Cooperative Graph-Based Model Predictive Search*

James R. Riehl†, Gaemus E. Collins‡, and João P. Hespanha†

Abstract— **We present a receding-horizon cooperative search algorithm that jointly optimizes routes and sensor orientations for a team of autonomous agents searching for a mobile target. By sampling the region of interest at locations with high target probability, we reduce the continuous search problem to an optimization on a finite graph. Paths are computed on this graph using a receding horizon approach, in which the horizon is a fixed number of waypoints. To facilitate a fair comparison between paths of varying length on a non-uniform graph, we use an optimization criterion corresponding to the probability of finding the target per unit time. Using this algorithm, we show that the team discovers the target in finite time with probability one. Simulations verify that this algorithm makes effective use of agents and performs significantly better than previously proposed search algorithms. We have also successfully tested this search algorithm on a physical system consisting of two UAVs with gimbal-mounted cameras.**

I. Introduction

Consider a search problem in which a cooperating team of agents is searching for a target in a bounded region. Each agent is equipped with a gimballed sensor that it can aim at a limited area within the search region. As they move around and take sensor measurements, the agents gather information on the likely locations of the target. The objective of each agent is to move itself and control its sensor in a way that minimizes the expected time for the team to discover the target. In this paper, we introduce a cooperative graph-based model predictive search (CGBMPS) algorithm to approximately solve this optimization problem.

Early search theory, developed by Koopman [5], Stone [10], and others, focused on the allocation of search effort to specific areas within the overall search region, which is appropriate to special cases in which 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. Mangel formulated the continuous search problem in [6] as an optimal control problem on the searcher's velocity subject to a constraint in the form of a partial differential equation, but this problem is only solvable for very simple initial target probability distributions. A more practical approach is to discretize the search space and formulate the search problem on a graph. Under such a formulation, the set of search paths is restricted to piecewise linear paths connecting a finite set of graph vertices. Although this discretization simplifies the problem to some extent, it is still computationally difficult. Trummel and Weisinger showed that the single-agent search problem on a graph is NP-Hard even for a stationary target [11]. Eagle and Yee [2] were able to solve somewhat larger problems than what had previously been possible by formulating the problem as a nonlinear integer program and using a branch and bound algorithm to solve it. However, the size of computationally feasible problems was still severely limited.

The problem gains additional complexity when we consider multiple agents that are cooperatively searching for a target. For this reason, literature on cooperative search generally seeks approximate solutions. A natural way to reduce the computation involved in finding optimal search paths to locate a moving target is to restrict the optimization to a finite, receding horizon. Somewhat surprisingly, even very short optimization horizons can yield good results. Hespanha *et al.* [3] showed that in pursuit-evasion games played on a discrete grid, one step Nash policies result in finite-time evader capture with probability one. Using slightly longer horizons, Polycarpou *et al.* introduced a finite-horizon look-ahead approach similar to model predictive control as part of a cooperative map learning framework [8]. A challenge faced by these receding horizon algorithms is that because they require the solution to an NP-hard search problem at each time step, the prediction horizon must be kept short to ensure computational feasibility. This means that if there are regions of high target probability separated by a distance that is much greater than an agent can cover on this horizon, the algorithm's performance will suffer. One does not have to produce a pathological example to generate such a situation, because even a uniform initial distribution can evolve into a highly irregular distribution as the searchers move through the region and update their estimates based on sensor measurements.

With the CGBMPS algorithm, we address this issue by performing the receding-horizon optimization on a dynamically changing graph whose nodes are carefully placed at locations in the search region having the highest target probability. The prediction horizon is defined as a fixed number of waypoints, and since the edges of the graph are not uniform, the algorithm chooses paths that maximize the probability of finding the target per unit cost. This dynamic graph structure facilitates two key strengths of our algorithm: (i) search agents only perform detailed searches in regions with high target probability, and (ii) long paths can be efficiently evaluated and fairly compared to short paths. We present a centralized version of the algorithm in this

*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, and by the Air Force Office of Scientific Research through grant FA9550-06-C-0119.

†{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.

‡gcollins@toyon.com, Toyon Research Corporation, 6800 Cortona Drive, Goleta, CA 93117.

paper, but suggest in Section VI how one could implement a decentralized version.

The remainder of the paper is organized as follows. Section II presents the cooperative search problem in a general discrete time setting. Section III describes our approach to this problem in the form of a receding horizon search algorithm on a graph. In Section IV, we present the main results on finite-time target discovery. Section V provides the numerical simulations of the algorithm and discusses the results. In Section VI, we provide conclusions and some ideas for future research.

