Computing and Executing Strategies for Moving Target Search

A. Kolling[*†] A. Kleiner[◇†] M. Lewis[*] K. Sycara[⋆]

Abstract— **We address the problem of searching for moving targets in large outdoor environments represented by height maps. To solve the problem we present a complete system that computes from an annotated height map a graph representation and search strategies based on worst-case assumptions about all targets. These strategies are then used to compute a schedule and task assignment for all agents. We improve the graph construction from previous work and for the first time present a method that computes a schedule to minimize the execution time. For this we consider travel times of agents determined by a path planner on the height map. We demonstrate the entire system in a real environment with an area of 700,000m^2 in which eight human agents search for two intruders using mobile computing devices (iPads). To the best of our knowledge this is the first demonstration of a search system applied to such a large environment.**

I. Introduction

In this paper we consider the problem of coordinating a team of searchers either robotic or human for finding multiple targets moving in a larger outdoor area. Although this is a fundamental problem inherent in many real-world applications, such as urban search and rescue (USAR), surveillance, and security, there has been only little attention on solutions that can be directly applied to large-scale environments.

We present a comprehensive system that coordinates and guides searchers in an outdoor area of the size of several square miles. To this end we contribute:

1) a novel hierarchical method for computing a graph structure given a height map, i.e. digital elevation model (DEM), of the environment and a solver for computing a search strategy on this graph.
2) A method for computing at each step of the strategy the task assignment from searchers to observation locations while minimizing execution time.
3) A real-time system for executing this schedule in the field, where human searchers can interact with the system via a mobile device, in our case an iPad.

We suppose that no prior information about the number of targets nor their movement or capabilities is given. Considering that the above applications are safety critical we still seek to provide a guarantee of detection and hence assume that targets are worst-case adversaries moving at unbounded speed and are omniscient. This turns our moving target search problem into a pursuit-evasion problem and we can leverage the existing literature in this field.

† joint first authors, ⋆ Robotics Institute, Carnegie Mellon University, 500 Forbes Ave., Pittsburgh, PA 15213, ◇ Computer Science Department, University of Freiburg, Georges-Koehler-Allee 52, 79110 Freiburg, Germany ∗ School of Information Sciences, University of Pittsburgh, 135 N. Bellefield Ave., Pittsburgh, PA 15260.

We demonstrate the entire system in a real environment with an area of approximately 700,000m^2 in which eight human agents search for two intruders using mobile computing devices (iPads). To the best of our knowledge this is the first demonstration of a search system based on pursuit-evasion strategies in such a large environment that considers the entire array of problems from the computation of strategies to task assignment, scheduling, path planning, and a coordinated execution in the field.

In the following we will first present related work in Section II and a problem description in Section III. The graph construction and computation of pursuit-evasion strategies on the graph follows in Section IV. In Section V we consider the computation of a schedule for any strategy and generalize the problem to a task assignment problem with mobile agents and tasks distributed in space. Finally, in Section VI we describe how to integrate the above into a working system and discuss results from a real world demonstration. We conclude with a discussion and an outline for future work in Section VIII.

II. Related Work

There are a large number of related research areas for moving target search within robotics. We shall very briefly mention a few that are most closely related to our problem with an emphasize on graph-based and probabilistic approaches that attempt to control many searchers within realistic environments.

Algorithms to compute search strategies for pursuit-evasion problems on graphs have been discussed in [6], [7] and [4]. Therein an emphasis is put on reducing the number of agents needed for a search strategy with all searchers moving on the graph. For search in real environments these algorithms become useful once a suitable graph can be obtained. In [8] and [5] it was demonstrated that such constructions are feasible and that we can apply graph-based search strategies to coordinate a search in real environments. These, however, only consider reducing the number of agents needed for the search and not the time this takes. In [1] time is considered and the authors present results on the complexity of computing strategies that minimize travel time. The travel time is given by weights on edges. It turns out that minimizing the overall travel time is already strongly NP-complete even on simple graphs such as on stars and trees. One problem with this approach, however, is that the graph on which strategies are computed usually has edges whenever contamination can spread between two vertices. Adding a travel time to such edges treats the graph like a roadmap which it may not be since it primarily captures how contamination and hence target motion can spread. Especially when dealing with complex 2.5d environments the actual best path in the map between any two vertices that are not directly connected with an edge may not correspond to a path in such a graph.

A different graph-based approach is presented in [9]. Therein a graph is created by randomly sampling locations in a 2d environments and it is assumed that targets move according to a probabilistic motion model which is presented by a Markov process that determines how contamination diffuses. The graph is used for an A^* search with a suitable heuristic to search for robot paths on the graph that reduces the level of contamination. Since this approach is computationally expensive a partitioning of the environment with another heuristic is presented. The heuristic attempts to split the graph into roughly two equal parts with a minimal border. Then A-star is run on both parts sequentially while the border is guarded by some sensors. This allows five robots to search a small indoor environment.

There are also a number of approaches that are not directly based on graphs. One probabilistic approach has been described in [3]. This line of work lead to a system presented in [10]. It has three robots searching for a target in a small uncluttered environment. The approach, however, has not been demonstrated to scale well to larger environments. Another problem, shared with most approaches that do not rely on a graph directly embedded into an environment, is to incorporate searcher over which the system has no precise and fine grained control, such as humans.

The first paper to provide an algorithm to construct graphs for our search problem with height maps is [8]. Therein the graph construction is based on randomly sampling locations in the map. Every location has an associated area in which targets are detectable if an agent is placed on the location. These areas are coined *detection sets* and the detection sets of all vertices cover the entire map. Edges between vertices are created whenever two detection sets overlap in a particular manner. Since we are building directly on this work we will present it in more detail in subsequent sections.

