Efficient Algorithms for Pursuing Moving Evaders in Terrains

Alon Efrat
Department of Computer Science
The University of Arizona
alon@cs.arizona.edu

Joseph S. B. Mitchell
Department of Applied Mathematics and Statistics
Stony Brook University
jsbm@ams.stonysb.edu

Parrish Myers
Department of Computer Science
The University of Arizona
pmyers@email.arizona.edu

Swaminathan Sankararaman
Department of Computer Science
Duke University
swami@cs.duke.edu

ABSTRACT

In this paper, we propose algorithms for computing optimal trajectories of a group of flying observers (such as helicopters or UAVs) searching for a lost child in a hilly terrain. Very few assumptions are made about the speed or direction of the child's motion and whether it might (either deliberately or accidentally) try to avoid being found. This framework can also be applied to a set of seekers searching for hostile evaders such as smugglers/criminals, or friendly evaders such as lost hikers.

Based on the features of the area of the terrain where the pursuit takes place, and the visibility and motion characteristics of the UAVs, we show how to plan their synchronized trajectories in a way that maximizes the likelihood of a successful pursuit, while minimizing their battery or fuel usage, which may, in turn, enable a longer pursuit. Our algorithm explores useful I/O-efficient data structures and branch-cutting (search pruning) techniques to achieve further speedup by limiting the storage requirements and total number of graph nodes searched, respectively.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics—*Autonomous vehicles, sensors, workcell organization, planning*

General Terms

Algorithms, Performance, Experimentation

Keywords

Algorithms, UAV, coordinated route planning, branch-cutting, sensors, I/O-Efficient data structures

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

1. INTRODUCTION

In this paper, we propose solutions for a set of optimization problems related to searching in terrains demonstrated by the following scenario. A young child is lost in hilly terrain. His/her actions are unpredictable and location unknown, but constrained to a known region of the terrain. A team of rescuers needs to be assembled, to conduct a search. The recent decrease in the cost of unmanned aerial vehicles (UAVs) together with the increase in their capabilities suggests the possibility that multiple (possibly a large number of) such UAVs could play the role of rescuers in this search. Due to the child's mobility, and the possibility that he/she intentionally attempts to avoid to being found, a careful synchronization between the UAVs' movements is needed to guarantee that he/she will be found as soon as possible. In this paper we present an algorithm for computing the UAVs' trajectories, and their synchronized motion along their trajectories, in a way that maximizes the probability that the child is found as quickly as possible while minimizing resources such as fuel or battery consumption, in order to increase operational search time. The process is referred to as a *pursuit*, and is *successful* once the child is found. This framework can also be applied to any scenario where terrain searching is needed; for example, a set of seekers searching for lost hikers, smugglers or terrorists, or to similar military missions. For generality, we will use the terms *evader* and *seekers*, rather than child and UAVs.

We next look at the constraints governing the search. To find the evader, one of the seekers, must be able to detect it. This limits the elevation above the surface and distances between the seekers. In particular, mountains or other terrain features must not visually occlude the evader from all seekers. However, even having a line of sight to the evader might not be a sufficient condition for finding it. For example, at one extreme of altitude, it might be possible for a helicopter at a high enough altitude to "see" every point of the search area of the terrain, yet the resolution of the optical or thermal imaging devices and poor visibility might prevent the seeker from distinguishing enough details to detect the evader. Vegetation (trees, bushes) also limits the visibility. At another extreme, flying helicopters at ground level is also problematic, since the regions they can search

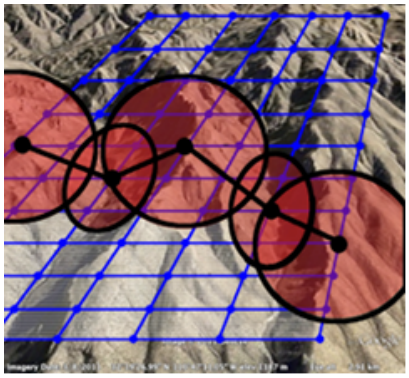

Figure 1: *Simulated search in Tucson, AZ mountains with a chain of seekers. Each seeker is represented by a black dot and the red ellipse it its visibility region. Overlapping the visibility regions of each seeker forms a chain. The blue grid is the entire defined search region.*

is much smaller, and requires more maneuvers than would be necessary at higher elevations. Thus, we assume that there is an interval of elevations between λ_{low} and λ_{high}, measured *above the terrain surface*, that are optimal for the search. We design seekers' trajectories according to terrain features, trying to keep them within this interval.

Proposed approach: Influenced by [8], our proposed solution starts by assuming that the terrain is split into large regions, with each being *"swept"* by a chain of seekers (a formal definition is below). The notion of "sweeping" here means that the seekers always form a chain and separate the terrain (or a portion of the terrain) into two distinct areas, commonly called the *clean region*, already searched and no possibility of the evader occupying this area; and the *contaminated region*, yet to be searched and possibly containing the evader. See Figure 1 for an illustration. At the beginning of the process, the contaminated region constitutes the whole terrain. As the chain moves (sweeps) the terrain, the clean region deforms continuously (not necessarily monotonically) until it constitutes the whole terrain. The seekers are always positioned so that their visibility regions (the portion of the terrain each one monitors at each moment) overlap, so the evader could not "sneak" between two of them without being noticed. It is clear that as long as the evader is in the terrain and is (at least partially) visible, this search strategy is guaranteed to find it, independent of its pattern of motion. The question then becomes: given a hilly terrain, what is the best motion of the chain of seekers to minimize expenditure of resources, but still thoroughly search the region. Natural criteria for optimization that come to mind include (i) minimizing the horizontal distance between the seeker and each point it surveys, (ii) minimizing the horizontal distance between each seeker and neighboring seekers, (iii) minimizing resources needed (such as fuel or battery) and (iv) minimizing time. Minimizing resources such as fuel depends on a variety of parameters, but factors that are relevant are total time, velocity and elevation gain. Minimizing the time is related not only to the constraints of the seeker itself, but also to time that the each seeker needs to spend to maintain synchronized motion of all seekers forming the chain. Sweeping the terrain could be accomplished using many geometric patterns, possibly optimizing different cost functions.

