\mathbf{G}$ whose total reward lies between $R^*_{\bar{G}_{\text{worst}}}(s, L)$ and $R^*_G(s, L)$.

Proof: To verify the lower bound, we use the optimal worst-case team path $\bar{\mathbf{p}}^* = (\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2, \ldots, \bar{\mathbf{v}}_f)$ on $\bar{\mathbf{G}}_{\text{worst}}$ to construct a feasible team path $\hat{\mathbf{p}}$ on $\mathbf{G}$ such that $\mathbf{C}(\hat{\mathbf{p}}) \leq L$. First, let us expand the team path to show the paths of each agent,

$$\bar{\mathbf{p}}^* = \begin{pmatrix} \bar{\mathbf{p}}^*_1 \\ \bar{\mathbf{p}}^*_2 \\ \vdots \\ \bar{\mathbf{p}}^*_s \end{pmatrix} = \begin{pmatrix} (\bar{\mathbf{v}}_1(1), \bar{\mathbf{v}}_2(1), \ldots, \bar{\mathbf{v}}_f(1)) \\ (\bar{\mathbf{v}}_1(2), \bar{\mathbf{v}}_2(2), \ldots, \bar{\mathbf{v}}_f(2)) \\ \vdots \\ (\bar{\mathbf{v}}_1(s), \bar{\mathbf{v}}_2(s), \ldots, \bar{\mathbf{v}}_f(s)) \end{pmatrix}$$

Now, we begin constructing the lower-level paths $\hat{\mathbf{p}}(a)$ by setting $\hat{\mathbf{p}} = \bar{\mathbf{p}}^*$ and then replacing the meta-vertices $\bar{\mathbf{v}}_i(a)$ with the paths computed by solving the worst-case cooperative search problem on the corresponding subgraphs $G|\bar{\mathbf{v}}_i(a)$. This involves grouping any agents occupying the same meta-vertex and assigning their paths based on the optimal team-path on that meta-vertex with the appropriate occupancy $\bar{o}$. For each agent a,

$$\hat{\mathbf{p}}_a = (\mathbf{p}^*_a(\bar{\mathbf{v}}_1(a), \bar{o}), \ldots, \mathbf{p}^*_a(\bar{\mathbf{v}}_2(a), \bar{o})), \ldots, \\ \ldots, \mathbf{p}^*_a(\bar{\mathbf{v}}_{f-1}(a), \bar{o})), \ldots, \mathbf{p}^*_a(\bar{\mathbf{v}}_f(a), \bar{o})),$$

Because of (6), the sum of the costs of these disconnected sub-paths in $\hat{\mathbf{p}}$ is equal to the sum of the vertex costs in $\bar{\mathbf{p}}^*$. Also, the total reward collected on these sub-paths is equal to the total reward collected on $\bar{\mathbf{p}}^*$ due to (5), so we know that $R^*_{\bar{G}_{\text{worst}}}(s, L) = \mathbf{R}(\bar{\mathbf{p}}^*) \leq \mathbf{R}(\hat{\mathbf{p}})$. We now fill in the gaps between consecutive sub-paths $\mathbf{p}^*_a(\bar{v}, \bar{o}))$ by connecting the last vertex in each previous sub-path to the first vertex in the next sub-path with the shortest path in G between them. We know from (7) that the cost of these connections is less than or equal to the edge costs in $\bar{\mathbf{p}}^*$. Hence, $C(\hat{\mathbf{p}}) \leq C(\bar{\mathbf{p}}^*) \leq L$ and $\hat{\mathbf{p}}$ is a feasible path in $\mathbf{G}$. Since $\mathbf{p}^*$ is optimal on $\mathbf{G}$, $\mathbf{R}(\hat{\mathbf{p}}) \leq R^*_G(s, L)$, and the lower bound in 11 holds.