III. Problem Description

In this section we review the basic problem formulation given in [8] and present our extensions. We are concerned with the problem of searching for a moving target in a large outdoor area represented by a height map $h : H \to \mathbb{R}^+$, where $H \subset \mathbb{R}^2$ is a continuous domain. For all practical purposes we will approximate H by a 2d grid map. Such maps can model complex terrain and capture some aspects of 3d visibility. An example of a real-word height map is shown in fig. 1. We write $\mathcal{E} \subset H$ for the free space in which agents and targets can move and require it to be connected. In contrast to [8] we allow annotations for H which classify the terrain into either: 1) cluttered (vegetation and other occlusions), 2) non-traversable, and 3) forbidden. In [8] the terrain was only classified as either non-traversable or traversable. Our additions serve two very practical purposes. For one, detailled height maps are difficult to obtain and hence not always up to date. In particular vegetation can change rather rapidly and impede visibility as well as prevent motion. To take this into consideration we elevate the terrain that is classified as cluttered by simply increasing H. Terrain that is not allowed is simply not part of the search area either because it is not of interest or dangerous for the agents.

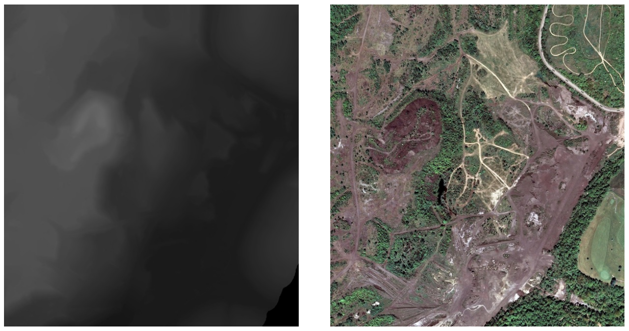

(a) (b)

Fig. 1. (a) A height map of Gascola. (b) A satellite image of Gascola.

To model the detection of an unknown number of unidentified targets we assume a worst-case scenario, i.e. targets have unbounded speed and are omniscient. They are, however, required to move on continuous trajectories within $\mathcal{E}$ and have a minimum height of h_t. We can hence represent the possibility of a target being located in part of $\mathcal{E}$ with contamination. Agents can clear contamination with their sensors and once $\mathcal{E}$ is cleared all targets that were in $\mathcal{E}$ must have been seen at least once by some sensor.

Targets are detected by an agent if they are within line of sight of the sensor mounted at height h_r and within sensor range s_r.[1] The consideration of heights allows us to distinguish the set of visible points in $\mathcal{E}$ from the set of points on which a target is detectable, so called *detection sets*. We write $D(p) \subset H$ for the detection set for an agent on point $p \in H$. More precisely, $D(p)$ is the set of all points p' such that a target of height h_t on p' is within line of sight of the agent on p. For the purpose of clearing contamination we can hence say that when an agent is located on p then $D(p)$ is cleared, i.e. in colloquial terms all targets that could potentially be present in $D(p)$ will have been detected by the sensor. For our purposes we assume an omnidirectional sensor and the computation of $D(t)$ can be done efficiently as shown in SectionIV. In principle one could choose other sensor footprints and hence alter the computation of $D(p)$. In the next section we shall show how to use detection sets to construct a graph representation of the environment and compute search strategies on this graph that will clear all of $\mathcal{E}$.

IV. Computing Strategies

The computation of pursuit-evasion strategies on graphs has been studied in many variations and as a result one has a number of algorithms readily available. But the construction

[1]Strictly speaking visibility is considered in a three dimensional space by embedding H into $\mathbb{R}^3$.

Algorithm 1 $Detection\_Set\_From(p, dir, D)$

```
$l \leftarrow$ set of points on the line segment of length $s_r$ in direction $dir$ from $p$ ordered by distance to $p$.
$a_{last} \leftarrow 0$
for $p' on l$ do
    $a \leftarrow \frac{h(p')-h(p)-h_r}{\|p-p'\|}$
    if $a \geq a_{last}$ then
        $visible \leftarrow true$
        $a_{last} \leftarrow a$
        $D \leftarrow D \cup p'$
    else
        $visible \leftarrow false$
        $a \leftarrow \frac{h(p')+h_t-h(p)-h_r}{\|p-p'\|}$
        if $a \geq a_{last}$ then
            $D \leftarrow D \cup p'$
        end if
    end if
end for
```

of appropriate graphs from real environments has received far less attention. We shall show in this section that improvements in the graph construction can yield significant improvements when using the same algorithms to compute strategies on the resulting graph. We now introduce a hierarchical construction that allows the computation of the detection set for every point on our grid in a lower resolution map. This enables us to rank the detection sets by size and find better locations for new vertices. The quality of a graph is measured by the number of agents that the final strategy requires. Hence, to compare our new construction to prior work from [8] the algorithm for computing strategies on graphs is kept identical to [8], but we shall nonetheless and very briefly discuss it in Section IV-B.

A. Graph Construction

The graph construction in [8] is based on a random sampling of locations for vertex placement. For each sampled location p the detection set $D(p)$ is computed and a vertex added to graph G. New locations p are sampled in parts of $\mathcal{E}$ that are not yet part of any detection set of existing vertices in G. This requires to compute for each selected location p the detection set and to remove it from $\mathcal{E}$. The detection set for location p is computed by casting rays radially and for each ray determine the points on which targets are detectable as shown by Alg. 1 and illustrated by Figure 2. We compute the set of grid cells belonging to the line segment starting in p with length s_r and direction dir with the Bresenham algorithm known from the field of computer graphics.