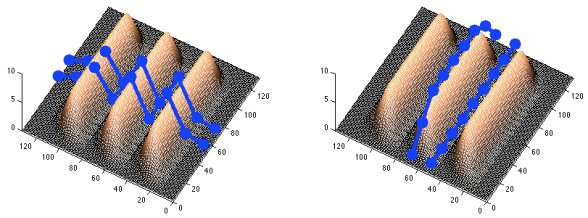

Figure 2: *Options for searching through a series of mountains while maintaining elevation λ above the terrain: (Left) optimal configurations show at time t_1 and t_2 to minimize elevation changes; (Right) poor choice if elevation changes should be kept to a minimum.*

Figure 2 demonstrates two different sweeping patterns of the same **synthetic** terrain. This terrain consists of 3 mountain ridges, which are assumed to be at a maximum elevation E, and the valleys in between are at elevation 0. Two natural ways to perform the sweep are by flying from south to north (Figure 2(a)) or from east to west (Figure 2(b)). In both cases, each seeker is always at elevation α above the terrain, where λ is a constant as above. Both strategies would sweep the terrain. However, if the major component of the cost is the elevation gain, then the first method is probably preferable, since each helicopter changes its height at most once and gains a total height that for some seekers is λ, and for the others it is $E+\lambda$ as they only need to reach the ridge of the mountain, on which it could fly horizontally. In the second approach (Figure 2(b)) each one of the seekers needs to gain an elevation of $3E+\lambda$, since it needs to "climb" each one of the three mountain ridges.

In this paper, we present a generic framework that maximizes the time for which available resources can be utilized by a group of seekers searching in a terrain while imposing minimal constraints on an evader's actions. We explore methods to minimize the data required to be in main memory and optimize data structures for execution speed.

Examples of seekers that our approach could be applied to, are depicted in Figure 3. Each can be used for search and rescue and purchased commercially. These types of smaller UAVs are less capable than the larger variety, but are less expensive to own and maintain, and can be outfitted with a variety of sensors, and equipment. This makes them well suited for coordinated search and rescue where many UAVs are needed. Efficient trajectories produced by our algorithm for different instances can be found at `http://www.cs.arizona.edu~pmyers/mbrain`.

2. PREVIOUS WORK

Our problem joins the vast literature of pursuit-evasion problems. These problems have been studied both in geometric domains [8, 13, 16] and in abstract graph settings [4, 17]; see [10, 1] for further details. In the graph setting, one of the common settings is that both seekers and evaders are located on the vertices of a given graph. At each time step each one could move to an adjacent vertex. The evader is "caught" once a seeker occupies a neighboring vertex. More relevant to our work is [16] that suggests strategies for chas-

[1]Camcopter is a registered trademark of Schiebel Corporation

Figure 3: *Examples of possible UAVs: (Left) Schiebel Camcopter*[1] *S-100, which can stay airborne for more than 6 hours with an operational range of 45km to 180km and costs approximately* $\$500,000$*; and (Right) Aeryon Labs Inc. Scout, which can stay airborne for up to 25 minutes with an operational range of 25km and costs approximately* $\$50,000$.

ing moving targets in a polygonal domain. Our work is also related to the problem of surveying a geometric domain, such as terrains. This problem, also called "sensor placement" or "art-gallery problem", has been studied extensively in different communities, stressing different aspects. In the abstract setting, one wishes to place static guards in a terrain, such that the smallest number of guards is needed and the region collectively seen by all guards is maximized. See for example, [2, 3, 7, 11, 8, 9, 12]. However, these works do not seem to be applicable directly to the case of mobile targets.

3. PROBLEM FORMULATION

A terrain in our context is a graph of an elevation map (or height above sea level) $\mathcal{H}(x, y)$. Here $\mathcal{H}(x, y)$ is the elevation of the point (x, y) above sea level. We are also given the *pursuit region* P, which is a rectangular region P on the xy-plane.

A *seeker* is defined as a generic UAV whose location is a point g above the terrain and is capable of detecting the evader. Its position at time t is represented as a point (x, y, z) where z is the elevation above sea level and $z \geq \mathcal{H}(x, y)$ and (x, y) lies within the pursuit region P.

The *visibility region* of g, denoted $\mathsf{Vis}(g)$ is the set of all points of the terrain at which an evader could be observed by g with sufficient confidence. So a point p belongs to $\mathsf{Vis}(g)$ if the segment connecting p to g does not intersect any other points of the terrain and the distance $\|p-g\|$ does not exceed a pre-specified threshold. Note that the visibility region is usually not a connected region of the terrain, and actually any obstacle in the terrain (e.g. trees) might create a region not seen by the seeker (see [5]). To be able to study our optimization problem, some abstraction is needed. As discussed in the introduction, the elevation of seekers needs to lie in an interval of elevations $[\lambda_{low}, \lambda_{high}]$. Thus, at (x, y, z), a seeker satisfies the *height constraint* if and only if $\lambda_{low} \leq z - \mathcal{H}(x, y) \leq \lambda_{high}$.

We assume that the pursuit starts at time $t = 0$, and denote the position of seeker i at time t by $g_i(t)$. Let $\Pi_{\mathcal{H}}(g_i(t))$ be the (unique) point of the terrain vertically below $g_i(t)$, and let $\Pi_{xy}(g_i(t))$ be the orthogonal projection of $g_i(t)$ on the xy-plane. Similarly, for a polygonal curve γ, let $\Pi_{xy}(\gamma)$ denote its vertical projection onto the xy-plane. Consider a **configuration**, $\vec{v}(t) = \{g_1(t), g_2(t), \ldots, g_n(t)\}$ of seekers (for a fixed time t). We say that this configuration $\vec{v}(t)$ is a *valid configuration* if the following conditions hold:

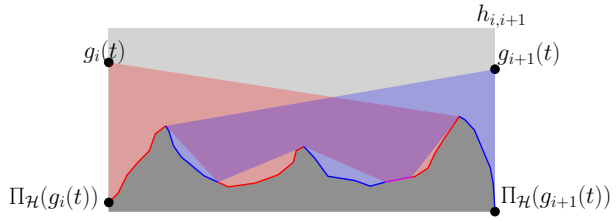

Figure 4: *The settings on the cross-section plane $h_{i,i+1}$ described in condition C2.*

C1: $\Pi_{\mathcal{H}}(g_1(t))$ and $\Pi_{\mathcal{H}}(g_n(t))$ are on the boundary ∂P of the pursuit region R.

C2: There is a path γ_i on the terrain connecting $\Pi_{\mathcal{H}}(g_i(t))$ to $\Pi_{\mathcal{H}}(g_{i+1}(t))$ and fully contained within $(\mathsf{Vis}(g_i(t)) \cup \mathsf{Vis}(g_{i+1}(t)) \cap h_{i,i+1}$, where $h_{i,i+1}$ is the plane normal to the xy plane containing both $g_i(t)$ and $g_{i+1}(t)$. See Figure 4.

