Finding the Wood by the Trees

Marc van Kreveld

Abstract

We'll give a strategy to locate all trees (points) from a starting tree using a rope, navigation equipment, computation, and some memory.

1 Introduction

It is a well-known fact that it is sometimes difficult to see the wood for the trees. In this note it will be nontrivial to find even the trees which constitute a wood. Suppose an environment with a set of trees is given of which the locations are unknown. The number of trees is also unknown. Imagine that you are walking amidst the trees, but somehow all trees are invisible (for instance, suppose you are blind). However, you are equipped with perfect distance and heading sensors, a memory, exact computation for reals, a pair of legs, the possibility to walk along a straight line to a specified point, and a rope.

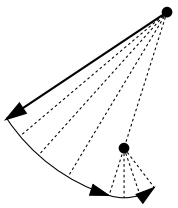

Figure 1: Finding another tree.

More abstractly, the trees are fixed points in the plane and the reader is a moving point. You are supposed to visit all trees (enumerate them), but you cannot see them. This is where the rope comes in. Suppose you, Y, are standing at a tree, and want to find another tree. You'll attach the rope to the tree, walk some distance away while hanging on to the rope, and then describe a circular arc with a tight rope. As soon as you discover that you is following a circle with different radius than the distance to the original tree, you have located and can compute the coordinates of a new tree using the distance and heading sensors, which let you know your position at all times.

Note that if Y doesn't know the location of any tree at all, there is no way of finding one (assuming the rope cannot be attached to the ground). We'll show that if Y is standing at some tree, Y can find them all, provided the rope is sufficiently long. We'll consider enumeration of the trees and convex hulls. One could also consider solving other computational geometry problems like closest pair and Delaunay triangulation. We'll also consider how much resources—the length of the rope and the amount of memory—are needed. Two rope lengths are of special interest:

- The rope should have length at least $\max_{t_1 \in T} \min_{t_2 \in T \setminus \{t_1\}} \mathrm{dist}(t_1, t_2)$, the maximum distance from a tree t_1 to its nearest neighbor t_2. Otherwise, Y cannot find all trees when starting at a given tree. This length is denoted $r_{\min}$.

- The rope can be of infinite length, but more information is needed for Y to know when you can stop walking to discover more trees. You should know an upper bound on either the maximum nearest neighbor distance, or an upper bound on the diameter or perimeter of the forest. Recall that the number of trees is unknown.

Similarly, two memory models are of particular interest. If Y has an unlimited amount of memory, Y can store the coordinates of all trees after finding them all (which is discussed soon), and compute everything by head. If Y has a constant amount of memory, then the situation becomes somewhat harder and finding all trees doesn't solve problems like convex hull anymore.

2 Finding the trees

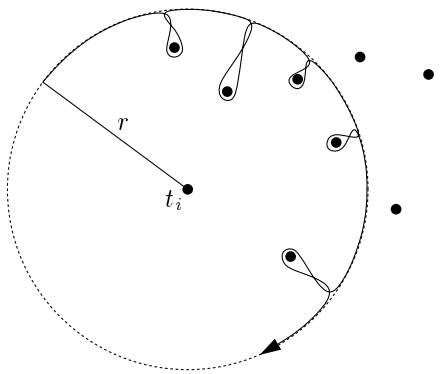

Figure 2: Finding all neighbors within distance r from a tree.

First we show how to find all neighbors within a distance r from a given tree t_i, using only constant memory. Simply describe a circle with radius r around t_i, walking inward around any tree found and going back to the circle where it was left afterwards, see Figure 2. With an infinite rope length and an upper bound on the diameter or perimeter of the forest, it's easy to find all trees from any tree at all. Starting from a tree t_i, just be sure to walk on a circle with radius at least the diameter and start finding all neighbors.

When the rope length is $r_{\min}$ and memory is unlimited, we can always find a new tree from a set of given trees, or decide that none exists, using the following strategy. Start at some tree, store it, and find all of its $r_{\min}$-neighbors. Store these too, and apply the same strategy to these. Only store a tree if it hasn't been stored before. Since every tree has a neighbor within distance $r_{\min}$, the whole forest will be found.

The situation is less simple when the rope length is $r_{\min}$ and memory is constant. We'll give a strategy that will surely find all trees, but if we count each tree at each visit, may visit $O(n^6)$ trees in a forest of n trees.