II. Search Problem Formulation

Suppose a team of M agents is searching for a mobile target in a bounded planar region $\mathcal{R} \subseteq \mathbb{R}^2$. Each agent is equipped with a gimballed sensor that it can aim at a limited area of the search region. The region that a sensor can view at a particular time instant is called the *field of view (FOV)*, and the subset of $\mathcal{R}$ viewable by the sensor as it is swept through its entire range of motion (while the agent on which the sensor lies is stationary) is called the sensor's *field of regard (FOR)* (See Figure 1).

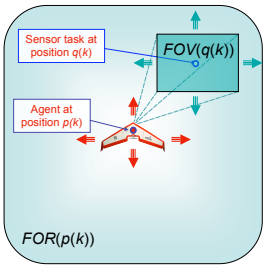


Fig. 1. Diagram of FOR and FOV.

Let $\mathcal{A} \subseteq \mathbb{R}^3$ be the space in which the agents move. We denote the state of the search team by $\mathbf{p}(k) := [p_1(k), p_2(k), \ldots, p_M(k)]$, where $p_a(k) \in \mathcal{A}$ is agent a's position at time k, and $k \in \mathbb{Z}_{\geq 0}$ is a discrete time variable belonging to the nonnegative integers. We assume the agents can move in any direction with unit velocity. For each agent, we define a *sensor task* $q_a(k) \in \mathcal{R}$ that specifies where agent a will point its sensor at time k, and we denote the vector of sensor tasks for the team by $\mathbf{q}(k) := [q_1(k), q_2(k), \ldots, q_M(k)]$. For simplicity of notation, we assume that the point $q(k)$ at which a sensor is aimed uniquely determines the sensor's field of view. Denoting the set of all possible sensor tasks by $\mathcal{Q}$, and the set of all subsets of $\mathcal{R}$ by $\mathcal{P}(\mathcal{R})$, we define a function $FOV : \mathcal{Q} \to \mathcal{P}(\mathcal{R})$ that maps a point at which a sensor is aimed to the subset of $\mathcal{R}$ in view of that sensor. Similarly, we will use the function $FOR : \mathcal{A} \to \mathcal{P}(\mathcal{R})$ to denote the subset of the search region lying in the field of regard of an agent.

In order to optimize how the agents and sensors should move, we need a mechanism to estimate the probability distribution of the target's location in $\mathcal{R}$. Specific estimation methods are treated in references [1] and [4]. However, to provide a more general approach, we use the concept of a *target state estimate*, denoted by $\mathbf{x}(k)$, which consists of position and weight components $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$. Each of these two components is an n-vector. The target state estimate may take the form of a particle filter, grid-based probabilistic map, or some other type of estimator. The key requirements we enforce on $\mathbf{x}$ are

$$x_i^p(k) \in \mathcal{R} \qquad \forall i \in \{1, \ldots, n\}, \forall k, \tag{1}$$

$$x_i^w(k) \geq 0 \qquad \forall i \in \{1, \ldots, n\}, \forall k, \tag{2}$$

$$\sum_{i=1}^{n} x_i^w(k) = 1 \qquad \forall k. \tag{3}$$

Condition (1) ensures that the target state estimate stays inside the search region, condition (2) requires the weights to be nonnegative, and condition (3) requires the weights to be normalized at each time step. The initial target state estimate $\mathbf{x}(0)$ is based on any prior knowledge of target location, *e.g.* an initial probability distribution. Future target state estimates depend on sensor tasks $\mathbf{q}(k)$ and previous target state estimates, and can be expressed by a dynamic model of the following form:

$$\mathbf{x}(k+1) = f\big(\mathbf{x}(k), \mathbf{q}(k)\big), \tag{4}$$

where the function f should preserve conditions (1)-(3). In Section II-A, we show what the dynamics of (4) are for a particle filter and the reader is referred to [9] for the dynamics that arise in probabilistic mapping.

We denote by $r(k)$ an instantaneous *team search reward* that effectively measures the probability that the team will discover the target at time k. We say that the target has been *discovered* if it lies within the field of view of at least one agent and one of those agents detects its presence. The initial value for the team search reward is $r(0) = 0$, and future values depend on the target state estimate $\mathbf{x}(k)$, the sensor tasks $\mathbf{q}(k)$, and previous rewards according to an expression of the following form:

$$r(k+1) = g\big(\mathbf{x}(k), \mathbf{q}(k)\big)\Big(1 - \sum_{\kappa=0}^{k} r(\kappa)\Big), \tag{5}$$

where the value of the function g corresponds to the conditional probability that the target will be found at time $k+1$, given that it was not previously found. The term multiplying $g(\cdot)$ corresponds to the probability that the target was not found at or before time k. Therefore, $r(k+1)$ indeed measures the probability that the target will be found at time $k+1$.