Remark: Intuitively speaking, these two conditions implies that the seekers imposes a partition of the terrain into two (or more if the γ_is may be self-intersecting) region. The condition C2 implies that every point on the curve γ_i is seen by at least of $g_i(t)$ and $g_{i+1}(t)$. The requirement that γ_i is on the plane $h_{i,i+1}$ serves two purposes: (i) it makes the decisions performed by the algorithm (see description below) computationally tractable, and (ii) it also implies that the existence of a "border" that deforms continuously with time. Note also that if the visibility of a seeker has a limited range, then C2 also imposes a maximum distance between consecutive seekers.

Note that γ_i and γ_{i+1} shares an endpoint (for every i). Let $\gamma(t)$ be the path resulting from the concatenation of $\gamma_1(t), \gamma_2(t) \ldots \gamma_n(t)$. The rational behind our definition follows from the following lemmas:

LEMMA 1. *If $\vec{v}(t)$ is a valid configuration, $\gamma(t)$ is a connected curve whose endpoints lie on ∂P.*

LEMMA 2. *If the location $g_i(t)$ is a continuous function of t, and $\mathcal{H}$ is a continuous function then $\gamma(t)$ is a continuous function of t as well.*

PROOF. By definition, the projection $\Pi_{xy}(\gamma_i(t))$ of $\gamma_i(t)$ on the xy-plane is a line segment within R. Hence $\Pi_{xy}(\gamma(t))$ is a polygonal curve with $\leq n + 1$ vertices. As t varies, its vertices move continuously within P. □

We call the set of all locations $\vec{v}(t) = \{h_1(t), h_2(t), \ldots, h_n(t)\}$ for $t \in [0, T_{\mathsf{final}}]$ a *schedule* $\mathcal{V} = \{\vec{v}(t) | 0 \leq t \leq T_{\mathsf{final}}\}$ of the seekers. We call this schedule a *valid schedule* if

1. $\vec{v}(t)$ is a valid configuration for every t,
2. At $t = 0$, all seekers are at the lower left corner of P (*starting state*).
3. At $t = T_{\mathsf{final}}$ all seekers are at the upper right corner, and the total distance that $g_n(t)$ has covered in the clockwise direction (moving counterclockwise contributes a negative term), plus the total distance that $g_1(t)$ has covered in the counterclockwise direction is at least the length of ∂P (*ending state*).

CLAIM 3. *Assume a valid schedule $\vec{v}(t)$, $0 \le t \le T_{\text{final}}$ exists. Then the corresponding family $\{\gamma(t) \mid 0 \le t \le T_{\text{final}}\}$ changes continuously with time, always partitions the pursuit region into two not necessarily connected regions,* clean *and* contaminated. *At $t = 0$ the clean region is empty, and at $t = T_{\text{final}}$, the contaminated region is empty.*

Assume for every two "close enough" configurations $\vec{v}, \vec{u}$ we are given a cost function $cost(\vec{v} \to \vec{u})$, that depends only on $\vec{v} \to \vec{u}$. That is, the cost of the transition does not take the history leading to a configuration into account. Define the cost of $\mathcal{V}$ as the sum of costs of switching between states.

Problem Formalization: Find a valid schedule of minimum cost.

Note that he 'no-memory' property simplifies the search for a cheapest schedule. On the other hand, it excludes some types of cost functions such as inertia.

4. ALGORITHMS

4.1 General Framework

To allow a computationally tractable search, we discretize the search space by imposing an orthogonal grid M, so the (x, y) coordinates of each seeker are confined to be a grid point. Note that this grid is much coarser that grid representing $\mathcal{H}(x, y)$ (in the case of a TIN input). Such a grid and how the seekers are confined to it is depicted in Figure 1. The basic framework is quite simple. We first compute the *pairs graph* $G_P(M, E_P)$, whose vertices are the grid vertices of M, and two vertices u, v are connected by an edge if condition C2 is maintained. We create the set of valid configurations, compute the cost between every pair of configurations, and use Dijkstra's algorithm to find cheapest path from starting to ending states.

A simple cost function was used to keep the algorithm as generic as possible. It only models the dominant effect the terrain imposes, elevation change. The cost function can easily be modified to conform to specific applications, so that it is based on an accurate fuel model, the number of turns or probability of detection by adversary. The cost function used in this paper is defined as follows. For a path of the seeker from p to p' sum (or integrate) the positive elevation changes along this path. Thus, for an edge $(p, p') \in E_P$ representing a valid move of a seeker from point $p \in M$, to point $p' \in M$, we choose the best path in the neighborhood of the segment connecting p, p', and define $\Delta^+(p, p')$ to be the cost of this path. In particular, if the terrain elevation is given only at vertices of M, then $\Delta^+(p, p') = \max\{0, \mathcal{H}(p') - \mathcal{H}(p)\}$. For a pair of configurations $\vec{u}, \vec{v}$, we define

$$\mathsf{COST}(\vec{u}, \vec{v}) = \sum_{i=1}^{n} \{\Delta^+(p'_i, p_i) + C\|\Pi_{xy}(p_i) - \Pi_{xy}(p'_i)\|_\infty + C'\},$$

where $\vec{u} = \{p_1, p_2, \ldots, p_n\}$, $\vec{v} = \{p'_1, p'_2, \ldots, p'_n\}$, $\|p'_i - p_i\|_\infty$ is the L_∞ distance between them, and C, C' are small constants. The rationale is that the fuel requirement of (positive) elevation gain dominates the other factors.

The cost function treats each seeker's movements independently; therefore a pre-compute step can be performed using one seeker for all possible transitions satisfying C2. This reduces the number of computations by replacing the need to calculate the transition of each seeker in a configuration transition with a table lookup.

4.2 Further Assumptions and Implementation Details

In our preliminary implementation, several simplifying assumptions are made, to help reduce the search space. We replace **C2** by **C2'** which is simplified.

C2': The horizontal distance between $g_i(t)$ and $g_{i+1}(t)$ cannot exceed some predetermined distance D.

To further restrict the search, we express bounds on the maximum velocity, by limiting the L_∞ distance $\max\{|x - x'|, |y - y'|\}$ where (x, y) and (x', y') are two consecutive projections on the xy-plane of a single seeker. Without loss of generality, we assume the maximum horizontal L_∞ distance a UAV might cover at each step is D times a small constant (taken as 2 in our implementation). Hence, a seeker can validly transition to 24 neighboring points if $D = 2$. To be precise, our algorithm considers the graph $G(V, E)$ where V represents the set of all valid configurations, and E represents all valid transitions between configurations. We assume that the edge connecting two valid configurations $(\vec{u}, \vec{v}) \in E$ if there is a valid transition from $\vec{u}$ to $\vec{v}$. Finally we assume that $\lambda_{low} = \lambda_{high}$, so there is one specific height λ above ground level which is optimal for the search.