One advantage of random sampling is that one does not have to compute the detection set for the majority of the points in $\mathcal{E}$, but only for those that are selected as graph nodes. Randomly selected locations, however, are not necessarily those from which larger parts of the map can be observed. They could be located in valleys or between shrubs and thus having occluded sight. Instead we should

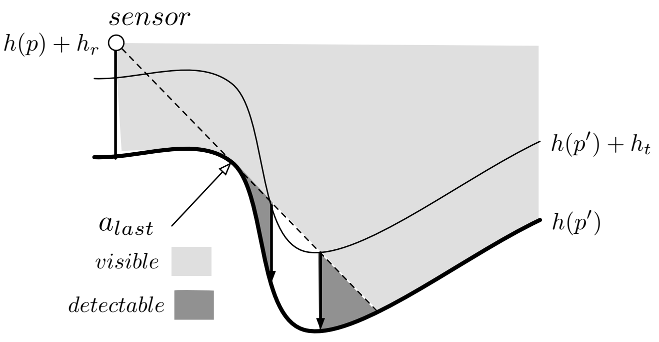


Fig. 2. An illustration how to compute detection sets for Algorithm 1.

rather select locations with good visibility, such as mountain peaks or bell towers.

Therefore, we present a hierarchical sampling approach that automatically selects locations with large detection sets for constructing the graph. This is carried out by generating a set of L low-resolution copies $\mathcal{M} = (M_1, ..., M_L)$ of the height map, where M_l denotes the map copy at level l with resolution $r_l = r_0 \frac{1}{2^l}$, and r_0 denotes the resolution of the original hight map. Height cells at lower resolutions are generated from the previous level by a conservative approach, which is to assign the maximum of the height values from the four corresponding cells on the lower level. Figure 3 depicts the generation of two low resolution maps at level 1 and 2 from the original map.

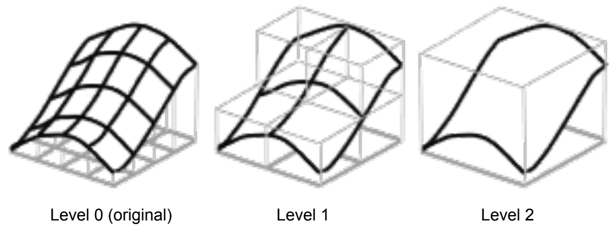


Fig. 3. Hierarchical simplification of height maps for computing detection sets. Level 0 represents the original map storing at each grid cell a height value. Higher levels combine the height values of the four grid cells of their predecessor level by the $max()$ operator.

Likewise as shown for the random sampling procedure, the idea is to successively sample locations p from $\mathcal{E}$ and to remove their detection set $D(p)$. But instead of randomly sampling points, we identify those with the largest detection set by a depth search on the hierarchy of $\mathcal{M}$. The search starts at the highest level L, i.e. lowest resolution, of the hierarchy by computing for each point p_L its detection set $D(p_L)$. From these sets the location with the maximum detection set $p_L^{max} = \text{argmax}_{p_L} |D(p_L)|$ is selected and the search is continued on level $L - 1$. After locating the maximum set on the highest level L, it suffices to search on lower levels in a depth-first search manner. This is carried out by computing the selection set S_{L-1} consisting of location p_{L-1} that corresponds to p_L^{max} plus further locations around this location within a small neighborhood radius ϵ. Principally, it suffices to select ϵ exactly to cover the four cells on $L-1$ from which p_L^{max} has been generated. However, in order to compensate for quantization errors we used $\epsilon = 4$ for our experiments. From S_{L-1} the best candidate of level $L-1$ is selected by $p_{L-1}^{max} = \operatorname{argmax}_p D(p \in S)$. This procedure is continued until level 0 is reached and thus location p_0^{max} on the original map with maximal detection is found. Then, $D(p_0)$ is removed from $\mathcal{E}$ and hierarchy $\mathcal{M}$ updated accordingly. The hierarchical sampling continues until the entire map has been covered. Once all vertices are sampled we can proceed by adding edges between these vertices. Options for how to determines these are discussed in detail in [8] and for our purposes we merely add edges between any two overlapping detection sets whenever the intersection is not covered by a detection set of a third vertex whose detection set is larger than either one of the two other vertices.

Notice that even though we are selecting vertices with larger detection sets this is still a heuristic and by no means guarantees better strategies. Yet, we shall show in Section VII that we do get a significant improvement for our Gascola test site.

B. *Strategies on Graphs*

Given the graph G constructed as above we now have to compute pursuit-evasion strategies on it. To allow a comparison between the graphs from [8] we use exactly the same algorithm for this purpose but shall very briefly describe a few details here.

We initially consider all vertices of G contaminated. An agent moving onto a vertex $v \in G$ will clear it since it detects all targets in the associated detection set. We shall refer to this as *guarding* a vertex since it also prevents any target from crossing through the detection set of v for the duration that the agent is guarding it. Note that once a guarding searcher is removed from a vertex contamination can spread through it. The problem is now to determine when to guard each vertex to clear a contaminated graph G with as few agents as possible. Additionally, we require that the strategy ensures that all cleared vertices are connected and that no vertices are recontaminated. These properties are known as contiguous and progressive, respectively. For our searchers this ensures a safe area within $\mathcal{E}$ and that every vertex has to be visited only once. But it has to be noted that allowing recontamination or disconnected cleared vertices can potentially lead to strategies using fewer agents and one could also choose to compute such non-contiguous or non-progressive strategies instead.

