An Optimal Algorithm for Pursuit-Evasion on a Tree

Richard Borie
University of Alabama
August 1, 2005

Recall: the problem
• Suppose the graph is a tree
• All robots, edges, and vertices have unit width
  - \(g(e)=1\) and \(g(v)=1\)
• Initial robot locations are arbitrary
• Goal: minimize the number of robots needed to clear the tree of all evaders

Recall: the problem (cont.)
• Algorithm 1 (which was discussed last time) is 2-approximate and runs in \(O(n)\) time

Recall: some critical trees
• \(T_1\) is a tree with 2 vertices and 1 edge
• \(T_2\) is a star with 4 vertices and 3 edges
• For \(r \geq 2\), \(T_r\) is built from three copies of \(T_{r-1}\) by fusing together one leaf from each copy
• For \(r \geq 2\), \(T_r\) is the smallest tree that requires \(r\) robots
  • Each \(T_r\) 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 \(r = O(\lg n)\)

Recall: some critical trees (cont.)
• If \(T\) 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
• \(T\) can be cleared by \(r\) robots iff \(T\) contains a path \(P\) such that splitting each degree-\(d\) vertex along \(P\) into \(d\) degree-1 vertices yields only trees that can be cleared by \(r-1\) robots
• Algorithm 1 (which was discussed last time) is 2-approximate and runs in \(O(n)\) time

Recall: a few other facts
• If \(T\) 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
• \(T\) can be cleared by \(r\) robots iff \(T\) contains a path \(P\) such that splitting each degree-\(d\) vertex along \(P\) into \(d\) degree-1 vertices yields only trees that can be cleared by \(r-1\) robots
• Algorithm 1 (which was discussed last time) is 2-approximate and runs in \(O(n)\) time

Creating an optimal algorithm
• Label each vertex \(v\) with a subset \(S(v) \subseteq \{1,2,...,r\}\) where \(r\) is the number of robots
  - Note: \(r = O(\lg n)\)
• The intuition is that when robot number \(\min(S(v))\) visits vertex \(v\):
  - Robot numbers \(\{\min(S(v))+1,...,\max(S(v))\}\) will be located within the subtree rooted at \(v\)
  - Any robots numbered larger than \(\max(S(v))\) must be located at ancestors of \(v\)

Creating an optimal algorithm (cont.)
• Also label each edge \(e\) with the robot number \(L(e) \in \{1,2,...,r\}\) that will clear edge \(e\)
• If robot \(i\) clears only one edge beneath \(v\), then it might be possible to extend robot \(i\)'s path upward along the edge \((v,\text{parent}(v))\)
• If robot \(i\) clears exactly two edges beneath \(v\), then it might be possible to merge these two paths at vertex \(v\)
• If robot \(i\) clears three or more edges beneath \(v\), then none of these paths will be extended or merged, and a higher-numbered robot will clear the edge \((v,\text{parent}(v))\)
Algorithm 2

- Perform a post-order traversal of T
  - Choose an arbitrary root vertex
- If v is a leaf then
  - S(v)← {1}
  - If v ≠ root(T) then L(v,parent(v)) ← 1
- Otherwise v has k ≥ 1 children c₁,...,cₖ
  - Let x ← the largest value that appears in at least two of the S(cᵢ), or 0 if no such value exists
  - Let y ← max(min(S(cᵢ))), that is, the largest value that is the minimum of any S(cᵢ)

Algorithm 2 (cont.)

- If (x < y) then
  - // attempt to extend path of robot y upward
    - S(v) ← ∪ S(cᵢ) - {1,2,...,y-1}
    - If v ≠ root(T) then
      - L(v,parent(v)) ← y

Algorithm 2 (cont.)

- If (x = y and this value appears in exactly two of the S(cᵢ) and is the minimum in both sets), then
  - // merge the two paths labeled y
    - Let S' = ∪ S(cᵢ) - {1,2,...,y-1}
    - If v=root(T) then
      - S(v) ← S'
    - Otherwise
      - Let z ← the smallest positive integer that is not in S'
      - S(v) ← S' - {1,2,...,z-1} ∪ {z}
      - L(v,parent(v)) ← z

Algorithm 2 (cont.)

