Guiding Sensor-Node Deployment Over 2.5D Terrain

Kerry Veenstra and Katia Obraczka
Computer Engineering Department
University of California, Santa Cruz, CA, USA
Email: {veenstra, katia}@soe.ucsc.edu

***Abstract*—We propose a novel distributed deployment algorithm for sensor networks whose nodes reside upon and are obstructed by 2.5D terrain. Our algorithm optimizes area coverage by computing cumulative visibility over terrain. Through simulation and comparison to centralized algorithms, we demonstrate that our distributed algorithm achieves good results and degrades gracefully with reduced internode communication. To the best of our knowledge, our distributed deployment algorithm is the first use of a distributed simulated annealing algorithm for sensor network deployment. In addition, to our knowledge, this is the first time range-limited cumulative visibility is used to guide sensor deployment over 2.5D terrain. Our results show that a centralized Simulated Annealing algorithm outperforms Pattern Search and Gradient Ascent approaches. Results also show that our version of Distributed Simulated Annealing performs well, degrading gracefully as communication radius is reduced.**

I. Introduction

Deployment algorithms for wireless sensor networks aim to find node locations that optimize network characteristics such as energy usage, network lifetime, and sensor coverage. These algorithms often constrain node locations to a region of a two-dimensional (2D) plane [3] [13] [14] [21] [32] and [37]. Some research has been conducted in deployment within a region of 3D space with or without obstructions [16] and [23]. 2D and 3D scenarios like these can model many real-world network deployments, but they may only approximate the characteristics of networks that are deployed over non-planar 2.5D *terrain* (The term "2.5D" has different meanings in different disciplines. We use the meaning from machining in which the third dimension is a function of the first two [33].)

Most sensors that do not physically touch a target must have a clear line of sight (CLOS) to sense their target's characteristics. In 2.5D deployments, the CLOS requirement is a constraint. For example, sensors that detect visible and near-visible light, such as cameras and motion detectors, cannot see through or around terrain. Other sensors that require CLOS include those that detect ultrasonic acoustic waves or microwave electromagnetic waves.

These scenarios can be modeled using opaque 2.5D surfaces, with corresponding deployment algorithms considering obstructions that are caused by the terrain itself [4] [5]. On 2.5D terrain the "obstructions" to CLOS are hills rather than the assumed vertical walls of 2D scenarios. Consequently apparent boundaries of obstructions in 2.5D seem to vary with the positions sensors and targets.

In this paper we propose a novel distributed deployment algorithm for sensor networks whose nodes reside upon and are obstructed by 2.5D terrain. Our algorithm optimizes area coverage by maximizing cumulative visibility over terrain. As will become clear in section V-C, our algorithm uses a distributed simulated annealing approach that is fundamentally different from other distributed simulated annealing algorithms. Through simulation and comparison to centralized algorithms, we demonstrate that our distributed algorithm achieves good results and degrades gracefully with reduced internode communication.

We believe that this is the first use of Distributed Simulated Annealing for sensor network deployment and the first use of range-limited cumulative visibility to guide sensor coverage over 2.5D terrain.

We start with a description of our coverage model in Section II. Then in Section III we define the 2.5D optimization problem that we aim to solve, and we show how the problem's characteristics affect our choice of optimization algorithms. In Section IV we briefly review some algorithms for non-linear optimization problems. Next we present and evaluate both centralized and distributed Simulated Annealing algorithms in Section V. Finally we present our results in Section VI and conclusions and future work in Section VII.

II. Coverage

Coverage is a key concept in sensor networks. Its definition depends on the type of sensor network and its application. Quantitatively, coverage measures the quality of service that the sensor network provides to its driving application. In this work, we define an objective measure of coverage $f(\mathrm{s})$ as the total number of targets that are visible to at least one member of the set of network sensors.

In the 2D plane, coverage often is defined using the *Disk Model*, which determines visibility between a sensor and a target using just the distance between them [30, ch. 2]. Using the Disk Model, we can evaluate the value of a sensor's position as the number of targets that the sensor's disk covers (where the radius of the disk is the sensor's range). We can extend this idea to measure the value of an entire sensor network in the 2D plane as the number of targets that reside within the union of all of the sensors' disks.

For this work, sensors and targets reside not in 2D but in 3D. However the locations of the sensors and the targets are confined to a 2.5D surface of which the third dimension is a function of the first two [33]. That is, given a height function $h(x, y)$, we define a location in 2.5D as $s = \big(x, y, h(x, y)\big)$.