It turns out that we can utilize prior work in pursuit-evasion on graphs with some modifications to determine such strategies. The details of these modifications are presented in [8] and the resulting algorithm determines strategies that are represented by a sequence of vertices S. Now, the sequence S determines when a vertex has to be guarded. A guarding searcher can be released from the vertex once all its neighbors in G are also cleared, ensuring that no recontamination occurs. The sequences S hence not only determines when agents are placed on vertices but also when they can be removed.

For our purposes we can consider any strategy S that determines when agents are placed and removed from vertices so long as it guarantees that if this sequence is followed then the entire graph is cleared. Hence the underlying algorithm for computing S can be readily replaced and does not have to be optimal. The main problem we are concerned with is how to translate a strategy S into a schedule for an agent team that assigns every agent to a vertex while minimizing the time it takes to travel to all locations. This problem is addressed in the next section.

V. Executing Strategies

Given a pursuit-evasion strategy that requires k agents, written $a_i, i = 1, \ldots, k$ we will now compute an assignment of the guarding tasks to agents and attempt to minimize the time it takes for all agents to execute the strategy. In our case a contiguous strategy is given by a sequence of vertices that need to be guarded. Let us write $v_1, \ldots, v_n$ for this sequence. Once a vertex has no contaminated neighbors anymore its guarding agent is free to move to another vertex without incurring recontamination. This occurs precisely when the last neighboring vertex is guarded and thereby cleared from contamination. We can hence generate a task τ_i for every $i = 1, \ldots, n$ that starts at step i and terminates after some step $j \geq i$, i.e. the agent is released at step j when task τ_j is started. In principle this conversion can be applied to other types of pursuit-evasion strategies such as Graph-Clear [7] which involves actions other than guarding as well as actions on edges.

We shall now define a task $\tau_i := (l_i, d_i)$ as a tuple of a location l_i that corresponds to the location of vertex v_i in the map H and d_i which is the step until which l_i needs to be occupied. The sequence of tasks is entirely determined by our strategy, but the assignment of agents to these tasks is not.[2]

To complete a task $\tau = (l, d)$ an agent a needs to arrive at location l and occupy it until we reach step d. Step d is completed once all locations $l_j, j \leq d$ had been reached (although some of the agents may already be released from these locations). Once step d is completed agent a can continue moving towards another task location. Some task locations are hence be occupied in parallel since multiple agents may be waiting for their release. By construction the total number of agents occupying task locations will not exceed k. Fig. 4 illustrates the new task sequence arising from a strategy.

The overall execution time for the strategy is determined by the speed at which agents can travel to the locations of their assigned tasks with each agent's time at the location depending on other agents. Let us now briefly formalize the problem.

[2]Note that there are pursuit-evasion problems and algorithms that immediately assign an agent to an action, but to our knowledge there are none that consider the number of agents as well as execution time with an underlying path planner.

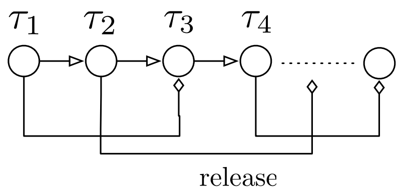


Fig. 4. A sequence of tasks arising from a strategy. Here $\tau_1 = \{l_1, d_1 = 3\}$ is released after the location l_3 of the third task τ_3 has been reached by an agent. Tasks τ_1, τ_2, τ_3 are guarded in parallel while the agent from τ_1 may be used to for τ_4 since it is released at step 3.

Definition 1 (Task Assignment): A task assignment is a surjective function $\mathcal{A} : \{\tau_1, \ldots, \tau_n\} \to \{a_1, \ldots, a_k\}$ with the following property: if $\mathcal{A}(\tau_i) = \mathcal{A}(\tau_j)$ for some $j > i$ then $d_i < j$.

In colloquial terms, this definition just ensures that every agent has at least one task and that an agent cannot be assigned to another task before it is released. To formalize the contribution of travel time let us write $a(t)$ for the location of agent a at time t. Further, write $\mathcal{T} : \mathcal{E} \times \mathcal{E} \to \mathbb{R}^+$ to represent a path planner that returns the time it takes for an agent to travel between two locations in $\mathcal{E}$ written $\mathcal{T}(l, l')$. Write t_i be the time at which step i is completed. We can now define t_i inductively via $t_0 := 0$ and

$$t_{i+1} := t_i + \mathcal{T}(\mathcal{A}(\tau_{i+1})(t_i), l_{i+1}). \quad (1)$$

Notice that the term $\mathcal{T}(\mathcal{A}(\tau_{i+1})(t_i), l_{i+1})$ may well be 0 if the agent $\mathcal{A}(\tau_{i+1})(t_i)$ is already on l_{i+1} at time t_i. In fact, with a larger number of agents we should expect this to occur frequently as agents are moving in $\mathcal{E}$ simultaneously. Fig. 5 shows three steps that finish at the same time, i.e. $t_3 = t_4 = t_5$.

