A VECTOR-BASED APPROACH TO WIRELESS NODE COVERAGE

Defining the problem
Coverage remains an open problem in the field of wireless networking. Much research has been devoted to developing methods of achieving perfect coverage in a distributed, asynchronous environment. Coverage can be used for anywhere from military applications (coordination of troops) to civilian applications, such as deployment of weather sensors and placement of wireless access points.

Overview
Wireless coverage in our case is defined as achieving a final state such that as much area as possible is reachable by the wireless nodes’ sensory capabilities. This must be achieved in a distributed environment, where no one node is superior to another, and no node has knowledge of anything beyond its sensor range – all nodes operate purely on local information. The only knowledge the nodes are capable of is rough direction of their neighboring nodes. Any given node is able to determine, for all of its neighbors, which quadrant the node lies in with respect to itself, and whether it meets a certain distance threshold. To maintain connectivity, the threshold can be set to a fraction (perhaps 80%) of the node’s sensory capability.

The specific problem we address is the following: **If a set of sensory nodes is dropped off in a tightly constricted area, how can the nodes then spread out to achieve maximum coverage while maintaining connectivity?**

Desired Properties
Throughout the protocol we ask that the following properties remain intact:

- **Distributed**: As aforementioned, no node may be superior in any way to any other node.
- **Local information dependency**: All nodes operate purely on local information gathered about their immediate environment.
- **Efficiency**: The algorithm must achieve an efficient solution to the above problem. Nodes should only move when required to. The efficiency of the algorithm will be determined by comparing its output to those of a “perfect” spread and a “random” spread of nodes.
- **Scalability**: The algorithm must be adaptive for any number of nodes.
- **Simplicity**: Abiding by Occam’s Razor, the algorithm must not be unnecessarily complex, nor resort to excessive computation (by solving for connectivity) to achieve its result.
- **Adaptability**: The algorithm should be adaptable so that the settings on it can be changed at will to make it work in various situations.

The Protocol
The original idea for this algorithm was sparked by a lecture by Prof. Brian Scassellati during one of his Intelligent Robotics classes. While discussing methods for allowing robots to move in a dynamic environment, Prof. Scassellati mentioned one scenario known as Motor Schemas. In this model, every aspect of behavior available to a robot could be determined by a set of vectors. If the robot was supposed to move ahead, there was a vector for that. If the robot was to avoid an object, there was a vector for that (in the opposite direction of the first vector). There were vectors to stay in the middle of a path as well as random vectors representing noise. The final action of a robot was determined by a combination of these vectors. Based on local information, different vectors were given different weights, and the final direction in which the robot moved was the aggregate of these vectors.
A similar methodology has been used for this algorithm. Every node gathers local information on its neighbors, determines individual vectors required to move away from them, and then finally moves in the direction suggested by the aggregate vector. This effect is shown on the figure on the next page. Vector b represents a move for node Y away from node X, and vector a represents a move for node Y away from node Z. The aggregate vector c is the final move that node Y would make.

This basic model is applied to all the nodes. To simulate real life situations, a certain amount of noise (roughly 10% of the amount the nodes would move) is added at every step. As with all variables used in this algorithm, this one is easily changed at the top of the code. The noise is in effect mainly to handle the collinear case:

Without noise the nodes would just move apart from each other in a collinear direction. However, due to the randomness added by the noise, the nodes are able to move in unpredictable directions, providing more robustness to the algorithm. Running the algorithm with noise has resulted in a better performance than without noise.

The basic structure, then, of the algorithm is as follows:

- Create an initial distribution of nodes, restricted to a tightly constrained area.
- Create appropriate “repulsion vectors” for the four quadrants.
- Run the algorithm on the array (repeat until exit initiated):
  - Increment repetition counter.
  - Make a copy of current distribution for later convergence testing purposes
  - For every node, determine whether every other node is a neighbor or not (meets distance threshold or not). Recognize when a node is being compared to itself.
  - For every node, determine direction of nodes that are within the distance threshold.
  - For every node, determine aggregate vector by adding together vectors of all the nodes that meet previous two requirements.
  - For every node, there is a chance that noise will be added. Add noise as appropriate.
  - For every node, move the node in the direction of the aggregate vector, scaled by the repetition counter. In essence, the nodes will move by V/t, where V = aggregate vector, and t = time step. The reason for this is under the analysis section.
  - If the distance moved between the last repetition and this one is under a certain threshold (determined by comparing current position array with above copy), exit the repetition.
  - Else, repeat.
- Upon exit from loop, perform analysis of the results:
  - Create two more distributions with the same number of nodes as above:
    - A perfect distribution of nodes within the final area (artificial even spreading of nodes)
    - A random distribution of nodes within the final area
  - Choose a random set of points within the final area and compare the results:
    - For all points, determine the least distance between the point and the closest node in the perfect array. Take the maximum of the distances over all the points.
For all points, determine the least distance between the point and the closest node in the random array. Take the maximum of the distances over all the points.