It is important to note here that the number of possible valid configurations in the graph can be quite large, easily exceeding the capacity of the cache of state-of-the-art desktops. Hence we cannot store the entire configuration space explicitly in main memory, causing paging issues. In Section 4.3, we explore approaches to reduce the search space.

The unit distance is chosen using multiple factors: size of the visibility regions, size of terrain features, number of seekers needed and the granularity of movement required. If the unit distance is too large to accurately match details of the topology or if the area to be searched is too large then the entire region can be broken into sub-regions. Each sub-region can then be searched independently in some order that matches operational requirements.

First, we explain how to compute the cost of a single seeker movement from point $p \in M$ to a point $p' \in M$. Recall that M is sparse so different trajectories might lead between these points. We use Dijkstra to find the cheapest trajectory (of this seeker alone), among the paths that fully lies within the disk whose diameter is $\Pi_{xy}(p), \Pi_{xy}(p')$. We repeat this process for all pairs of points p, p' for which are valid.

Next, we describe the search in the configurations graph. Due to the combinatorially huge number of configurations several modifications have been made: the transition graph is not pre-computed but built "on the fly" which is equivalent to declaring all transition costs in the complete graph to infinity (as in the original Dijkstra algorithm); additional methods were used to "branch-cut" the transition graph to reduce the number of configurations the algorithm has to process; and records in the solution graph were written to the disk to limit the amount of memory used. Following the standard Dijkstra framework, we maintain during the algorithm a set S of configurations to which the optimal path is found. We loosely follow the notation of [6]. We use a hash table H to store discovered configurations. Some of the elements of S itself are stored in H, while others (details provided below) are stored in a B-tree file F.

For each configuration in H or F, we store the following: seekers' locations, its cost, its parent, whether it is in S and some fields to aid data structure management, see Figure 5.

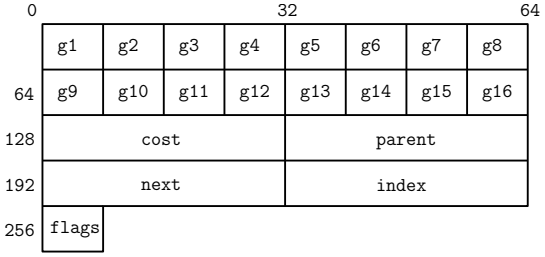

Figure 5: Record for configuration when inserted into transition graph. Units are in bits.

The hash function used was

$$h(\vec{u}) = \left(\sum_{i=1}^{n} A_i u_i[x] + B_i u_i[y] \right) \mod H_{\text{Length}},$$

Here $u_i[x]$ and $u_i[y]$ are the coordinates of seeker i in the configuration $\vec{u}$ and A_i, B_i, and H_{Length} are all prime numbers. It uses the configurations' locations in the terrain graph as the key into the hash table. The minimum priority queue Q used by Dijkstra is implemented as a heap, where each element in Q points to its parent. The implementation closely follows the description in [6]. Additionally, configurations are removed from the hash table H when the storage limit is reached. This entry "settle" attempts to minimize the in-memory size of H, storing entries already in S in file F. This is done because the in-memory requirements become quite large with a modest number of seekers, see Section 4.3 for further discussion on storage requirements. This idea was adapted from [15]. To mark a configuration $\vec{u}$ for removal from H it must: (i) be in solution set S, and (ii) have all its neighbors in S. Hence, for a configuration $\vec{u}$, the number $C[\vec{u}]$ of neighbors of $\vec{u}$, not in S, is tracked for each configuration as the algorithm progresses. When the count reaches 0, the configuration is removed from H and added to the file. The file is needed for efficient retrieval of the optimal path when the algorithm terminates. Care is taken to minimize its representation in memory, to reduce the need for paging.

The record placed in the transition graph can be seen in Figure 5. It was defined as a packed `C` struct with a pre-allocated maximum number of 16 seekers. The first 128 bits was used to identify the coordinates of the seekers. The next 64 bits were used to record the cost of the configuration and the configurations parent, respectively. The next 32 bits was to aid in storage of the record in the hash table. The next 32 bits were used to store the index of the record in the minimum priority queue. This aided in the `DECREASE_KEY()` function. The last 8 bits were used to also aid in Dijkstra's algorithm by determining if the configuration was in the solution set.

When the pre-calculation and initialization steps complete, the algorithm then transitions into search (see Procedure 1). When the search is started the special $\vec{s}$ configuration is added to the transition graph, which is implemented using a hash table H, and its cost is set to 0. This configuration is then added to the minimum priority queue Q.

The `SETTLE` function (Procedure 2) marks each configuration to be removed from Q with a count of transitions that do not connect back into S, i.e. all neighbors of $\vec{u}$ that are not in the solution set yet. Then for each configuration transition the count is updated based on which configurations are already in S. When the count reaches 0, this signals removal from the hash and addition to the file.

Procedure 1 `SCHEDULE_GENERATOR`$(G, \vec{s}, \vec{t})$: Main search function to find optimal schedule.

```
Q ← φ and H ← φ
INSERT(Q, s⃗, 0) and ADD(H, s⃗)
while Q ≠ φ do
  u⃗ ← TOP(Q) and S[u⃗] ← true
  if u⃗ = t⃗ then
    return solution
  N ← GENERATE(u⃗) {get list of transitions}
  C[u⃗] ← |N ∉ S|
  for each v⃗ ∈ N do
    if v⃗ ∈ S then
      SETTLE(v⃗, File, H)
    else
      RELAX(u⃗, v⃗, Q)
```

Procedure 2 `SETTLE`$(\vec{v}, File, H)$: Removal of Configuration from Hash table when all neighbors enter solution table.

```
C[v⃗] ← C[v⃗] − 1
if C[v⃗] = 0 then
  ADD(File, v⃗)
  REMOVE(H, v⃗)
```

The transition graph is not pre-computed when the algorithm starts and hence the relax step of Dijkstra's algorithm is changed from the original and contains minimum priority queue calls to maintain Q; see Procedure 3.