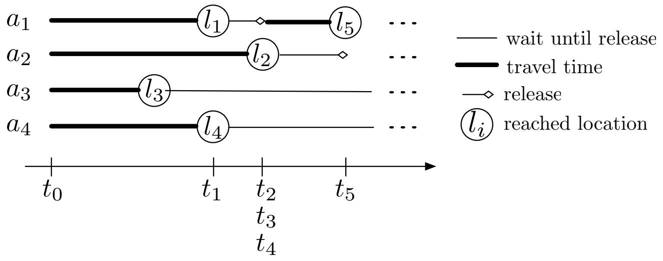


Fig. 5. Agents a_1, a_2, a_3 and a_4 move to locations l_1, l_2, l_3 and l_4 respectively. Step 1 is completed once a_1 reaches l_1. Other agents may already be at their assigned locations at this time. At step 2 agent a_1 is released and proceeds to l_5. Since a_3 and a_4 have already reached their task locations at t_2 we have $t_2 = t_3 = t_4$. Agent a_2 is released once a_1 reaches l_5 and so on.

Obviously, the above assumes that agents actually move towards their next assigned tasks immediately after release. For an agent a let $\mathcal{A}|_a := \{\tau \mid \mathcal{A}(\tau) = a\}$ be the set of all tasks assigned to a, ordered with their index ascending as before. For convenience let us write $\mathcal{A}|_a = \{\tau_1^a, \tau_2^a, \ldots, \tau_{n_a}^a\}$ where $n_a = |\mathcal{A}|_a|$. At time t_0 every agent a immediately moves towards their first task $\tau_1^a = (l_1^a, d_1^a)$ following the planner and needing $\mathcal{T}(a(t_0), l_1^a)$ time units and at every subsequent release they move immediately towards the next assigned task location. We can now formalize our main problem:

Definition 2 (Minimal Assignment): Given a fixed $\mathcal{T}$ a sequence of tasks $\tau_1, \ldots, \tau_n$ and agents $a_1, \ldots, a_k$ let the minimal assignment $\mathcal{A}_{min}$ be such that:

$$\mathcal{A}_{min} = argmin_{\mathcal{A}}\{t_n\} \quad (2)$$

To compute a task assignment $\mathcal{A}$ it is helpful to compress the notation. Instead of the sequence of tasks we now consider sets of tasks whose locations have to be reached before another task is released. This is useful because between releases we have a constant number of agents available and a constant number of tasks that all have to reached before new agents become available.

Let us write $\tilde{t}_0, \tilde{t}_1, \ldots, \tilde{t}_{\tilde{n}}$, for the times at which at least one agent is released. At $\tilde{t}_0$ all agents are free and can be assigned to tasks. Write $\tilde{F}_0 = \{1, \ldots, k\}, \ldots, \tilde{F}_{\tilde{n}}$ for the set of free agents after the step completing at $\tilde{t}_i$. Let $\tilde{T}_i$ be all tasks that have an agent on their location at time $\tilde{t}_i$, $i = 1, \ldots, \tilde{n}$.

This compressed notation gives us an immediate first insight. Namely, to minimize the time difference $\tilde{t}_i - \tilde{t}_{i-1}$ we have to solve a Linear Bottleneck Assignment Problem (LBAP) and match some agents from F_{i-1} to the new tasks $\tilde{T}_i \setminus \tilde{T}_{i-1}$, $i = 1, \ldots, \tilde{n}$. The cost of an assignment between a free agent a and a task $\tau = (l, d)$ is simply given by the difference between $\tilde{t}_{i-1} - t_{arrival}$ where $t_{arrival}$ is the earliest time, as determined by $\mathcal{T}$, at which a can be at l, i.e. $t_{arrival} = t_{last} + \mathcal{T}(a(t_{last}), l)$ where t_{last} is the time at which a became released and hence free. Using this we can build a cost matrix $c(a, \tau)$ to capture the cost of each possible assignment. Note that $|F_{i-1}| \geq |\tilde{T}_{i+1}|$ and we have to add an idle task τ_0 so that the LBAP would assign some robots to a dummy task τ_0 that the agent simply ignores when moving to the next location. From here on any LBAP algorithm can be applied to minimize $\tilde{t}_i - \tilde{t}_{i-1}$ and for $\tilde{t}_i$ this would give us the minimum possible value, given that $\tilde{t}_{i-1}$ was fixed. But this does not guarantee that $t_n = \tilde{t}_n$ is minimal and brings us directly to the main problem which is best illustrated with the following example.

Suppose we have four agents a_1, a_2, a_3 and a_4 and $\tilde{T}_1\{\tau_1, \tau_2\}$ with $\tau_1 = \{l_1, 2\}, \tau_2 = \{l_2, 4\}$ and $\tilde{T}_2 = \{\tau_3, \tau_4\}$ with $\tau_3 = \{l_3, 4\}, \tau_4 = \{l_4, 4\}$. Fig 6 shows the locations of the agents and $l_1, \ldots, l_4$ and two different assignments for $\tilde{F}_0$ on $\tilde{T}_1$ that in turn allow different assignment for F_1 onto $\tilde{T}_2$. The assignments are also shown in fig. 7. It is easy to see that an optimal solution to the LBAP for $\tilde{F}_0$ on $\tilde{T}_1$ leads to an overall worse solution. In colloquial terms, we can sacrifice some time in an assignment at one step and instead choosing to give an idle task to an agent that will travel to its tasks for a subsequent assignment and thereby improving it. This can lead to overall less time spent, i.e. a smaller t_n.