It will be convenient to have an expression for the total reward collected by the team as a summation of the rewards collected by each individual agent. With this in mind, we rewrite r as

$$r(k+1) = \sum_{a=1}^{M} r_a(k+1),$$

where r_a is an instantaneous *agent search reward*, which we can express by

$$r_a(k+1) = g_a\big(\mathbf{x}(k), \mathbf{q}(k)\big)\Big(1 - \sum_{\kappa=0}^{k} r(\kappa)\Big), \tag{6}$$

for each agent $a \in \{1, \ldots, M\}$. Here, the function g_a measures the conditional probability that agent a will find the target at time $k{+}1$ given that it was not previously found. A summation over the g_a's yields the function g used in (5): $g(\mathbf{x}, \mathbf{q}) = \sum_{a=1}^{M} g_a(\mathbf{x}, \mathbf{q})$.

Using the target weights $\mathbf{x}^w$, we can explicitly write the function g_a as

$$g_a(\mathbf{x}, \mathbf{q}) := \sum_{i=1}^{n} x_i^w \rho_a(x_i^p, \mathbf{q}),$$

where $\rho_a(x, \mathbf{q})$ is an *agent reward factor*, which measures the conditional probability that, when the team's sensors are aimed at the set of points $\mathbf{q}$, agent a will find the target at the point x given that the target is located at x. Assuming that each agent correctly detects a target located inside its sensor's FOV with probability $P_D \in (0, 1]$, we can express the agent reward factor as

$$\rho_a(x, \mathbf{q}) := \delta_v(x, q_a) P_D \prod_{\alpha=1}^{a-1} \big(1 - \delta_v(x, q_\alpha) P_D\big), \quad (7)$$

where the function $\delta_v(x, q)$ indicates whether a particular point x is in view of a sensor aimed at the point q, and is given by

$$\delta_v(x, q) := \begin{cases} 1, & x \in FOV(q) \\ 0, & \text{otherwise} \end{cases}.$$

Although we have introduced in (7) a specific ordering to the agents for the purposes of computation, one can show that changing this order does not affect the value of g.

A. Example: Particle Filter

Particle filtering is a Monte Carlo-based method for state estimation that is especially well-suited to systems with non-linear dynamics and non-Gaussian distributions. It involves modeling a system with a large number of dynamic "particles" whose states evolve according to some stochastic model. Each particle is assigned a weight representing the likelihood that the actual state is near that particle. Weights are typically updated upon the arrival of new measurements and are then generally normalized so that the total weight of the particles is equal to one. See [1] for a more detailed treatment of particle filtering. We now give an example of how to implement (4) using particle filter estimation.

Let $\mathbf{x}(k) := [\mathbf{x}^p(k), \mathbf{x}^w(k)]$ represent the states of a simple particle filter, where each $x_i^p(k)$ is the position of the i^{th} particle and $x_i^w(k)$ its weight. The system dynamics are then given by

$$\tilde{x}_i^w(k{+}1) = x_i^w(k) \prod_{a=1}^{M} \big(1 - \delta_v(x_i^p(k), q_a(k)) P_D\big) \quad \forall i,$$

$$\mathbf{x}^w(k{+}1) = \tilde{x}^w(k{+}1) \frac{1}{\sum_{i=1}^{n} \tilde{x}_i^w(k{+}1)},$$

$$x_i^p(k{+}1) = f_p\big(x_i^p(k), w_i(k)\big),$$

where $\tilde{\mathbf{x}}^w$ is an intermediate vector of unnormalized target weights, and $\mathbf{w}(k) := [w_1(k), \ldots, w_n(k)]$ is a process noise sequence. The function f_p is typically used to propagate the particle positions according to a model of the expected target dynamics, randomized with $\mathbf{w}(k)$, which for our purposes is fixed for the duration of the search and known by all agents.

B. Problem Statement

We define a *search policy* to be a function μ that maps the current team state $\mathbf{p}(k)$ and target state estimate $\mathbf{x}(k)$ to the next set of controls to be executed by the team, that is, each agent's next sensor task is given by $q_a(k) = \mu(\mathbf{p}(k), \mathbf{x}(k))$. The goal of the search problem is to find the search policy that results in the fastest possible discovery of the target by the team.