Procedure 3 `RELAX`$(\vec{u}, \vec{v}, Q)$: Relaxation of neighbors of input configuration.

```
if v⃗ ∉ Q then
  INSERT(Q, v⃗, d[u⃗]+ COST(u⃗, v⃗))
  π[v⃗] ← u⃗
else if d[v⃗] > d[u⃗]+COST(u⃗, v⃗) then
  DECREASE(Q, v⃗, d[u⃗]+COST(u⃗, v⃗))
  π[v⃗] ← u⃗
```

The key to this algorithm is the `GENERATE` procedure contained within the search; see Procedure 4. It searches for all transitions available to a given configuration. The current implementation only moves one seeker at a time when discovering transitions. While it might seem more sensible to allow each configuration to find the transitions without this constraint, it turns out that this method, while presenting more configurations to the search algorithm, requires fewer I/O operations while not narrowing the search space or jeopardizing optimality. The largest contributor to runtime is the page swapping that occurs when the number of configurations stored in H exceeds the amount of main memory. This leads the search for efficient I/O and data storage techniques to eliminate the need to keep all configurations in memory, rather than optimize the `GENERATE` procedure.

4.3 BRANCH CUTTING

We implement several types of filtering (branch cutting) techniques to reduce the number of configurations that need

Procedure 4 `GENERATE(`$\vec{v}$`)`: Generate the neighbors of an input configuration.

```
N ← φ
for each p ∈ v⃗ do
  v⃗′ ← replace p with p′ in v⃗
  if VALID(v⃗′) then
    d[v⃗′] ← ∞
    if v⃗′ ∉ H then
      ADD(H, v⃗′)
    N ← N ∪ v⃗′
return N
```

not be considered, thus reducing the search space.

Self-Intersections: The first type of filtering is based on the following observation. Let $P(t)$ be the polygonal path obtained by connecting the projections on the xy-plane of the j^{th} and $(j+1)^{th}$ seekers at time t.

OBSERVATION 1. *If $P(t)$ is simple (does not have two crossing edges) then, possibly after re-labeling the seekers, with high probability, $P(t+1)$ is crossing free as well.*

The basis for this observation is the following lemma which pertains to the continuous case when the seekers are not restricted to lie on the grid and we consider a continuous elevation map of P. We also do not consider the limitation on the velocity of the seekers. In a schedule consider two configurations $\vec{v}(t)$ and $\vec{v}(t+k)$ and assume that at time t, $P(t)$ is not self-intersecting. Also, let $P(t+k)$ be self-intersecting. Under these conditions.

LEMMA 4. *There exists a configuration $\vec{w}(t+k)$ corresponding to the seekers' positions at time $t+k$ such that the same region is swept and*

$$\mathsf{COST}(\vec{v}(t), \vec{v}(t+k)) \geq \mathsf{COST}(\vec{v}(t), \vec{w}(t+k)).$$

PROOF. Let there be a crossing between the movements of two seekers i and j between the two configurations. Let the intersection point of the two trajectories be the point p. Now, moving seeker i from $h_i(t)$ to p and then continuing to $h_j(t+k)$ in $\vec{v}(t+k)$ and moving seeker j from $h_j(t)$ to p and continuing to $h_i(t+k)$ results in a configuration $\vec{w}(t+k)$ with no crossings. Clearly, the cost of reaching $\vec{w}(t+k)$ is the same as that of the costs of the original trajectories. Thus, the lemma is proved. □

Remark: Note that the correctness of this lemma resulted from the symmetry between the seekers — to follow a specific trajectory, each of them needs the same amount of resources. In contrast, if different seekers have different capabilities, then, during the optimal schedule two seekers might switch places to enable, say, a seeker that is short on fuel to avoid a particularly hilly part of the terrain.

Based on Observation 1, we can constrain the search to consider only configurations with no self-intersections. Section 5 includes results showing the computational reductions.

Pinching: A different but related part of configurations are the ones demonstrating **pinching** defined as the case when some $h_i(t), h_j(t)$, for $j > i+1$ are located above the same grid-point, while $h_{i+1}(t)$ is located elsewhere.

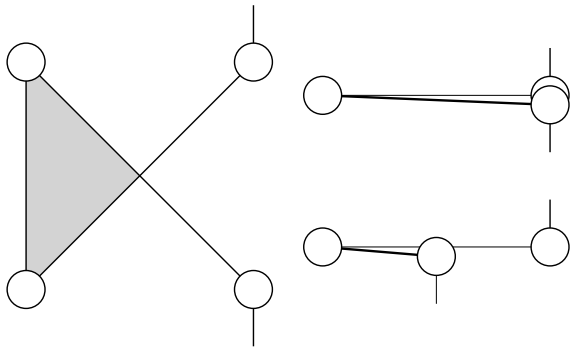

Figure 6: Prohibited Configurations, (Left) crossing seekers making a non-simple path, (Right) pinching

A second class of branch cutting was added to reduce the number of configurations stored at the hash table entry portion of algorithm. Figure 6(a) illustrates a crossing between configurations that are non-simple and produce a redundant clean region colored gray. Even so, crossings can be advantageous. During a crossing, seekers can be swapped and relabeled, in an attempt to rebalance the energy consumption of the chain. This reconfiguration is assumed, given the cost function, which minimizes energy of the sum. Hence, these configurations are removed from the possible configurations allowed. Figure 6(b) illustrates pinching configurations that add no area to the "clean" side.

A natural question is whether guards should be allowed to move backward. That is, for a configuration $\vec{u}(t)$ at some time t, for any Δt, it is tempting to constrain $\vec{u}(t+\Delta t)$ such that the paths $\gamma(t)$ and $\gamma(t+\Delta t)$ do not cross. We call such configurations "crossing" configurations. This implies that no guards of $\vec{u}(t)$ can move "backward" so that some of the clean region becomes contaminated. Note that the paths $\gamma(t)$ and $\gamma(t+\Delta t)$ may still have overlapping portions. The author's intuition is that this branch cutting heuristic would not jeopardize the quality of the solution for most realistic scenarios. However, cases exist where this claim is definitely not true. In particular,

THEOREM 5. *There is a terrain that that could be searched by 4 seekers flying at elevation λ, but if we do not allow crossing configuration, then at least one of the seekers should reach higher elevation.*

The proof is removed, due to lack of space. It is related to results on searching polygons by two-guards [14].

5. EXPERIMENTAL RESULTS

The algorithm was coded in `C++` using gnu `C++` and used Berkeley DB[2] for storage of settled configurations. It was run on an Intel Core i7 860 @2.80 GHz with 8 GB of RAM running Ubuntu Linux 10.04. Examples of our algorithm can be viewed at `http://www.cs.arizona.edu/people/pmyers/mbrain`. To keep the runtime of this algorithm reasonable the grid size was limited to a 16×16 and a maximum of 7 seekers. Valid distance D was set to 2 grid points. The fixed cost constant C' was set to 0.01 units. To establish good

[2]Berkeley DB is an efficient and cache-aware B-tree data structure, used as a fast and long-term storage for stored configurations

visibility by each UAV, we would assume that it is above ground. So we assume that its flying trajectory should be set so its height is 1 unit distance above the terrain.