The dependency between subsequent assignments is due to the fact that some robots can be assigned to tasks for future steps if previous steps do not utilize all agents. If at every step the number of tasks is equal to the number of agents, then the

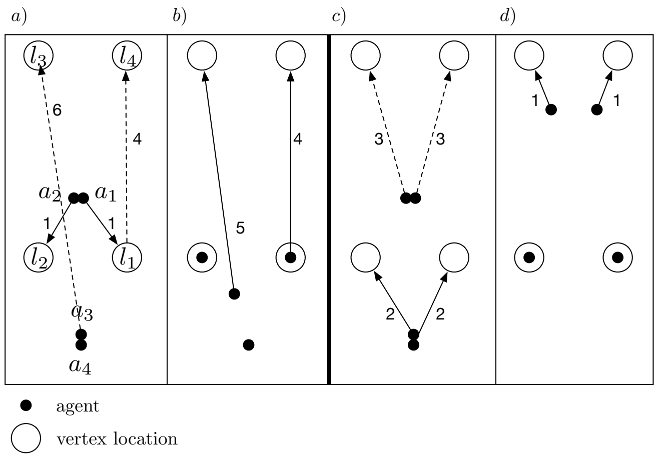


Fig. 6. Each part a)-d) shows four locations and agents. Part a) and b) show one assignment in which agents a_1 and a_2 move to l_1 and l_2, the optimal assignment to minimize $\tilde{t}_1$. After $\tilde{t}_1$ agent a_2 is released and at this point assigning a_2 and a_3 is the optimal assignment to minimize $\tilde{t}_2$ given that $\tilde{t}_1$ is fixed. Part c) and d), however, show an assignment that leads to a larger $\tilde{t}_1$ but smaller $\tilde{t}_2$.

repeated solving of the linear bottleneck assignment problem (LBAP) will yield an optimal solution. Otherwise, from a global perspective, the repeated computation of locally optimal LBAP solution is a greedy algorithm.

Fig. 7 shows the assignment of agents to tasks in a familiar manner for LBAP problems in the form of consecutive bipartite graphs. Repeated assignment problems are also known as multi-level assignment problems and one variant that has some resemblance to our problem is presented in [2]. Unfortunately it is $NP-complete$ and we conjecture that this may be the case for our minimal assignment as well. A detailed exposition is, however, beyond the scope of this paper and for our purposes the presented approach to solve multiple LBAPs is sufficient and in Section VII we shall see that this already leads to a significant improvement.

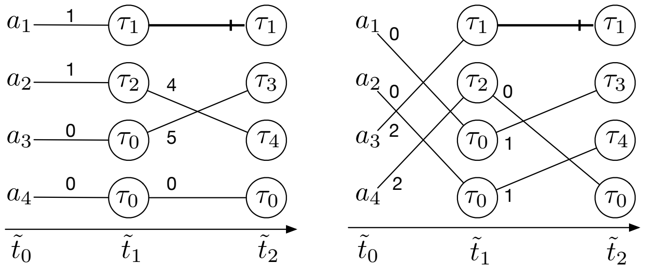
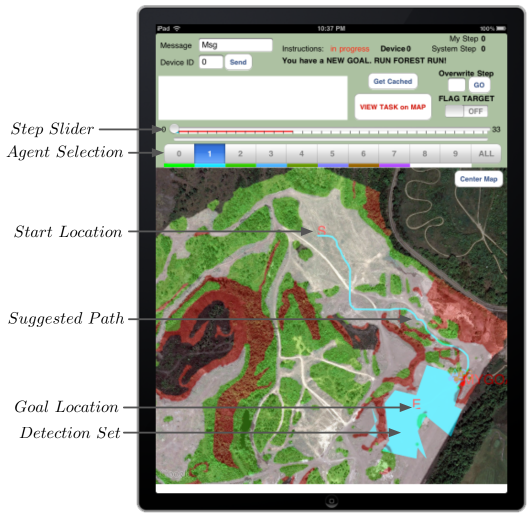


Fig. 7. Two task assignments visualized as graphs that correspond to fig. 6. The agent assigned to τ_1 has to remain there until release while τ_2 can be reassigned in the second level LBAP. The consecutive LBAP solution for both levels shown on the left is, however, not optimal for $\tilde{t}_2$.

VI. System

Few of the prior work on searching for moving targets or pursuit-evasion has ever been tested in real world applications, especially not in large and realistic environments. One main obstacle is the integration of all aspects of the problem from mapping up to the computation and coordinated execution of search strategies. In this section we describe a system that integrates all solution to these problems presented here and in previous work and guides searchers through a large outdoor environment.

Using an annotated height map we constructed a graph representation as described in Section IV-A but taking into account additional obstructions for visibility due to cluttered terrain. On this graph we computed a strategy, using the algorithm presented in [8] and a corresponding task assignment and schedule following the procedure described in Section V. This schedule is then made available to all agents. For our demonstration we used human agents equipped with mobile devices (IPads) on which we programmed a custom Objective-C application. All devices were communicating via a 3G connection and all data was logged at a central location. The interface of the application is shown in Fig. 8. All searching agents had information about the instructions of all other searching agents and their locations by exchanging GPS data at two second intervals. The evading agents, i.e. targets, were also given a device each to simulate a smart evader. Only evading agents were able to see the location of other evaders. When a searching agents saw an evader they logged the encounter by touching the respective area inside the map.



Message Msg
Device ID 0
Instructions: in progress Device 0
My Step 0
System Step 0
You have a NEW GOAL. RUN FOREST RUN!
Overwrite Step
FLAG TARGET OFF

Fig. 8. The IPad interface for all agents. Additionally each agents receives location information on other searchers in the display and can monitor their progress. Evading agents receive additional information about other evaders.

All agents received a warning signal if their GPS indicated that they were close to terrain that was classified as not traversable. Once agents reach their assigned location for a step in the execution the system sends messages to other agents informing them about the progression and their new tasks. Fig. 9 shows the main parts of the system in an overview.