- Finally, if (x > y) or if (x = y and the conditions in the preceding case do not hold), then
  - // no paths can be extended or merged
    - Let z ← the smallest integer that exceeds x and that is not in ∪ S(cᵢ)
    - S(v) ← ∪ S(cᵢ) - {1,2,...,z-1} ∪ {z}
    - If v ≠ root(T) then
      - L(v,parent(v)) ← z

Example 1: tree T₄

- Algorithm 1 yields 4 robots
- Algorithm 2 yields 4 robots

Example 2: T₄ minus one leaf

- Algorithm 1 yields 4 robots
- Algorithm 2 yields 3 robots
Example 3

- Algorithm 1 yields 5 robots
- Algorithm 2 yields 3 robots

Correctness (sufficiency)

- T can be cleared using \( r = \max(S(\text{root}(T))) \) robots
- First locate the path \( P \) along which robot number \( r \) moves (using the edge labels, \( L \))
- Begin by sending robot \( r \) to an endpoint of \( P \)
- As robot \( r \) visits each vertex \( v \) along \( P \), robots \( \{1, 2, \ldots, r-1\} \) recursively clear each subtree of \( v \)
- Also, when robot \( r \) visits the vertex \( v \) of \( P \) that is nearest to \( \text{root}(T) \), robots \( \{1, 2, \ldots, r-1\} \) recursively clear the portion of \( T \) that lies above \( v \), unless of course \( v = \text{root}(T) \)

Correctness (optimality)

- Algorithm 2 maintains the following invariant at every vertex \( v \):
  - Consider any feasible set of robots that satisfies the same conditions that \( S(v) \) does
    - Robot numbers \( (\min(S(v)) + 1, \ldots, \max(S(v))) \) are located within the subtree rooted at \( v \)
    - Any robots numbered above \( \max(S(v)) \) are located at ancestors of \( v \)
  - Among all possible such sets that can clear the subtree rooted at \( v \)
    - When sorted into descending order, \( S(v) \) is lexicographically the smallest

Analysis

- Algorithm 2 runs in \( O(n \times r) \) time, where recall \( r = O(\lg n) \)
  - This is because the time needed to determine \( S(v) \) at each node \( v \) is proportional to the product of \( r \) and the number of children of \( v \)
  - Assuming that each set \( S(v) \) is maintained as a sorted doubly-linked list, so each union can be performed in \( O(r) \) time

More careful analysis

- Algorithm 2 runs in \( O(n) \) time, independently of \( r \)
  - Each union operation on two sets can be implemented to require at most \( m \) comparison steps, where \( m \) is the lesser of the maxima of the two sets whose union is being performed
  - Such a union is destructive, that is, it might destroy the two sets whose union is being taken
  - An induction shows that for \( 1 \leq m \leq r \), the number of unions that involve two sets that each contain a value \( m \) or greater is at most \( 2n/2^m \)

More careful analysis (cont.)

- Finally, the total time for all the unions is at most proportional to:
  \[
  \sum_{1 \leq m \leq r} [m \times 2n/2^m] \\
  \leq \sum_{1 \leq m \leq r} \sum_{i=1}^{m} 2n/2^m \\
  \leq \sum_{1 \leq m \leq r} \sum_{i=1}^{m} 2n/2^m \\
  \leq \sum_{1 \leq m \leq r} 4n/2^i \\
  \leq 4n \\
  = O(n) 
  \]
Another result (next week)

- The following variation of the pursuit-evasion problem is NP-complete
  - Input:
    - Arbitrary graph $G = (V,E)$
    - Even if $G$ is restricted to be a treewidth-2 graph (series-parallel graph)
    - Edge widths $g(e) \geq 1$, vertex widths $g(v) \geq 1$
    - Number of available robots, $r$
  - Output:
    - Can graph $G$ be cleared using at most $r$ robots?