Let T^* denote the time at which the target is discovered by any agent. This time is unknown at the beginning of the search, but we can write its expected value as

$$E_\mu[T^*] = \sum_{k=0}^{\infty} k r(k). \quad (8)$$

We can now express the objective of the search problem as

$$\min_{\mu} E_\mu[T^*] \quad (9)$$

subject to

$$FOV\big(q_a(k)\big) \subseteq FOR\big(p_a(k)\big) \qquad \forall a, k. \quad (10)$$

The constraint (10) requires each agent's sensor task to be feasible with respect to its current field of regard. This constraint is needed because of the physical constraints of the gimballed sensors.

For agents with such sensor and motion constraints, the problem posed in (9) is a difficult combinatorial optimization problem. Even for the case of a single agent, stationary target, and fixed sensor, this search problem is known to be NP-Hard [11]. Our approach consists of approximating the solution by choosing paths on a graph that maximize the reward collected over a finite, receding horizon. This method is designed to reduce computation by optimizing over a shorter time horizon on a much smaller state space, while also using careful vertex placement to maintain route flexibility in regions of high reward. Although our algorithm does not necessarily result in a policy that minimizes $E[T^*]$, the resulting policy does lead to a finite value for this expected time and we provide an upper bound for this quantity in Section IV.

III. Cooperative Graph-based Model Predictive Search Algorithm

In this section, we present a receding horizon search algorithm that plans paths and sensor tasks for a team of agents based on predicted future target states. We use a prediction horizon based on waypoints rather than time because we would like to consider paths of varying duration. This allows agents to travel between high-reward regions separated by large areas with low reward.

A. Graph Search Formulation

Let $G := (V, E)$ be a graph for path-planning with vertex set V and edge set E. The vertices of the graph will serve as waypoints for the agents, which are connected by graph edges to form paths. When there is no ambiguity, we will use the notation v to denote both the vertex in the set V and its spatial location in $\mathcal{A}$. Similarly, $e = (v, v')$ will denote both the edge in the set E and the set of points on the line segment connecting its endpoint vertices v and v'. Lastly, we define an edge-cost function $c : E \to [0, \infty)$, representing the transit time between vertices, assuming the agents move with unit velocity.

In the CGBMPS algorithm, whenever an agent reaches a waypoint, it will compute a new ℓ-step path on the graph, where ℓ is the length of the prediction horizon. A *path* in G is a sequence of vertices $P := (v_1, v_2, \ldots, v_{\ell-1}, v_\ell)$ such that $(v_i, v_{i+1}) \in E$. The *path cost* is the total duration of the path and is given by

$$C(P) := \sum_{i=1}^{\ell-1} c(v_i, v_{i+1}).$$

The reward for a path P will depend on the expected sensor coverage along that path.

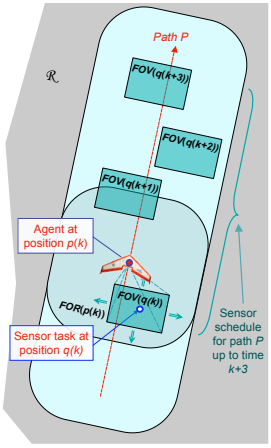


Fig. 2. Example sensor schedule for an agent.

In an attempt to minimize the expected time $E[T^*]$ to find the target, each agent will select the path from the set of all possible ℓ-step paths, that maximizes the team's probability of finding the target. We will shortly define a *path reward* corresponding to this probability, and a *path cost* corresponding to the duration of a path. In order to compare paths of varying length, the algorithm uses an optimization criterion based on path reward per unit cost. As each agent travels along its path P, it executes a sequence of sensor tasks, which we call a *sensor schedule* and denote by $S(P, k) := \big(q(k), q(k+1), \ldots, q(k+T_P)\big)$, where each $q(\cdot)$ is a scheduled sensor task and $T_P := \lfloor C(P) \rfloor$ is the total number of sensor tasks that may be executed along the path P. In this discrete time setting, we assume that the duration of each sensor task is one unit of time. Figure 2 shows a diagram of an agent executing a sensor schedule along its path P. The sensor tasks $q(k)$ may be chosen arbitrarily or computed by another optimization. This allows for several possible methods of algorithm implementation, including (i) *fixed sensor tasking*, which constructs $S(P, k)$ assuming a fixed pattern of sensor motion, such as slewing side to side within the field of regard; and (ii) *joint routing and sensor optimization*, which constructs $S(P, k)$ by optimizing over all possible sensor schedules along each edge in the path P). Fixed sensor tasking requires much less computation, but joint routing and sensor optimization will generally yield better results. We will revisit these methods in Section V, but for the remainder of this section, we will assume that we are given an algorithm to compute the sensor schedule $S(P, k)$ for a given path P.