To verify the upper bound, we construct a feasible path $\tilde{\mathbf{p}}$ in $\bar{\mathbf{G}}_{\text{best}}$ out of the optimal search path $\mathbf{p}^* = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_f)$ in $\mathbf{G}$ that generates reward $R^*_G(s, L)$. We begin by setting $\tilde{\mathbf{p}}$ equal to $\mathbf{p}^*$ and then replacing each $\mathbf{v}_i(a)$ with the $\bar{\mathbf{v}}_i(a)$ that contains it. Now, we remove any consecutive repetitions from each $\tilde{\mathbf{p}}_a$, and then pad the end of agent's paths with repeated meta-vertices where necessary to make the paths of all agents the same length. This allows us to form the team-vertices that make up $\tilde{\mathbf{p}}$, making a feasible path in $\bar{\mathbf{G}}_{\text{best}}$ without adding to the path-cost. We infer from (8) and (10) that $C(\tilde{\mathbf{p}}) \leq C(\mathbf{p}^*)$, so $\tilde{\mathbf{p}}$ meets the cost constraint. Finally, due to (9), all reward is collected from each meta-vertex visited by $\tilde{\mathbf{p}}$, and the upper bound $R^*_G(s, L) \leq R^*_{\bar{G}_{\text{best}}}(s, L)$ holds. ■

V. Computational Complexity

The computational complexity of the search problem for a single agent is known to be NP-Hard [1] on the number of vertices n. Reducing the s-agent cooperative search problem to a team search problem with n^s vertices results in a problem that is NP-Hard on n^s. In the worst case, an exhaustive search on a complete graph would have complexity $O(n^s!)$. Although there are some more efficient algorithms to solve this problem such as the branch and bound methods of Eagle and Yee [2], this problem is still computationally infeasible for large values of n^s.

For the rest of this section, we let $f(s, n, k)$ denote the computational complexity of an s-agent cooperative search problem on n vertices decomposed into k meta-vertices. Without decomposition, the complexity is $f(s, n, 1)$. Although there is some computation involved in setting up the problem, such as in partitioning and finding the shortest paths for the meta-edge costs in (7), this analysis assumes that n is large enough to render any overhead computation negligible compared to the search computation. Now, let us see what happens with one level of decomposition. Suppose that we partition the graph into k sub-graphs each containing roughly $\frac{n}{k}$ vertices. The computational complexity of the worst-case decomposed problem is then

$$f(s, n, k) = f(s, k, 1) + kf(s, \frac{n}{k}, 1), \tag{12}$$

where the first term comes from searching the meta-graph, and the second term comes from searching the k subgraphs. Decomposing on a second level yields

$$\begin{aligned} f(s, n, k_0) &= f(s, k_0, k_1) + k_0 f(s, \frac{n}{k_0}, k_2) \\ &= f(s, k_1, 1) + k_1 f(s, \frac{k_0}{k_1}, 1) \\ &\quad + k_0 f(s, k_2, 1) + k_0 k_2 f(s, \frac{n}{k_0 k_2}, 1). \end{aligned} \tag{13}$$

A choice of $k = \sqrt{n}$ in (12) makes the computation of the upper-level meta problem equal to that of the k subproblems.

For two levels, the analogous choices are $k_0 = \sqrt{n}$, and $k_1 = k_2 = \sqrt{k_0} = \sqrt[4]{n}$. Continuing the recursive decomposition in this manner gives the following computational complexity results:

$$\begin{aligned}
&\text{Level } 0: f(s,n,1)\\
&\text{Level } 1: (\sqrt{n}+1)f(s,\sqrt{n},1)\\
&\text{Level } 2: (\sqrt{n}+1)(\sqrt[4]{n}+1)f(s,\sqrt[4]{n},1)\\
&\qquad\vdots\\
&\text{Level L}: \sum_{i=0}^{2^L-1} n^{\frac{i}{2^L}} f(s,n^{\frac{1}{2^L}},1)\\
&\qquad = \left(\frac{n-1}{n^{\frac{1}{2^L}}-1}\right) f(s,n^{\frac{1}{2^L}},1).
\end{aligned}$$

The number of decomposition levels is limited to $L_{\max} = \lfloor \log_2(\log_2(n)) \rfloor$ because there is no advantage to decomposing a graph with only two vertices. It follows that as L approaches this maximum value, the computational complexity approaches $(n-1)f(s,2,1)$, which is equivalent to $O(n)$ since $f(s,2,1)$ is a constant. Hence, as the number of decomposition levels increases, the complexity approaches linearity.

VI. Test Results

We now apply the methods described above to a test case, simulating the search for an object in a large building with many rooms. Figure 3 shows a model of the third floor of Harold Frank Hall at UCSB, where a known initial probability distribution for the object is indicated by the shaded regions (dark represents high probability). The floor has been divided into 646 cells, each about 4 square meters in size. There is a graph vertex on each cell and pairs of vertices lying on adjacent cells are connected by an edge in the graph. We assign each edge a cost of 1, modeling a one second transit time between cells, and each vertex a cost of 2, supposing that it takes 2 seconds to search a cell.