5.1 Synthetic Terrains

To demonstrate the behavior of the algorithm we have generated 4 synthetic terrains (see Figures 8 though 11): (i) a set of parallel ridges that extends horizontally as shown in Figure 7 (**horizontal ridges**), (ii) a set of parallel ridges that extends vertically (**vertical ridges**), one M-shaped ridge (**M-shaped ridge**), and (iv) a pair of ridges that are diagonal to the direction of motion (**diagonal ridges**). In each case, the graph M with no accompanying fine grid δ is constructed "by hand". The cost function is then modified to only operate on the coarse graph. The elevation of all the large unconnected discs represents ridge peaks with a height of 3 units. The smaller discs represent the "foothills" associated with the ridge peaks, with an elevation of 1.5 units. The portions of the grid with no discs are at "sea-level" with zero elevation. The connected blue discs represent configurations, with a few snapshots of optimal subplots (a) through (f) showing key configuration transitions.

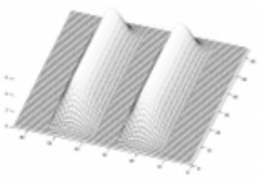

Figure 7: A synthetic terrain consisting of two parallel ridges that extend horizontally.

In each example the configuration starts in the lower left hand corner of the graph and expands to align itself with the terrain feature. The alignment is dictated by the cost equation that attempts to minimize elevation changes. This cost equation represent our simplified fuel model. Each execution attempt to limit the number of positive elevation changes each seeker makes, in essence attempting to reduce fuel usage, and lengthen operational time of the search. Because the terrain features in these examples cannot be avoided by the configuration search, it minimizes the number of seekers that must "climb" up those features. Hence, what occurs is that some of the seekers get elected to climb onto the terrain ridges, allowing the others to move around them. This optimizes the energy use of the set of seekers, and subsequently maximizing operation time. When the terrain feature has been searched, the configuration then collects at the upper right hand corner of the graph.

In Figure 9, four seekers have been "elected" to climb the foothills rather than one per peak. It turns out that this solution has the same cost as a single seeker climbing each peak, as the foothills are exactly half the height of the peak. Hence the result is still optimal, given the cost function.

Figure 12 demonstrates that a naive schedule where each seeker climbs to the ridge height is not optimal. This for example would be the case if they are all restricted to move along a straight line. Instead, in Figure 12, only two seekers (and one edge connecting them) climb above the minimum elevation, and even these two reach only the height of the foothill (depicted by medium-dized disks). This is an interesting example showing that to reach optimality, some seekers must stay still whlie other move. It also shows that the criteria for conclusion (that the terrain is clean) must be based on on the total signed distance the seeker moves, rather than just the meeting point on the upper right corner.

The run-time of the algorithm was also examined, as seen in Tables 1 through 4. Because of the branch-cutting used to limit the number of configurations the search procedure "sees", rows are shown with and without the optimizations. The tables show the number of configurations added to the hash table, how many configurations were marked for storage as a result of `SETTLE`, and the execution time.

5.2 Real Terrain

The terrain data was gathered from the ASTER Global Digital Elevation Model Explorer[19]. It provides DEM[3] data at 30 meter resolution. See Figure 13. The configuration starts in the lower left hand corner of the graph, aligns itself with the terrain feature and begins cleaning the map region. When the terrain feature has been cleared, the configuration then collects in the upper right hand corner of the map. This result is consistent with the synthetic terrain solutions explained in the previous section.

6. EXTENSIONS AND IMPROVEMENTS

6.1 Further Approaches

To use all of the benefits of the the visibility condition C2 from Section 3, we may precompute all valid pairs of vertices of M. This could be obtained extremely fast with the use of GPU and shader programming.

Our solutions assume a "memory-less" setting – the cost of reaching from a specific configuration $\vec{u}$ to another, does not depend on the cost of reaching $\vec{u}$. This limits the algorithm, since it cannot take into account inertia and states of the vehicles performing the search. It seems that a simple dynamic programming algorithm could overcome this barrier, but this is left for future study.

One might prefer to seek solutions optimizing multi-criteria cost functions, rather than a single scalar. For example, to find the most fuel-efficient search but whose duration is bounded by a threshold. Similarly, one might seek a schedule that minimizes the maximum positive elevation any seekers gain. As a rule, multi-criteria shortest path problems are known to be NP-Hard, but efficient approximation techniques exist [20]. It would be interesting to see how they perform in our setting. One might also seek other cost functions that the one we have experimented with. For example, a natural cost function is the maximum of the total elevation gain of any UAV, rather than the sum. Small maximum elevation gain implies a longer duration of the whole swarm before refueling/recharging is needed. This problem is closely related to *multi-criteria* shortest path problem which is also NP-Hard but for which approximation techniques exist [18]. This line of work is left as one possible future study.

6.2 Ridge-Driven Algorithms

We consider the ridges of a terrain as guidelines to seekers' trajectories, i.e., we look of a schedule at which each seeker flies over a ridge. This is reasonable, since any point on the ridge sees more than any other in its vicinity. We still use the previous assumptions — seekers form a chain whose

[3]Digital Elevation Model: a grid of elevation values indexed by latitude and longitude used to describe a section of terrain.

Table 1: Horizontal ridges on a 9×9 grid with 7 seekers (Figure 9 Left).

Branch-cutting	# config.	# stored	Time (s)
None	82,292,560	5,269,161	613.15
No self intersection	65,336,071	4,984,307	661.03
No pinching	330,379	132,535	2.37
No pinching nor self-intersection	329,963	132,592	3.68

Table 2: Vertical ridges on a 9 × 9 grid with 7 seekers (Figure 9 Right).

Branch-cutting	# config.	# stored	Time (s)
None	83,285,762	5,271,094	628.18
No self intersection	65,337,247	4,987,228	618.23
No pinching	287,775	124,795	2.23
No pinching nor self-intersection	286,303	124,714	2.40

Table 3: M-shaped ridge on a 15 × 6 grid with 5 seekers (Figure 10).

Branch-cutting	# config.	# stored	Time (s)
None	440,787	90,327	2.14
No self intersection	338,457	83,183	2.05
No pinching	17,156	8,704	0.08
No pinching nor self-intersection	17,156	8,704	0.08

Table 4: Diagonal ridges on a 15 × 6 grid with 5 seekers (Figure 11).