For a path P_a, we define the *path reward* for agent a as

$$R_a(P_1, \ldots, P_a) := \sum_{\kappa=k}^{k+T_{P_a}} r_a(\kappa), \tag{11}$$

where $r_a(\kappa)$ is computed using (6) with the sensor tasks $q_i(\kappa)$ from the sensor schedules $S(P_i, k)$ obtained from the paths P_i of agents $i \in \{1, \ldots, a\}$. Note that the paths P_i of agents $i \in \{a+1, \ldots, M\}$ do not affect (11). To provide a fair measure of reward over paths of different lengths, we define the *normalized path reward* for the path P_a as

$$\bar{R}_a(P_1, \ldots, P_a) := \frac{R_a(P_1, \ldots, P_a)}{C(P_a)}. \tag{12}$$

As mentioned previously, each agent computes a new path whenever it reaches a waypoint. Since the graph edges have nonuniform length, the agents will generally arrive at waypoints, and hence compute paths, at different times. At each time k, we therefore define two groups of agents: the set $A_{\text{plan}}(k)$ of agents that have arrived at waypoints and need to plan new paths, and the set $A_{\text{en}}(k)$ of the remaining agents that are en route to waypoints. For simplicity of notation, we assume that the indexing of agents $\{1, \ldots, M\}$ is given by the sequence $\{A_{\text{en}}(k), A_{\text{plan}}(k)\}$ at each time k. In the CGBMPS algorithm, whenever the set $A_{\text{plan}}(k)$ is nonempty, the agents in that set compute new paths assuming that the agents in the set $A_{\text{en}}(k)$ will continue on their current paths. With this formulation, we can express each step in the receding horizon optimization as follows:

$$\{P_a^* : a \in A_{\text{plan}}(k)\} := \underset{\{P_a : a \in A_{\text{plan}}(k)\}}{\arg\max} \sum_{i=1}^{M} \bar{R}_i(P_1, \ldots, P_i). \tag{13}$$

Often, the set $A_{\text{plan}}(k)$ consists of a single agent, reducing (13) to an optimization for the path of that agent. If this is not the case, a computationally practical approximation to (13) consists of a sequential optimization by the planning agents, which we can express as

$$P_a^* := \underset{P_a}{\arg\max}\, \bar{R}_a(P_1^*, \ldots, P_{a-1}^*, P_a), \forall a \in A_{\text{plan}}(k), \tag{14}$$

where each P_i^* is either the current path of an en route agent $i \in A_{\text{en}}(k)$ that was computed at a previous time or a path

computed using (14) by an agent earlier in the sequence $A_{\text{plan}}(k)$.

B. Dynamic Graph Generation

Although the CGBMPS algorithm will work with any graph, the structure of the graph will have a major effect on computation, and the vertex and edge placements will affect the algorithm's performance. Also, since the target state estimate is constantly changing due to sensor measurements and predicted target motion, the graph should be periodically updated to allow the agents to search new high-reward areas that were low-reward areas in previous time steps. This graph update should occur at each time k at which the set of agents $A_{\text{plan}}(k)$ is nonempty. We now present a dynamic graph construction process that will guarantee some performance properties of the CGBMPS algorithm. Figure 3 shows an example of this process.

In the following procedure, we will use the function $W(\mathbf{x}, e)$ to denote the sum of the weights of all components of the target state estimate lying inside the FOR of some point on the edge e. We define this as

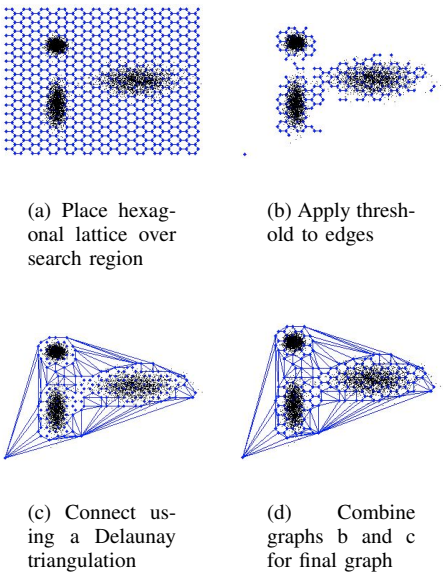
$$W(\mathbf{x}, e) := \sum_{i=1}^{n} x_i^w \delta_r(x_i^p, e),$$

where $\delta_r(x, e) \in \{0, 1\}$ indicates whether or not the point x_i^p is inside the FOR of at least one point on the edge e, and is expressed by

$$\delta_r(x, e) := \begin{cases} 1, & x \in \bigcup_{p \in e} FOR(p) \\ 0, & \text{otherwise} \end{cases}.$$