VII. Demonstration and Results

Here we present our results from applying all of the above to our test site Gascola. The height map of Gascola shown in

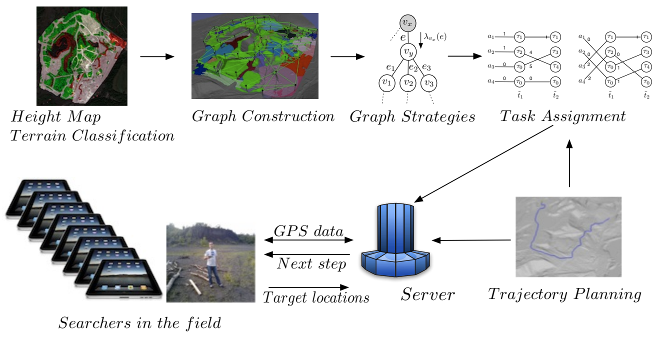


Fig. 9. A high level overview of the system.

fig. 1 has a resolution of $1m$ per pixel. The entire area of the site is approximately 700,000 m^2. The lowest point in the map is set to $0m$ elevation and the highest point is at $122m$. Gascola has a lot of seasonal shrubs and other vegetation that influence visibility and movement of agents. We hence surveyed the terrain a week prior to the deployment of the agents and added the annotations seen in fig. 10. Collecting detailled height maps is a considerable efforts and these annotations allow us to accomodate short term changes in the terrain.

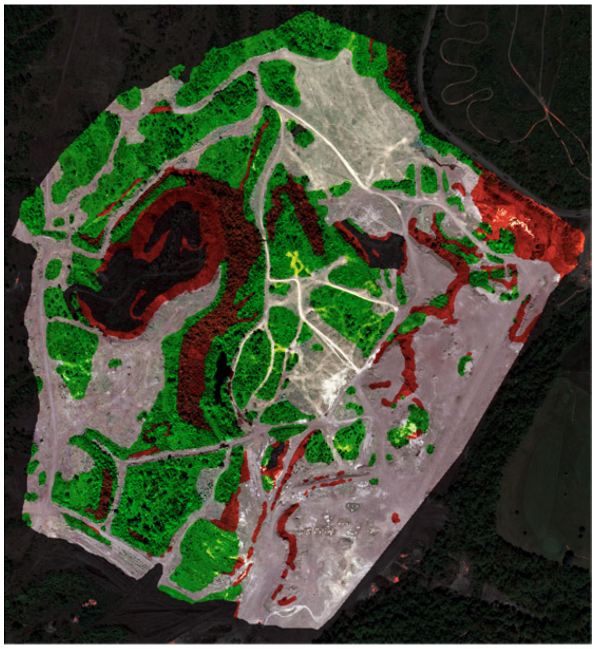


Fig. 10. Our sample map of the Gascola area outside of Pittsburgh with additional annotation. In green we see cluttered terrain, mostly shrubs and debris, red are steep areas that are not traversable by our agents and black areas not admissible and define the area of the search.

The hierarchical approach yields a significant reduction of the graph complexity particularly on cluttered maps while guaranteeing a full coverage of the terrain. On the *Gascola* map the random sampling yielded in average 102 vertices and 240 edges, whereas the hierarchical approach reduced this amount in average to graphs with 62 vertices and 130 edges. The reduction had a positive impact on the number of agents needed for the schedule. Solutions computed based on randomly sampled graphs needed in average 13.9 ± 1.1 agents, whereas solutions based on hierarchical sampling required in average 7.8 ± 0.2 agents. Note that results were averaged over 100 experiments each. Finally, we selected a graph and with a strategy requiring eight agents computed on the graph shown in 11. The associated detection sets for all vertices are also shown in fig. 11. These were uploaded to the mobile devices so that agents can see what detection sets they are responsible for.

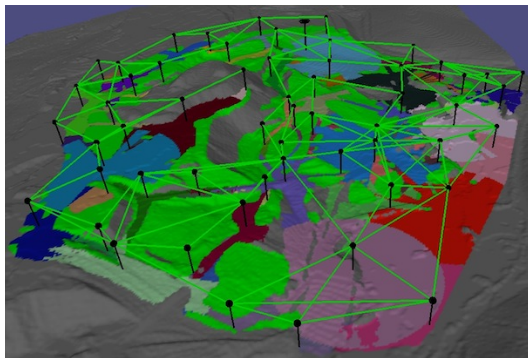


Fig. 11.

We then computed the execution time using our procedure from Section V yielding an assignment that takes 175 minutes to execute. In order to determine the impact of our procedure on execution time we compared it to 10,000 random assignments. These random assignments simply assign free agents randomly to new tasks at each step. Here we get a solutions with a mean execution time of 349.3056 ± 34.0350 minutes and with a maximum at 491.6365 and minimum at 236.4207. Hence the improvement is significant and can safe our searchers in the field in Gascola a whole hour of search time. Obviously, the problem deserves further study and experimentation on more maps. It should also be noted that instead of using an LBAP solution at each level we can solve the general assignment problem and thereby minimize the sum of all travel times instead of the maximum. This could be useful for applications in which energy conservation is more important and some of the execution time can sacrificed.