Branch-cutting	# config.	# stored	Time (s)
None	470,754	121,179	2.55
No self intersection	430,144	112,624	2.52
No pinching	19,687	16,172	0.13
No pinching nor self-intersection	19,687	16,172	0.14

trajectories are coordinated so conditions C1 and C2 from Section 3 are present. So we are left with the need to design an optimal algorithm for this subproblem — *which seeker follows which ridge*. The main reason to use ridges as a leading term is that their order is preserved independent in the location — two ridges could not cross each other. Hence it suits well to dynamic programming techniques.

Algorithm Sketch: We note that in many cases, the ridges of the terrain form a topological tree, or at least a small number of disjoint trees. Consider one such tree $\mathcal{T}$, whose leaves are (from left to right) $v_1 \dots v_m$ and assume $n < m$. We need a leaf assignment for each one of the seekers $g_1 \dots g_n$ which respects their order; if g_i is assigned to leaf v_j, then seeker g_{i+1} is assigned to one of the leaves $v_{j+1}..., v_m$. The seeker then would follow the ridge to the root of $\mathcal{T}$ (while satisfying conditions C1, C2 together with the other seekers).

As is usually the case for dynamic programming algorithms, we maintain a table T_k of size $m \times m$ for each $k = 1, \dots, n$, where $T_k[i, j]$ is the cheapest schedule that uses k seekers and cleans the ridges ending at $v_i \dots v_j$. To compute $T_k[i, j]$, we assume recursively that the solutions to all table entries $T_{k'}[i', j']$ for which $|j' - i'| < |j - i|$ and $k' < k$ have been computed, and sum the cost. Details are obvious and omitted. We finally return $T_n[1, m]$. It is not hard to see that the total running time is $O(n^3)$.

7. CONCLUSION

In this paper, we propose a different approach to the pursuit-evasion problem on terrains. This approach provided an algorithm to plan the trajectories of a chain of seekers so that an evader is located with high probability. Branch cutting and I/O-efficient data structures were explored to decrease the run-time of the algorithm. Experimental results were presented for both synthetic and real terrains. This is the start of exploration in this area and there are numerous ways to extend our algorithm: (i) parallelization or "distributization" of the algorithm; exploring distributed algorithms would be helpful in enabling networked UAVs to execute the search algorithms, (ii) utilization of other search algorithms such as A^*-search, (iii) cost functions that match other operational constraints and possible operation-specifc optimizations, (iv) more elaborate UAV models that calculate fuel usage with greater accuracy, and (v) automation of the terrain graph based on optimization parameters.

We have also proved important characteristics of the optimal solution, enabling to maintain the search space under control. In addition, we have proposed a much faster polynomical-time algorithm that uses ridges of the terrain to guide the search.

8. REFERENCES

[1] B. Alspach. Searching and sweeping graphs: a brief survey. *Le Matematiche*, 59(1, 2):5–37, 2006.

[2] T. Baumgartner, S. P. Fekete, A. Kröller, and C. Schmidt. Exact solutions and bounds for general art gallery problems. In *Proc. 2010 SIAM Workshop on Algorithm Engineering and Experiments*, pages 11–22, 2010.

[3] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 515–524, 2005.

[4] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proc. 21st International Joint Conference on Artificial intelligence*, pages 59–66, 2009.

[5] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *J. Symb. Comput.*, 7(1):11–30, 1989.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[7] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *Int. Trans. Oper. Res.*, 18(4):425–448, 2011.

[8] A. Efrat, L. Guibas, S. Har-Peled, J. S. B. Mitchell,

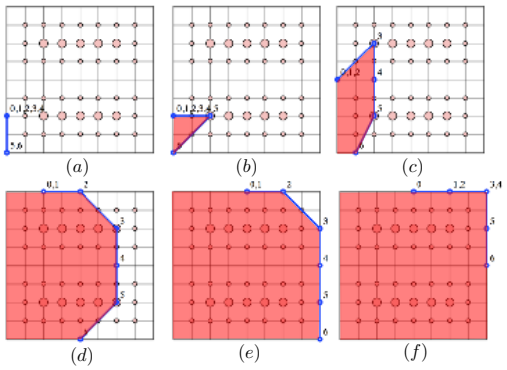

Figure 8: *Intermediate steps of the results of our algorithm for sweeping terrains consisting of horizontal parallel ridges. The large disks indicate an elevation of 3 units and the smaller disks 1.5 units. (a) seekers start moving up to align to ridges, (b) one seeker moves to the ridge top, (c) a second guard moves to the ridge top, (d) the expanded configuration moves across the terrain graph along ridge tops and valleys, (e) both guard move to boundary to finish search, (f) entire search area has been swept.*

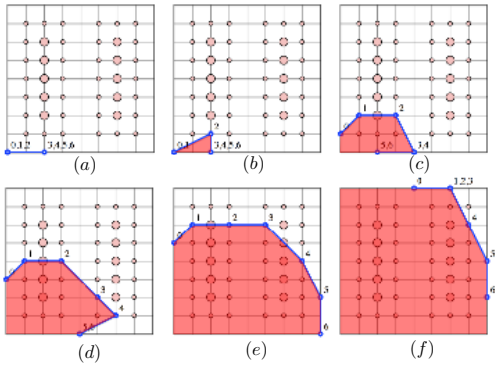

Figure 9: *Sweeping a terrain consisting of vertical parallel ridges. Note the difference in the sweeping pattern from Figure 8. (a) seekers start expanding right to align with ridges. (b) one seeker moves to the first ridge, (c) two seekers have elected to straddle the first ridge, (d) a second set of seekers move to the second ridge, (e) the expanded configuration moves along the ridge line, (f) most of the area has been swept and the seekers are collecting in the upper right hand corner.*