Define the $r_{\min}$-circle graph as the graph where the nodes are the trees, and two nodes are connected by an arc if their trees are at most distance $r_{\min}$ apart, see Figure 3. This graph is a connected, embedded, not necessarily planar graph. Next we consider a planar version of it, where all intersections of arcs are included as nodes. To avoid confusion, we call the new graph $\mathcal{G}$, and it consists of vertices and edges rather than nodes and arcs. So each edge is part of an arc or a whole arc, and each vertex is a node or lies on (at least) two arcs. Each vertex of $\mathcal{G}$ can be represented by one tree, or by two pairs of trees of which the connecting line segments intersect; the intersection point is the vertex. Similarly, each edge of $\mathcal{G}$ can be represented by three pairs of trees. One pair gives the arc on which the edge

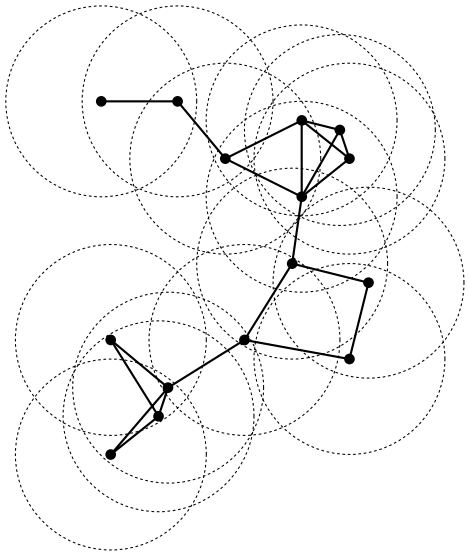

Figure 3: Set of trees and circles with radius $r_{\min}$, and corresponding graph.

lies, and the other two pairs define arcs that intersect the arc of the first pair in vertices. The edge lies between these vertices.

Lemma 1 *Given a vertex defined by two pairs of trees, we can find the next vertex on the arc to any of the trees (using a rope of length $r_{\min}$ and constant memory).*

Proof:

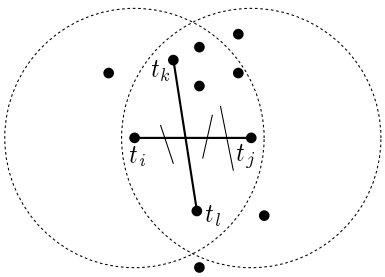

Figure 4: Finding the next edge on an arc (t_i, t_j).

Let the pairs of trees be (t_i, t_j) and (t_k, t_l), and suppose that we want to find the next vertex on the arc (t_i, t_j), closer to t_j. We'll determine all arcs that intersect (t_i, t_j), and select the one defining the next intersection. For each of t_i, t_j, find all of its neighbors t, and for each of its neighbors t', determine if and where (t, t') intersects (t_i, t_j). This correctly locates all arcs intersecting (t_i, t_j), because t or t' (or both) must be within distance $r_{\min}$ from t_i or t_j (or both).

□

The lemma just given is useful because it gives a means to explore the graph $\mathcal{G}$ one step further. Imagining that $\mathcal{G}$ is stored in a doubly-connected edge list, we can use the lemma to find the next edge of a cycle of edges bounding a face of the graph $\mathcal{G}$. Using (t_i, t_j) and (t_j, t_i) to represent different directions of the arc and its edges, we can do the primitives next-halfedge, previous-halfedge, twin-halfedge, origin-of-halfedge, and destination-of-halfedge. Now we apply the algorithm of de Berg et al. [1, 2] for traversal

of a subdivision without using extra storage or mark bits, which applies to any planar, embedded, connected graph.

3 Finding the convex hull

In any of the unlimited memory models it is trivial to find the convex hull once all trees are known and stored. With constant memory and a rope of length the diameter or perimeter of the forest the convex hull determination is also easy. First determine the leftmost tree, then start gift wrapping with the rope around the forest. In case the rope has length the diameter of the forest we may need to go back and untie the rope, then continue at the tree last discovered.

The case with constant memory and a rope of length $r_{\min}$ again is most interesting, because we cannot do gift wrapping directly. The rope is too short. However, we can find the leftmost tree by enumerating all, then mimic gift wrapping by finding the next tree of the convex hull again by enumerating all trees. After $O(n^7)$ tree visits all convex hull edges have been found in order.

4 Toy open problems

Only little imagination is needed to list many open problems. For instance, find the Delaunay triangulation, closest pair, diameter, or minimum spanning tree. As a complexity measure we used the number of trees visited. Try to find strategies that are most efficient. Prove or disprove that a constant memory strategy with rope length $r_{\min}$ exists if and only if the problem in the usual model of computation can be solved in polynomial time.

Acknowledgements. Conversations with Andrew Frank, Jack Snoeyink, Hakan Jonsson, and Christian Icking have led to this note.

References

[1] M. de Berg, M. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal on Geographical Information Science*, 11:359–373, 1997.

[2] Mark de Berg, René van Oostrum, and Mark Overmars. Simple traversal of a subdivision without extra storage. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages C5–C6, 1996.