With sensors and targets confined to the surface of 2.5D terrain, sensor-target visibility can be blocked by features of the terrain itself. For coverage, we use *range-limited visibility*, which requires not only that a target be in a sensor's range, but also that the target has a clear line of sight to the sensor

(CLOS). Furthermore, *range-limited cumulative visibility* is the number of targets that a sensor can see over 2.5D terrain. Using this definition of coverage, our measure of a sensor network's deployment $f(\mathbf{s})$ is its *range-limited cumulative visibility*, which is the number of targets that have CLOS to at least one in-range sensor.

visibility	The ability of a sensor to see a target without obstruction. Requires clear line of sight (CLOS).
range-limited visibility	Visibility that is restricted by a maximum range. The range limit can be due to, for example, attenuation (a microphone hearing sound) or resolution limits (a camera viewing a distant object).
cumulative visibility	The number of targets that a sensor or a sensor network can see (point coverage) or the visible area of the target region (area coverage).
range-limited cumulative visibility	Cumulative Visibility that is restricted by a maximum range.

III. Problem Formulation

In this section we define the 2.5D problem that we aim to solve. But first we note that when we are presented with a sufficiently simple coverage problem, we have available many avenues for finding a good sensor placement that optimizes the Objective Function $f(\mathbf{s})$. For instance, when the number of potential sensor locations L is very small, we can exhaustively compute the Objective Function for all useful combinations of sensor locations and then choose the best. (The number of combinations evaluated will be no more than 2^L.) Alternatively, given a sufficiently small point coverage problem, we can treat it as an equivalent problem in set covering, which can be attacked straightforwardly using a greedy algorithm [8]. Another approach for a point coverage problem is to create an equivalent set of linear inequalities which can be solved effectively using integer linear programming [30, p. 56].

In all of these cases, the number of potential sensor locations L and the number of targets n both are small enough for these algorithms to be tractable. But in this paper we consider area coverage, and so L and n literally are the number of distinct placement locations across terrain. Consequently our problem requires a different approach.

In this paper we compare centralized deployment algorithms and a distributed deployment algorithm. *These algorithms attempt to find locations for a given set of nodes that maximize area coverage over 2.5D terrain.* The centralized algorithms compute locations for all of a set of nodes assuming that all location and coverage information is available. The distributed algorithm considers communication constraints: each node attempts to optimize the overall Objective Function while it has only limited position information for other network nodes.

IV. Non-Linear Optimization

Presented with a large optimization problem, often one cannot find an algorithm that guarantees an optimal solution in reasonable time. Instead one uses an iterative heuristic to find a good solution. Examples of heuristics are gradient ascent, pattern search, and the Nelder-Mead simplex algorithm.

Gradient-based optimization algorithms follow the slope of the objective function to select a series of candidate solutions [15, Ch. 5] [10, p. 111]. These algorithms assume smooth gradients. However, considering outdoor deployments, practical 2.5D terrain elevation data is noisy [11, ¶100], making computed visibility-based objective functions noisy as well. Consequently we prefer optimization algorithms that do not rely on objective-function gradients.

There is an alternative optimization scheme that requires no derivative computations and compares just the values of the objective function: *direct search* [36]. Examples of direct search are Fermi and Metropolis's *coordinate search* [9] and Nedler and Mead's *simplex search* [19, §9.5], [22].

While less susceptible to noise than gradient-based methods, direct-search methods still are greedy, and so they can get caught in a local optimum. Even a seemingly simple terrain function $h(\cdot)$ can lead to in an objective function $f(\mathbf{s})$ that has multiple local optima. This shortcoming of direct search algorithms traditionally leads one to restart the algorithms from numerous random starting points in an attempt to improve the chance of discovering the global optimum.

V. Proposed Solution

Rather than repeatedly restarting an optimization algorithm from the beginning, an alternative strategy is to develop an optimization algorithm that backtracks partway from a local optimum. The heuristic algorithm Simulated Annealing has been developed to use this strategy.

We have defined our optimization problem as determining the locations of a set of nodes, $\mathbf{s}$, that maximize cumulative visibility $f(\mathbf{s})$. In this section we present a distributed deployment algorithm that uses our version of distributed simulated annealing. But we start by justifying our choice of terrain representation since it affects the algorithm (Section V-A). Afterward, in Section V-B we present a centralized placement algorithm using classic Simulated Annealing, and then in Section V-C we present an algorithm that uses our distributed version of Simulated Annealing.