Graph Construction Process

1) Construct a uniform lattice graph $G := (V, E)$ over the search region $\mathcal{R}$ having the property that for every point r in $\mathcal{R}$, there exists an edge $e \in E$ such that the point r falls within the FOR of some point along the edge e. This can always be achieved by choosing a sufficiently small vertex spacing in the lattice [cf. Figure 3(a)].
2) Choose a reward threshold $\tau > 0$ and let $G_k^{\text{th}} := (V_k^{\text{th}}, E_k^{\text{th}})$ be the subgraph of G induced by the edge set $E_k^{\text{th}} := \{e \in E : W\big(\mathbf{x}(k), e\big) \geq \tau\}$. The vertex set V_k^{th} consists of the union of the set of vertices connected by edges E_k^{th} with the set of vertices that serve as initial waypoints v_1^a for the agents [cf. Figure 3(b)].
3) Let $G_k^{\text{Del}} := (V_k^{\text{th}}, E_k^{\text{Del}})$ be the graph generated by a Delaunay triangulation of V_k^{th} [7]. Next, let $G_k^{\text{Del}<d} := (V_k^{\text{th}}, E_k^{\text{Del}<d})$ be the subgraph of G_k^{Del} obtained by keeping only the edges connecting vertices of degree less than d, where d is the degree of the lattice graph G. That is $E_k^{\text{Del}<d} := \{(v, v') \in E_k^{\text{Del}} : \deg(v) < d \text{ and } \deg(v') < d\}$ [cf. Figure 3(c)].
4) The final graph $G_k := (V_k, E_k)$ is the combination of graphs G_k^{th} and $G_k^{\text{Del}<d}$, with vertex set $V_k := V_k^{\text{th}}$ and edge set $E_k := E_k^{\text{th}} \cup E_k^{\text{Del}<d}$ [cf. Figure 3(d)].



(a) Place hexagonal lattice over search region

(b) Apply threshold to edges

(c) Connect using a Delaunay triangulation

(d) Combine graphs b and c for final graph

Fig. 3. Example of graph construction process.

In step 1, any type of lattice graph will suffice, but we have found that a degree-3 hexagonal lattice is a particularly good choice. The threshold τ in step 2 should be carefully chosen so that it includes only high-reward edges but does not exclude so many edges that the number of feasible paths is severely restricted. The purpose of step 3 is to ensure that the graph G_k is connected and provides paths between separated areas of high reward. We include only the Delaunay edges connecting vertices of degree less than 3 because these are the edges between boundary vertices of the disconnected components in the graph. The remaining Delaunay edges are unnecessary. Step 4 combines the graphs of steps 2 and 3 to produce the final graph.

The following assumption requires that the total target weight within the FOR of an edge cannot drop to zero instantaneously.

Assumption 1: For every edge $e \in E_k$, there exists a constant $\gamma > 0$ such that for every $m \in \mathbb{N}$, the evolution of the target state estimate $\mathbf{x}(k)$ governed by (4) *satisfies the property*

$$W\big(\mathbf{x}(k+m), e\big) \geq \gamma^m W\big(\mathbf{x}(k), e\big).$$

Under this assumption, Lemma 1 provides a lower bound on the conditional probability that the target will be found at some time in the interval $[k+1, k+T]$ given that it was not found at or before time k.

Lemma 1: There exists a constant $\epsilon > 0$ and a time $T > 0$ such that for every graph G_k generated by the Graph Construction Process and every set of ℓ-length paths $P_1, \ldots, P_M$ in G_k, there exists a sequence of sensor tasks $(\mathbf{q}(k), \ldots, \mathbf{q}(k+T-1))$ for which $\sum_{\kappa=k}^{k+T-1} g\big(\mathbf{x}(\kappa), \mathbf{q}(\kappa)\big) \geq \epsilon$.

Lemma 1 is proved in [9]. The value for ϵ is

$$\epsilon = \gamma^T \tau P_D (A_{\text{FOV}}/A_{\text{FOR}}),$$

where $A_{\rm FOV}$ and $A_{\rm FOR}$ are the areas of the FOV and the FOR, respectively. The time T corresponds to the maximum possible edge cost and is given by $T := \lfloor D_{\max} \rfloor$, where $D_{\max}$ denotes the maximum Euclidean distance between any two vertices in G_k.

C. CGBMPS Algorithm

The Cooperative Graph-Based Model Predictive Search algorithm is described in Table I. It begins with initialization of the graph and each agent's path and sensor schedule in steps 2 and 3. At each subsequent time step k, whenever the set of agents $A_{\rm plan}(k)$ is nonempty, the graph is updated, and the agents in $A_{\rm plan}(k)$ select new paths and sensor schedules according to (13). This process repeats until the target is found at time T^*, which is proven to be finite with probability one in Theorem 1.