For all points, determine the least distance between the point and the closest node in the algorithm’s array. Take the maximum of the distances over all the points.

- Compare the result of the maximum distance using the algorithm’s distribution to the maximum distance using the perfect and random distribution. Ideally, the algorithm should be as close as possible to the perfect distribution, and better than the random distribution.

The running of the algorithm and following analysis is performed several times to obtain average values for the final distances.

A sample output of the algorithm with 100 nodes is provided at the end of the paper.

**Scenario Applications**

As we can see based on the results, the algorithm performs better than a completely random distribution of nodes in the given area, but not quite as well as a perfect distribution. One can imagine a situation where a plane that is flying overhead releases a set of sensory nodes within a small, localized area. It is then up to the nodes to distribute themselves (the physical details of which are an engineering and application question) to best cover a large area and provide maximum coverage to monitor the weather.

In a military application, the algorithm may well be used to coordinate troop movements, place outlook posts, or guide reconnaissance unmanned planes to best cover enemy grounds. This algorithm can be used to either run simulations and then place posts or perform real-time calculations for the best spots to place posts. As we have seen, merely placing random posts is not as helpful as running the algorithm first.

**Analysis of protocol**

**Distributed:** The algorithm is completely distributed and does not rely on any oracle or special node to guide it.

**Local information dependency:** No information is ever used that cannot be gained from local sensors.

**Efficiency:** Although the algorithm does not provide a perfect answer (does not find global minima), it does do better than a random algorithm. A node is not moved unless it is required to do so. Thus it is efficient.

**Scalability:** Although the sample run used in this paper was conducted with 100 nodes, the algorithm has been run with as few as 10 nodes, and as many as several thousand. It is limited only by the hardware on which it runs.

**Simplicity:** The algorithm works on a basic idea of “repulsion” from nearby nodes. Thus, it is a simple application of a simple idea.

**Adaptability:** The strongest point on the algorithm is its flexibility. The algorithm can be changed in several ways by altering the values at its beginning. This adds functionality to the algorithm to allow it to be used in various settings.

One of the components in the algorithm is the gradual scaling down of the amount the nodes move by 1/t, where t = current time step. The reason for this was to allow the algorithm to converge. If the nodes are moving less at each time step, they will eventually reach a limit and stop moving. However, this limit should be a limit of their sensor ranges, and not the convergence series being used. Thus, scaling by 1/t^2 would not be appropriate. As can we seen by the equations below, the limit of the indefinite integral of 1/x is infinite, whereas that of 1/x^2 is 1. Simply stated, if we had scaled by 1/t^2, the algorithm would converge too quickly, and not be given a fair chance at achieving optimal coverage.

\[
L = \lim_{x \to 0} \int x^{-1} = \lim_{x \to 0} \ln x = \ln x \bigg|_1^\infty = \ln \infty - \ln 1 = \infty
\]

\[
L = \lim_{x \to 0} \int x^{-2} = \lim_{x \to 0} (-x^{-1}) = -x^{-1} \bigg|_1^\infty = 0 + 1 = 1
\]
Conclusion
By taking an idea from one aspect of computer science and applying it to a completely different area, we have shown that it is possible to achieve optimum solutions by sharing ideas. The algorithm is a good step towards an efficient solution to the problem of coverage, but it has room for improvement. For example, adding several discrete levels of distances and having appropriate vectors would be an improvement, since it would more accurately determine how far the nodes should move. Additionally, the quadrants can be further broken down into octants or even more specific angle ranges for better granularity. However, all of those take away from the beauty of the algorithm: even with very rough knowledge of the environment, it is possible to achieve good coverage. We can even achieve better coverage than would be possible with a random algorithm.

An open question that still remains is whether this algorithm could have been run without any direction information whatsoever. Given only a set of distances between n nodes, would it be possible for the algorithm to compute the same answer. The author believes such an attempt would not be successful, since this algorithm relies very heavily on knowing which vector to take, knowledge which is directly dependent on the direction of a node’s neighbors. Perhaps with enough knowledge of the nodes’ positions, it would be possible to determine the direction, but that would violate the initial condition of not using direction in the algorithm.

Finally, the author wishes to thank Richard Yang for his immense continual guidance and flexibility during this project. The times he spent with the author to solidify the idea were invaluable and the work would not have been completed without his help. The author also wishes to thank David Goldenberg for his assistance on the code and continual input.
Initial distribution of nodes: \{((0.0, 0.1), (0.0, 0.1))\}
All graphs courtesy of Simple Data Grapher:
http://id.mind.net/~zona/simpleDataGrapher/simpleDataGrapher.html
Perfect distribution of nodes: {(-210, 200), (-210, 200)}
Random distribution of nodes: {(-210, 200), (-210, 200)}
Algorithm’s distribution of nodes: {(-210, 200), (-210, 200)}
Analysis:
  Algorithm’s distribution was 1.51464 times worse than perfect distribution
  Algorithm’s distribution was 1.51613 times better than random distribution
  Repetitions required for algorithm convergence: 64