A. Terrain Representation

Practical terrain data is discontinuous, as it is a set of distinct elevation samples. Our evaluations for this paper use a dataset from NASA's 2005 Shuttle Radar Topography Mission (SRTM) [26], [27]. One result of that mission is a dataset of global elevations with one-second-of-arc resolution (approximately 30 meters or 100 feet in the latitude direction). This sort of data is called a Digital Elevation Model or DEM [7]. Such a high-resolution DEM offers an opportunity to evaluate the coverage of a WSN deployment.

Since DEM data does not define a continuous surface, visibility computations must assume a spatial representation for the data in order to compute obstructions. For example, if one were to represent terrain using a Regular Square Grid (RSG) [24], the world would be a collection of tightly packed prisms with each prism's height determined by a corresponding point in the DEM. Then the author of a visibility algorithm would need to decide what "seeing a datapoint" means: is it seeing the top face of the prism? Is it seeing part of an edge? An RSG representation is far removed from actual topography, and our reading of the literature suggests that it is not the most popular representation in visibility computations.

A more popular representation is a Triangulated Irregular Network (TIN) [25], which most GIS (Graphical Information Systems) visibility algorithms use. The advantage of using a TIN is that a group of TIN triangles can be merged into a larger triangle for efficient *storage*. De Floriani evaluates several TIN-based visibility algorithms [12].

Although TIN is popular for visibility computations, sensor nodes require an efficient *algorithm*, and so we have selected the terrain representation that is used by a dynamic-programming-based viewshed algorithm by Wang, Robinson, and White [31]. This algorithm represents terrain as a grid of vertical height poles, fitting sight planes through the viewpoint and selected pairs of adjacent poles and then computing visibility by checking whether a more distant pole pierces the sight plane.

B. Centralized Algorithm

Simulated Annealing is a general stochastic optimization method whose operation is inspired by the annealing of a metal. The method is notable in that it has been used successfully in finding solutions to various discrete optimization problems, including deployment of sensor networks [18], [20] and even the physical design of VLSI circuits [35].

A Simulated Annealing algorithm generates a sequence of incremental candidate solutions, determining at each point in the sequence whether the candidate solution is accepted or rejected. The criteria for acceptance depends on the change in the system's global utility, comparing a candidate solution to the solution that was most recently accepted. (Often Simulated Annealing acceptance criteria depends on the change in *cost* rather than the change in *utility*. One negates the change in cost to obtain the change in utility.) Given a prior accepted solution s and a candidate solution s', the probability of accepting s' is computed from the potential change in utility

$$\Delta = f(\mathrm{s}') - f(\mathrm{s}) \tag{1}$$

yielding

$$P(\text{accept s}') = \begin{cases} 1 & \text{if } \Delta \geq 0 \\ e^{\Delta/T} & \text{if } \Delta < 0 \end{cases} \tag{2}$$

In (2) T is a control parameter that is analogous to a system's temperature during annealing. (Stated more accurately, T combines temperature and Boltzman's constant, but the Simulated Annealing algorithm functions without their separation and refers to their combination as "temperature" [35, p. 5].)

Our centralized Simulated Annealing algorithm mostly follows the classic algorithm presented in [29, p. 54] except for the mechanism that determines the annealing schedule. As in the classic algorithm, the outer **repeat/until** loop in lines 2–17 controls temperature. During each iteration of this loop, the contained **for** loop of lines 4–13 (called the Metropolis loop) generates and evaluates a succession of M candidate solutions at a fixed temperature T. The value of T is recomputed only after every M solution evaluations.

An important aspect of any Simulated Annealing algorithm is the mechanism whereby one chooses a sequence of values for T. In line 11 of the listing, where the exponential of the probabilistic acceptance criteria (2) is implemented, note that as $T \to \infty$ all attempts to decrease utility are accepted, but as $T \to 0$ all such attempts are rejected. Early applications of Simulated Annealing just entered the outer loop with a large value of T that forced acceptance of all attempts at utility reduction. The loop then periodically multiplied T by a constant between zero and one, reducing T monotonically [29, p. 54]. However, a disadvantage of this simple method is that the rate of temperature reduction is fixed without any feedback. So if one reduces T too rapidly the system "quenches" early before finding an optimum, degenerating early to a greedy algorithm. But if one changes T too slowly, execution time of the algorithm extends unnecessarily.