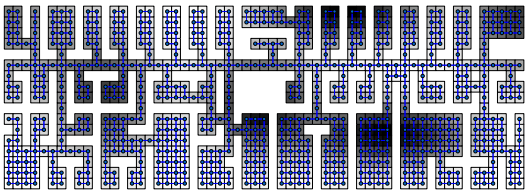

Fig. 3. Model of third floor of UCSB's Harold Frank Hall divided into 646 cells and overlaid with a graph. Dark cells indicate high target probability.

In this test case, we have 4 searchers and only two minutes to find the object. The goal is to find the path for each agent that approximately maximizes the probability of finding the object in 120 seconds. To get an idea of the magnitude of computation posed by this problem, consider a solution by total enumeration of feasible paths. The average degree of a vertex in this graph is about 3, meaning that the average degree of a vertex in the team graph is $3^4 = 81$, that is, there are about 81 possible moves for the team at each vertex. A cost bound of 120 allows the searchers to visit up to 40 vertices along their paths. This translates to roughly $81^{40} \approx 10^{76}$ paths that must be evaluated to find the optimal search path by total enumeration.

We now apply the fractal decomposition method to this problem. Using two levels of decomposition, we partition the top level into 7 groups and each of the lower-levels into 8, because there are roughly 56 rooms on the floor. We use the automated graph partitioning algorithm in [5], which tries to minimize the total cost of cut edges by clustering the eigenvectors of a (doubly stochastic) modification of the edge-cost matrix around the k most linearly independent eigenvectors. For our purposes, we define cost of cutting an edge between adjacent vertices with rewards r_1 and r_2 to be $e^{-|r_1-r_2|}$. This causes the algorithm to favor cutting edges with very different rewards, and thus grouping vertices with similar rewards. Figure 4 shows the top-level partition on our test graph, with vertices of the same color belonging to the same partitioned subgraph. Figure 5 shows the second-level partition applied to the subgraph in the upper left corner of Figure 4. The remaining subgraphs are similarly partitioned.

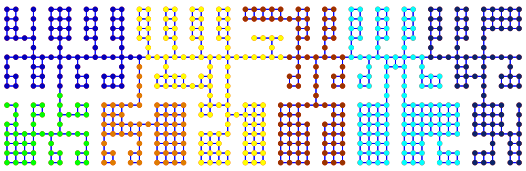

Fig. 4. Top-level partition on the graph into 7 subgraphs.

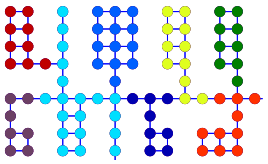

Fig. 5. The subgraph in the upper-left corner of Fig. 4 partitioned on a second level into 8 sub-subgraphs.

We choose a cost bound allocation of 25 seconds on each of the 56 lower-level subgraphs and 120 seconds on the 7 top-level subgraphs. The maximum vertex occupancy on all levels in set to 1. Now we are ready to run the algorithm. Figure 6 shows the approximately optimal paths computed for four searchers. The cost of these paths is 110 seconds, and the searchers collect a reward of 0.29, which lies between the worst-case lower bound of 0.26 and best-case upper bound of 1.0. Table I shows the results of the algorithm for one to four searchers. The fact that s searchers are able to collect almost s times the reward of 1 searcher shows that this algorithm achieves good cooperation between agents. In all four tests, the best-case upper bounds are equal to the total

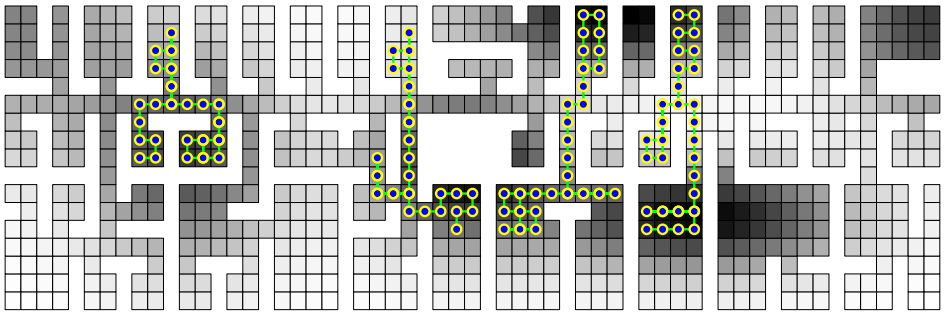

Fig. 6. Results of 4-agent cooperative search simulation.