TABLE I
COOPERATIVE GRAPH-BASE MODEL PREDICTIVE SEARCH ALGORITHM

Algorithm 1 CGBMPS

1: $k := 0$
2: Construct $G_0 = (V_0, E_0)$ as in Section III-B
3: For each agent $a \in \{1, \dots, M\}$, assign an arbitrary initial path $P_a = (v_1^a, v_2^a)$ on G_0 and a corresponding sensor schedule $S(P_a, 0)$
4: **while** target has not been discovered **do**
5: Each agent a points its sensor at $q_a(k)$ and takes measurements
6: Compute the set of agents arriving at waypoints $A_{\rm plan}(k)$
7: **if** the set $A_{\rm plan}(k)$ is nonempty **then**
8: Update G_k as in Section III-B
9: **for** each agent a in the set $A_{\rm plan}(k)$ **do**
10: Compute the path P_a^* starting from v_2^a using (13)
11: Assign a sensor schedule $S(P_a^*, k)$
12: **end for**
13: **end if**
14: Agents move toward next waypoint in path
15: Evaluate $\mathbf{x}(k{+}1)$ and $r(k{+}1)$ using (4) and (5)
16: $k := k{+}1$
17: **end while**
18: $T^* := k{-}1$

Whenever a path is computed, it starts from the waypoint after the one that was reached. There are two key reasons for implementing the algorithm in this way. First, this allows time for computation of the next path, avoiding situations where the agent has reached a waypoint and does not have any destination until it completes a computation. Second, it is practical to have one waypoint planned in advance so that sharp turns in the upcoming path may be smoothed.

D. Computational Complexity

The CGBMPS algorithm provides a computationally efficient method for approximating the original search problem posed in (9) by using a graph that is designed to allow agents to optimize over a set containing only the most rewarding paths. Let K_e denote the maximum time required to compute the reward collected along an edge e in the graph. For example, K_e may represent an upper bound on the computation time required by *fixed sensor tasking* or *joint routing and sensor optimization*, as described in Section III-A. Performing an exhaustive search over this set of paths results in a bound on the computation time that depends on the maximum degree of the graph d, the length of the prediction horizon ℓ, and the number of agents M, by the expression $M!K_e\ell d^\ell$. Note that this is a worst-case computation bound because in a non-uniform graph, it is quite rare that all agents will arrive at waypoints simultaneously and hence compute new paths all at the same time. The factor $M!$ corresponds to the optimization given in (13), in which an exhaustive search is performed over every possible order of agents. If one opts to use the computationally simpler, sequential approximation (14) instead, the expression becomes $MK_e\ell d^\ell$. Additionally, in the computation of the optimal path P_a^*, we can take advantage of the property that the reward for the paths (v_1, v_2, v_3) and (v_1, v_2, v_4) is the same between vertices v_1 and v_2. The fact that one only needs to compute the reward for the segment (v_1, v_2) once, further reduces the total computation required for this algorithm. This results in the slightly improved bound of $MK_e \sum_{i=1}^{\ell} d^i$ to compute paths for the team of agents. It is clearly computationally advantageous to keep the degree d of the graph low and the prediction horizon ℓ short. One can also reduce the total number paths to evaluate by doing some reasonable pruning on the set of candidate paths $\mathcal{P}_a$, such as eliminating backtracking and paths with sharp turns that the agents cannot physically follow. Here, we have shown computation results for an exhaustive search over the prediction horizon, but it should be noted that algorithms such as branch and bound [2] may further reduce complexity.

IV. PERSISTENT SEARCH

In this section we show that the CGBMPS algorithm results in finite-time target discovery with probability one. We use the notion of a *persistent search policy*, which is inspired by the persistent pursuit policies discussed in [3].

A search policy is said to be *persistent on the average* if there is some $\epsilon > 0$ and some $T \in \mathbb{N}$ such that

$$\sum_{i=0}^{T-1} g\big(\mathbf{x}(k{+}i), \mathbf{q}(k{+}i)\big) > \epsilon \qquad \forall k \in \mathbb{Z}_{\geq 0}, \tag{15}$$

where the sensor tasks $\mathbf{q}(k)$ are generated by the search policy μ. The time T is called the period of persistence. The following lemma is proved in [3].