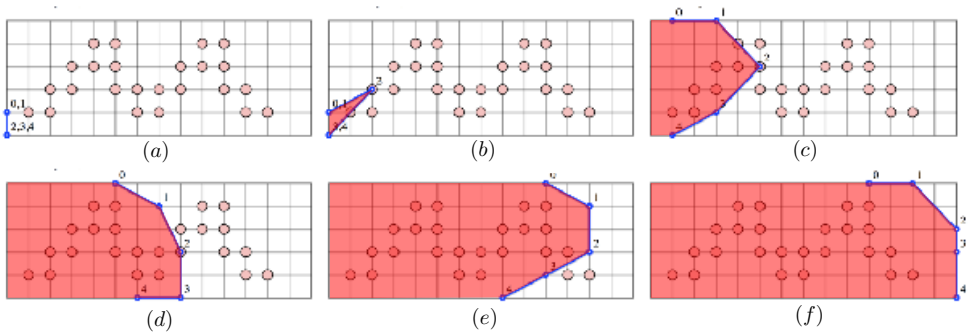

Figure 10: *Sweeping a terrain consisting of one wiggly shaped ridge, (a) the seekers expand up to align with ridge, (b) one seeker moves onto the ridge, (c) the expanded configuration moves along the ridge with only 1 seeker on the ridge, (d) the search continues with only one seeker on the ridge and the expanded configuration moving in coordination, (e) seeker 2 moves off of the ridge and moves down to allow pursers 3 and 4 to sweep around the ridge rather than moving up the ridge, (f) most area has been swept and pursers are collecting in upper right corner.*

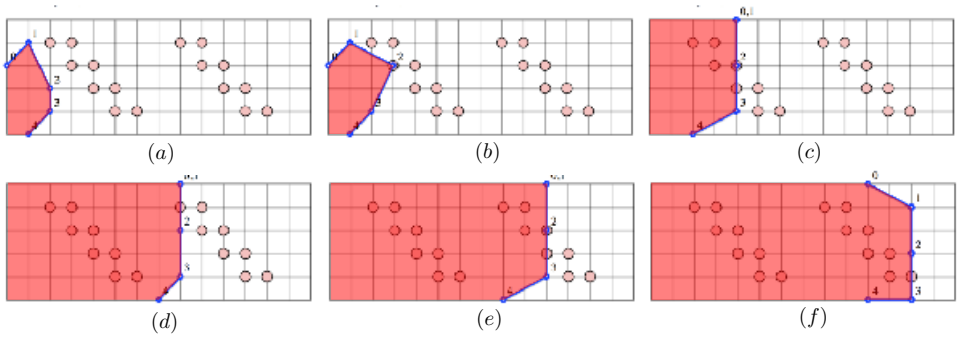

Figure 11: *Sweeping a terrain consisting two diagonal ridges, (a) seekers expand to align with ridge, (b) one purser moves onto the ridge, (c) the expanded configuration moves right with only one seeker on the ridge, (d) the configuration aligns with seconds ridge, (e) the expanded configuration moves right with only one seeker on the second ridge, (e) seeker 2 moves off of the ridge and down to allow seekers 3 and 4 to move around the ridge instead of moving onto the ridge.*

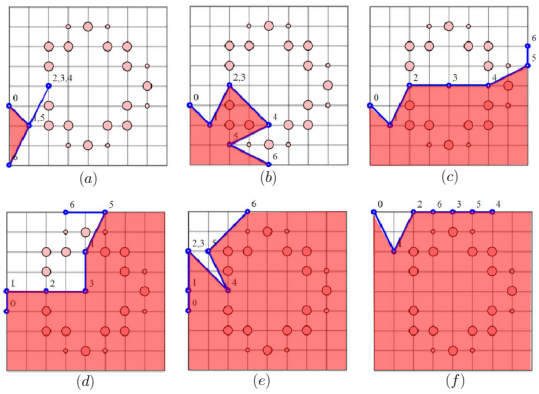

Figure 12: *An interesting example where a naive approach would be suboptimal. Note that seeker 2 is stationary for most of the schedule and the rest of the seekers' chain "revolves" around 2.*

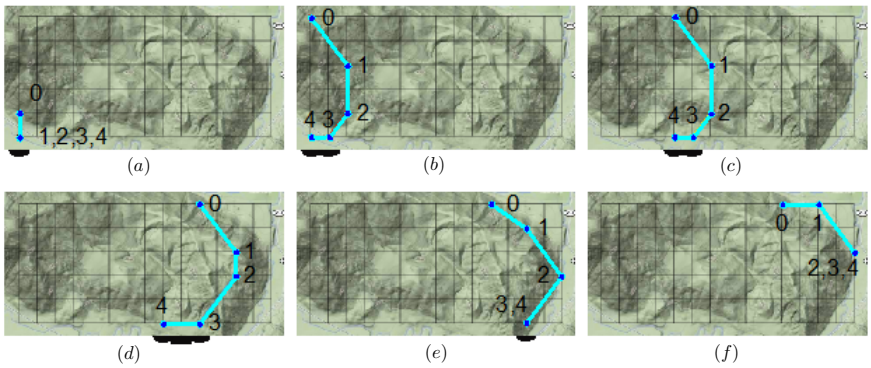

Figure 13: *Real terrain located in Washington State: (a) configuration detects ridge orientation and starts expanding North, (b) with configuration fully expanded it starts moving east, (c) continues east with full expansion, (d) end of ridge is being approached and configuration aligns with it, (e) alignment in preparation to collect all guards in North-East corner, (f) fully off ridge and collection of seekers occurring. This example can be seen at* `http://www.cs.arizona.edu/people/pmyers/mbrain`.

and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete Comput. Geom.*, 28(4):535–569, 2002.

[9] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[10] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[11] O. Franklin and C. Vogt. Multiple observer siting on terrain with intervisibility or lo-res data. In *In XXth Congress, International Society for Photogrammetry and Remote Sensing*, pages 12–23, 2004.

[12] W. R. Franklin and C. K. Ray. Higher isn't necessarily better: Visibility algorithms and experiments. In *Advances in GIS Research: Sixth International Symposium on Spatial Data Handling*, pages 751–770. Taylor & Francis, 1994.

[13] L. Guibas, J.-C. Latombe, S. Lavalle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 17–30. Springer Berlin / Heidelberg, 1997.

[14] C. Icking and R. Klein. The two guards problem. *Inter. J. Computational Geometry and Applications*, 2(3):257–285, 1991.

[15] R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier search. *J. ACM*, 52(5):715–748, 2005.

[16] S. LaValle, D. Lin, L. Guibas, J.-C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *Proc. 1997 IEEE International Conference on Robotics and Automation*, pages 737–742, 1997.

[17] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, and C. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.

[18] S. Misra, G. Xue, and D. Yang. Polynomial time approximations for multi-path routing with bandwidth and delay constraints. In *Proc. 28th IEEE Conference on Computer Communications*, pages 558–566, 2009.

[19] USGS. Aster gdem explorer. `http://demex.cr.usgs.gov/DEMEX/`.

[20] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman. Polynomial time approximation algorithms for multi-constrained qos routing. *IEEE/ACM Trans. Netw.*, 16(3):656 –669, 2008.