TABLE I
RESULTS OF COOPERATIVE SEARCH FOR 1-4 AGENTS

Searchers	Cost	Reward	$R^*_{\bar{G}_{\text{worst}}}(s, 120)$	$R^*_{\bar{G}_{\text{best}}}(s, 120)$
1	107	0.081	0.072	1.0
2	107	0.15	0.14	1.0
3	110	0.22	0.20	1.0
4	110	0.29	0.26	1.0

reward contained in the graph. This is not ideal, but when using multiple decomposition levels, it is difficult to avoid a very optimistic upper bound unless the meta-edge costs are significantly larger than the meta-vertex costs. This is one issue for future research.

There is also some backtracking along the paths in Figure 6, some of which could be eliminated with a simple algorithm implemented in post-processing. Once this is done, there will be some unused cost available, and the path could be further improved with a greedy algorithm, for example.

Although the initial searcher positions were not fixed in this example, it is straight-forward to apply this algorithm to a problem where they are fixed, by preselecting the initial (meta-)vertices in the top-level search paths as well as those for paths on any subgraphs where the searchers are initially located.

VII. CONCLUSIONS AND FUTURE WORK

We introduced a cooperative graph search algorithm that is able to compute approximately optimal paths on very large graphs by recursively decomposing the problem into smaller problems of exactly the same form. Solutions to worst-case and best-case versions of the decomposed problems provide lower and upper bounds on the optimal reward. With a simulated search on a floor of a large university building, we showed that the algorithm presented is computationally fast, yields good results, and achieves true cooperation between searchers.

There are several potential directions for future work on this algorithm, three of which are listed below.

- Increase tightness of worst-case bound by allowing partial reward to be collected from a vertex while also incurring only a portion of the cost. Also, it would be useful to obtain a less optimistic upper bound, possibly by exploiting the structure of the graph in some way.
- Generate a distributed version of this algorithm by having each of the agents keep estimates on the locations of its teammates.
- Generalize the algorithm to the moving target case, a somewhat more difficult problem because the target probability distribution is constantly changing as time passes and new information is collected.

REFERENCES

[1] B. DasGupta, J. Hespanha, J. Riehl, and E. Sontag. Honey-pot constrained searching with local sensory information. *Nonlinear Analysis: Hybrid Systems and Applications*, 65(9):1773–1793, Nov. 2006.

[2] J. Eagle and J. Yee. An optimal branch-and-bound procedure for the constrained path, moving target search problem. *Operations Research*, 28(1), 1990.

[3] J. R. Frost and L. D. Stone. Review of search theory: Advances and applications to search and rescue decision support. Technical report, U.S. Coast Guard Research and Development Center, Groton, CT, Sept. 2001.

[4] K. B. Haley and L. D. Stone, editors. *Search Theory and Applications*. Plenum Press, New York.

[5] J. P. Hespanha. An efficient matlab algorithm for graph partitioning. Technical report, University of California, Santa Barbara, CA, Oct. 2004. Available at `http://www.ece.ucsb.edu/~hespanha/techreps.html`.

[6] B. O. Koopman. *Search and Screening*. Operations Evaluations Group Report No. 56, Center for Naval Analyses, Alexandria, VA, 1946.

[7] B. O. Koopman. *Search and Screening: General Principles and Historical Applications*. Pergamon Press, New York, 1980.

[8] M. Mangel. Search theory: A differential equations approach. In D. V. Chudnovsky and G. V. Chudnovsky, editors, *Search Theory: Some Recent Developments*, pages 55–101. Marcel Dekker, New York, 1989.

[9] M. M. Polycarpou, Y. Yang, and K. M. Passino. A cooperative search framework for distributed agents. In *Proc. of the IEEE International Symposium on Intelligent Control*, 2001.

[10] J. R. Riehl and J. P. Hespanha. Fractal graph optimization algorithms. In *Proceedings of the 44th Conference on Decision and Control*, 2005.

[11] T. J. Stewart. Experience with a branch and bound algorithm for constrained searcher motion. In K. B. Haley and L. D. Stone, editors, *Search Theory and Applications*, pages 247–254. Plenum Press, New York, 1980.

[12] L. D. Stone. *Theory of Optimal Search*. Academic Press, New York, 1975.

[13] K. E. Trummel and J. R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.