All participants, eight searchers and two evaders, received a 15 minute instruction on how to use the application. The two evaders were given a head-start of another 15 minutes. They were instructed to make use of the available information on all searchers as best as possible to try to avoid being captured. Most agents were instantly able to follow the suggested paths and reach their locations. Two agents, however, had considerable difficulty at first to orient themselves and each one got lost once causing a delay of the execution but never leading to a breach between the boundary of contaminated and cleared space. After the first hour, however, all agents were comfortable following the instructions as the execution proceeded further. The experiment continued until the first IPads ran out of battery power at which points all participants were recalled. The searchers managed to execute two thirds of the entire strategy during this time and catching every evader at least once.

Fig. 12 shows a snapshot of the GPS data during the first half of the execution of the strategy. The purple evader was caught three times by three different agents attempting to move into the cleared areas undetected. The blue agent, however, managed to run behind the area controlled by one of the guards at the top of the map and successfully breached the perimeter. The GPS log clearly shows that the searcher in charge abandoned his area without instructions. This issue illustrates the necessity for thoroughly instructing the searchers when applying the system. The blue evader was, however, subsequently detected by another agent.

The main conclusions to draw from this field demonstration is foremost the feasibility of such an integrated system. Secondly, we observed that a team of human agents is by no means a homogenous team. Each agent has different walk speeds and capabilities in following the instructions. Furthermore, also the environment despite a recent survey had changed due to rainfall and some of the precomputed paths were in fact blocked. This had no effect on the guarantee of the strategy but did delay execution since new paths had to be found by the affected searchers. These two issues, heterogeneity and dynamic changes in the environment clearly outline problems for further study.

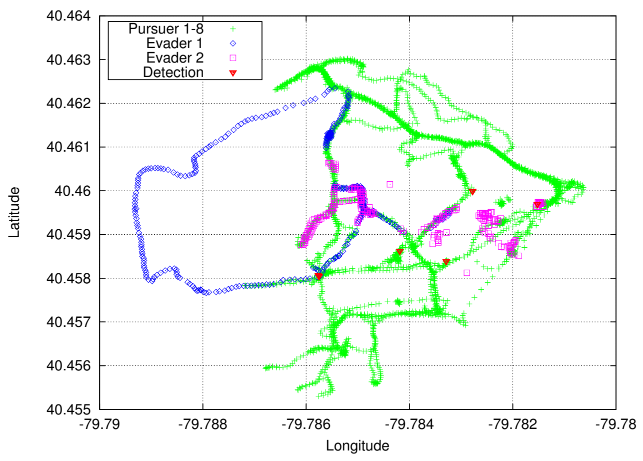


Fig. 12. A snapshot of the GPS log from all searchers and evaders.

VIII. Discussion and Conclusion

In this paper we presented three primary contributions. First we improved the graph construction from previous work which now allows us to apply pursuit-evasion algorithms to a large complex height map and obtain strategies that use significantly fewer agents. Second, we defined and provided a first algorithm for the problem of executing search strategies by assigning individual search tasks to agents in order to minimize the overall time needed to execute the search. The difficulty of finding optimal solutions, however, is an open problem and our algorithm only computes optimal solutions under restricted conditions. Third, the reduction in the number of agents and the minimization of execution time allowed us to demonstrate the entire set of solutions in one comprehensive system in a wide outdoor environment. Despite the potential for even further improvements the current setup is already feasible for helping teams of agents to execute wide area searches as demonstrated by the field test. To the best of our knowledge this is the first large-scale demonstration of the application of a graph-based pursuit-evasion problem.

Evidently, there are a number of issues that still need to be addressed more thoroughly. First, the hierarchical graph construction will need to be evaluated against other constructions on more maps to determine its merit beyond our Gascola map. Second, the computation of minimal assignments for the execution of search strategies needs further study to determine its complexity and find either optimal polynomial time algorithms or approximation algorithms with performance bounds.

Furthermore, one of the main insights from our field demonstration is that agent teams are never truly homogenous, especially with human agents. Developing algorithms that accomodate this fact is highly desirable. Another important question relates to the fact that real environments and searchers are dynamic. New observations or unanticipated events such as an immobilized agent or a blocked path are currently not dealt with. Hence it is desirable to extend the presented work to such dynamic scenarios in which the agents and observations about the environment interact with the strategy and adapt it. Addressing these issues will bring us closer to a feasible and viable systems that can have real implications for large scale search in the real world.

References

[1] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 59–66, 2009.

[2] P. Carraresi and G. Gallo. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *European Journal of Operational Research*, 16(2):163–173, 1984.

[3] J.P. Hespanha, H.J. Kim, and S. Sastry. Multiple-agent probabilistic pursuit-evasion games. In *IEEE Conference on Decision and Control*, volume 3, pages 2432–2437. Citeseer, 1999.

[4] G. Hollinger, A. Kehagias, S. Singh, D. Ferguson, and S. Srinivasa. Anytime guaranteed search using spanning trees. Technical Report CMU-RI-TR-08-36, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2008.

[5] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2323–2328, 2008.

[6] A. Kolling and S. Carpin. Multi-robot surveillance: an improved algorithm for the Graph-Clear problem. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2360–2365, 2008.

[7] A. Kolling and S. Carpin. Pursuit-evasion on trees by robot teams. *IEEE Transactions on Robotics*, 26(1):32–47, 2010.

[8] A. Kolling, A. Kleiner, M. Lewis, and K. Sycara. Pursuit-evasion in 2.5d based on team-visibility. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. (accepted for publication).

[9] M. Moors, T. Röhling, and D. Schulz. A probabilistic approach to coordinated multi-robot indoor surveillance. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3447–3452, 2005.

[10] R. Vidal, O. Shakernia, H. Kin, D. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, 2002.