Lemma 2: For a persistent on the average search policy μ with period T, $\mathbf{P}(T^* < \infty \,|\, \mu) = 1$, and $E_\mu[T^*] \leq T\epsilon^{-1}$, with ϵ given in (15)*.*

We now state the main result on finite-time target discovery, which proves that the CGBMPS algorithm terminates with probability one.

Theorem 1: The CGBMPS algorithm results in target discovery in finite time with probability one and

$$E[T^*] \leq \frac{D_{\max}}{\epsilon},$$

where $\epsilon = \gamma^T \tau P_D (A_{\rm FOV}/A_{\rm FOR})$.

From Lemma 1, we conclude that the search policy generated by the CGBMPS algorithm, using optimal sensor schedules,

is persistent on the average. Theorem 1 then follows directly from Lemma 2 with ϵ as in Lemma 1.

V. Simulations

We now simulate a cooperative search scenario with four agents searching for a target in a 5 km by 5 km square region. The FOR of each agent is a circle 600 m in diameter and its FOV is a circle 150 m in diameter. The agents take measurements every 10 seconds. These parameter values are inspired by a physical system with cameras mounted on UAVs that has been successfully tested by our team. The initial target pdf consists of a weighted sum of four randomly placed Gaussian distributions with random covariances, and the target state estimate is constructed using a particle filter as described in Section II-A. For path planning, we use a degree-3 hexagonal lattice graph, which is dynamically updated using the process described in Section 3. The agents use a three-step prediction horizon.

TABLE II
Results of Cooperative Search for 1-4 Agents

Number of Agents (M)	Mean Reward (500 Runs)
1	0.22
2	0.39
3	0.53
4	0.64

Table II shows the results of the CGBMPS algorithm for one to four cooperating agents. The fact that the reward increases significantly with each increase in the number of searchers shows that this algorithm makes effective use of additional agents.

TABLE III
Algorithm Performance Comparison

Method (4 agents)	Mean Reward (300 Runs)
Greedy search	0.69
3-step fixed RH Search	0.72
CGBMPS (fixed sensor)	0.81
CGBMPS (joint sensor optimization)	0.87

In Table III, we compare the performance of the CGBMPS algorithm with previously proposed search algorithms. The greedy search algorithm in line 1 chooses one-step paths on a static square lattice graph that maximize the probability of finding the target. This is similar to the greedy pursuit policies proposed in [3]. In line 2, we used a three-step receding horizon algorithm on a static square lattice graph, which is similar to the algorithm described in [8]. Lines 3 and 4, when compared to lines 1 and 2, show that the CGBMPS algorithm performs significantly better than the other algorithms that use a fixed time horizon. Additionally, in this scenario, the method of joint routing and sensor tasking performed 7% better than fixed sensor tasking.

The CGBMPS algorithm has also been successfully tested in hardware using a team of two UAVs with cameras to locate a target in a region approximately 2 km by 2 km in size. In each of several tests, the UAVs successfully found a target on the ground within 15 minutes.

VI. Conclusions and Future Work

We proposed a cooperative search algorithm that uses receding horizon optimization on a dynamically updated graph and achieves finite-time target discovery with probability one. The key contributions of this algorithm are (1) optimizing on a dynamic graph updated based on changing target probability distribution, (2) using a waypoint-based prediction horizon allowing for comparison between paths of any length, and (3) incorporating the use of gimballed sensors whose orientations can be jointly optimized with the paths of the agents. Simulations showed that the waypoint-based prediction horizon with a strategic placement of graph vertices causes a significant boost in search algorithm performance over traditional implementations having a fixed time horizon. Using joint routing and sensor optimization provides an additional increase in performance. Furthermore, we have successfully tested the CGBMPS algorithm on a physical system consisting of two UAVs with gimbal-mounted cameras.

We designed this algorithm with a decentralized implementation as the end goal, but have only presented the centralized algorithm in this paper. One could implement a decentralized version of the algorithm by having each agent keep estimates on the positions and sensor measurements of its teammates, updating these estimates whenever agents communicate with each other. In the near future, we plan to present results on the decentralized implementation of the CGBMPS algorithm and an analysis of its robustness to communication loss.

References

[1] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2), 2002.

[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. P. Hespanha, H. J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. volume 3, pages 2432–2437, December 1999.

[4] J. P. Hespanha and H. Kızılocak. Efficient computation of dynamic probabilistic maps. In *Proceedings of the 10th Mediterranean Conference on Control and Automation*, 2002.

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

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

[7] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley, New York, 1992.

[8] 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.

[9] J. R. Riehl and J. P. Hespanha. Cooperative graph-based model predictive search. Technical report, Dept. of Electrical and Computer Eng., University of California, Santa Barbara, September 2007.

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

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