Pursuit-Evasion Problems

Richard Borie
University of Alabama
July 18, 2005

Joint work with:
Craig Tovey, Georgia Tech
Gary Parker, Georgia Tech
Sven Koenig, Southern Cal

Initial example
- Graph has 2 vertices \(x, y\) and 3 parallel edges (tunnels)
- We wish to find a thief as quickly as possible
- Number of available police robots is 3
- A police robot can detect a thief within 2 meters in any direction from its center
- A thief can move much faster than a robot
  - Assume the thief travels at infinite speed
  - Thief is unable to travel through a robot’s detection region
- Tunnel 1 is at most 4m wide, so one robot can scan it
- Tunnels 2 and 3 are each between 4m and 8m wide everywhere, so two robots working together can scan either tunnel

Solution to example
- Initially place all 3 robots at one vertex (say \(x\))
- Robot 1 stays at \(x\)
- Robots 2 and 3 scan tunnel 2
- Robot 2 stays at \(y\)
- Robot 3 scans tunnel 1
- Finally robots 1 and 3 scan tunnel 3
- Note: two robots are not sufficient, even if all three tunnels were only at most 4m wide

Some applications
- Searching for people that are deliberately or accidentally evasive
  - Clearing a museum of people who try to hide inside when it closes for the day
  - Finding confused elderly folks or children that wander off
  - Capturing criminals who are fleeing either by vehicle or on foot
  - Finding terrorists that might be hiding in a cave system

Modeling the problem
- Graph \(G = (V, E)\)
- Function \(g: E \cup V \rightarrow Z\)
  - \(g(e)\) denotes the minimum number of robots or humans needed to scan edge \(e\)
  - \(g(v)\) denotes the minimum number of robots or humans needed to monitor vertex \(v\)
- Determine the minimum number of robots needed to clear \(G\), and a feasible plan
Modeling the problem (cont.)

- The situation at vertex $v$ may actually be more complex than what is represented by $g(v)$
- In principle we could ask, for each subset $S$ of edges incident on $v$, how many robots are needed to guard the entrances to $S$
- In the future we might redefine function $g : E \cup (V \times 2^V) \rightarrow \mathbb{Z}$, but not today

Adding a time dimension

- Function $h : E \rightarrow \mathbb{Z}$
  - $h(e)$ denotes the amount of time that $g(e)$ robots need to scan edge $e$
  - Might be an inadequate definition of $h(\cdot)$ since more robots might be able to scan edge $e$ faster
- So could redefine function $h : E \times \mathbb{Z} \rightarrow \mathbb{Z}$
  - $h(e, k)$ denotes the amount of time that $k$ robots need to scan edge $e$
  - Value is $\infty$ or undefined if $k$ is too small
- Given $k$ robots, determine a plan that clears $G$ in minimum possible time

Other problems and models

- Some variations include:
  - Non-uniform robot scanning capabilities or speeds
  - Minimizing mean or expected seek time
  - Minimizing total fuel expended or distance traveled
  - Fixed or variable initial locations of the robots
  - Evader has known maximum speed
  - Given a probability distribution on the thief's location, maximize the probability of detection within a given time limit
  - Combinations of the above and others

A special case

- Suppose the graph is a tree
  - We may choose any vertex as the root
- All robots, edges, and vertices have unit width
  - So each $g(e)=1$ and $g(v)=1$
- Initial robot locations are arbitrary or chosen by an adversary
- Goal: minimize the number of robots needed to clear the tree of all evaders

Justification

- Many other variations of the pursuit-evasion problem are generalizations of our special case
  - Most useful graph classes include trees
  - The minimum time or total distance will be finite iff the number of robots is sufficient
- So the solutions to many other problem variations must generalize (or subsume) our solution

Some small instances

- A tree can be cleared by one robot iff it is a path
  - Let $T_1$ denote the tree with two vertices and one edge
- The smallest tree that requires two robots is a star with 4 vertices and 3 edges
  - Call this tree $T_2$
Claims (proofs omitted)

- Let $T_r$ denote the smallest tree that requires $r$ robots.
- For $r \geq 2$, $T_r$ can be built by taking three copies of $T_{r-1}$ and fusing together one leaf from each copy.
- $T_r$ has $n=3^{r-1}+1$ vertices.
- The number of robots required for any tree with $n$ vertices is $r \leq 1+\log_3(n-1)$, so $r = O(\lg n)$.

More claims

- If a given tree $T$ can be cleared by $r$ robots, then $T$ can be cleared by $r$ robots in such a way that at any given time, all robots lie along a common path.
- A tree $T$ can be cleared by $r$ robots iff it contains a path $P$ such that splitting each degree-$d$ vertex along $P$ into $d$ vertices of degree 1 produces a forest of trees that can each be cleared by $r-1$ robots.
- Intuition: the minimum $T_r$ trees correspond to the case when each path $P$ has only one vertex.

Algorithm 1

- Perform a post-order traversal of $T$, and label each vertex of $T$ as follows:
  - If $v$ is a leaf then $R(v) \leftarrow 1$.
  - Otherwise $v$ has $k \geq 1$ children $c_1, \ldots, c_k$.
    - Let $x \leftarrow \max \{R(c_1), \ldots, R(c_k)\}$.
    - If $x = R(c_i)$ for only one child $c_i$, then $R(v) \leftarrow x$.
    - Otherwise $R(v) \leftarrow x+1$.

Analysis

- Algorithm 1 runs in $O(n)$ time.
  - At each vertex $v$, $R(v)$ can be computed in time proportional to the number of children of $v$.
  - The total number of children over all vertices is of course $n-1$.

Correctness (sufficiency only)

- $T$ can be cleared using $R(\text{root}(T))$ robots:
  - Initially move all robots to vertex $v = \text{root}(T)$.
  - Recursively, use $R(c_i)$ robots to clear the subtree rooted at each child $c_i$ of $v$.
  - However, if $R(v) = R(c_i)$ for some child $c_i$, then the subtree rooted at this $c_i$ must be cleared last.
  - Note that there can be at most one such child $c_i$.
  - Eventually one robot will reach each leaf.
  - When the current subtree rooted at $c_i$ is clear, then the $R(c_i)$ robots move upward to $v$ before clearing the next subtree of $v$.

Non-optimality

- Counterexample
- Algorithm 1 yields 3 robots.
- But 2 robots would be optimal (How?)
Non-optimality (cont.)

- Algorithm 1 is not optimal because it restricts all paths $P$ (mentioned earlier) to be sub-paths of root-to-leaf paths.
- Also, because it restricts the robots to always lie along a common root-to-leaf path at any given time.

Another counterexample

- Algorithm 1 yields 5 robots.
- But 3 robots would be optimal (How?)

Approximation ratio

- Algorithm 1 is 2-approximate.
  - Consider any path $P$ in the optimal solution that does not lie along a root-to-leaf path.
  - Let $w$ be the vertex of $P$ that is nearest the root.
  - Split $P$ into two subpaths $P_1$ and $P_2$ at $w$.
  - Place one robot at $w$.
  - Use a second robot to traverse $P_1$.
  - Also use this same second robot to traverse $P_2$.
  - Algorithm 1 can find this solution, using two robots where only one robot should be needed.

Preview

- An optimal algorithm for this pursuit-evasion problem on a tree will be described next time.