To avoid both quenching and long run times, adaptive cooling schedules have been developed. For our Simulated Annealing algorithm, we adopt an efficient general cooling schedule [29, p. 72]. This schedule attempts to control T such that consequential increases in the objective function are limited to the value of the objective function's standard deviation. (See the reference for the inspiration behind this choice.)

Another important aspect of a Simulated Annealing algorithm is the generation of candidate solutions. Candidate solutions should be incremental changes to the current solution rather than large jumps. In addition, the algorithm that generates candidate solutions should allow the possibility of returning to any earlier solution. For our algorithm, a candidate solution is generated by selecting a random node and computing a random new location for it. The new location is selected from a uniform circular distribution that is centered on the current location. The radius of the distribution is set to an arbitrary value over all test runs.

```
1:  Initialize state s, cycle count $M$, and temp. $T$.
2:  repeat
3:    Select a random sensor $k$.
4:    for $i = 1$ to $M$ do
5:      Select a random potential move for sensor $k$.
6:      Create a potential next state $\mathrm{s}'$ from this move.
7:      Compute $\Delta = f(\mathrm{s}') - f(\mathrm{s})$
8:      if $\Delta > 0$ then
9:        $\mathrm{s} \leftarrow \mathrm{s}'$
10:     else
11:       $\mathrm{s} \leftarrow \mathrm{s}'$ with probability $e^{\Delta/T}$
12:     end if
13:   end for
14:   time $\leftarrow$ time $+ M$
15:   $M \leftarrow \beta M$ $\{0 < \beta < 1\}$
16:   Reduce T. {see text}
17: until time $\geq$ time_max or $T = 0$
18: Return the best s seen.
```

C. Distributed Algorithm

A single computer running the centralized deployment algorithm that we just described must know all of the node's positions. We relax that requirement in this section when we distribute the deployment algorithm among the nodes and limit each node's knowledge to the positions of just its neighbors. The distributed Simulated Annealing algorithm that we discuss here is not the same as a parallel implementation of a Simulated Annealing algorithm: instead each node runs its own annealing loop with different, incomplete knowledge of the other nodes.

The best way to distribute a Simulated Annealing algorithm depends on the method of generating candidate solutions and the nature of the system's objective function. Arshad and Silaghi present a straightforward distributed Simulated Annealing algorithm, DSAN, in which all nodes attempt to solve an identical scalar optimization problem [1]:

```
1:  Initialize value $v$, cycle count $M$, and temp. $T$.
2:  repeat
3:    Randomly choose a value for $v$.
4:    for $i = 1$ to $M$ do
5:      if $v$ changed then
6:        Send the new value of $v$ to neighbors.
7:      end if
8:      Collect neighbors' new values of $v$, if any.
9:      Choose a random neighbor's value $v$.
10:     Compute $\Delta = f(v') - f(v)$
11:     if $\Delta > 0$ then
12:       $v \leftarrow v'$
13:     else
14:       $v \leftarrow v'$ with probability $e^{\Delta/T}$
15:     end if
16:   end for
17:   Reduce T.
18: until time $\geq$ time_max or $T = 0$
19: Return $v$
```

In DSAN, during each iteration of the Metropolis loop (the **for** loop), each node shares its current best solution to the problem with its neighbors (line 6) and then adopts a random neighbor's solution based on the Simulated Annealing probabilistic acceptance criteria given by (2) (see lines 11 through 15).

Our distributed Simulated Annealing algorithm, which we present below, differs from DSAN in two fundamental ways. First, while a node running DSAN communicates its solution to its neighbors during each iteration of the Metropolis loop, our algorithm saves energy by performing inter-node communication only at the completion of the loop (that is, after every M iterations, when temperature parameter T is recomputed). This change reduces communication energy by a factor of M.

Second, and more significantly, nodes running DSAN work together to converge on a single scalar value, while our 2.5D deployment problem needs a vector of different locations, a different location for each node of the network. In other words, while nodes using DSAN probabilistically adopt a neighbor's *scalar* solution, the current solution (or state) of a node in our system corresponds to a *vector* of node locations, with the node's own location being the only one known with certainty. A node running our distributed algorithm will create a new solution through two steps: (1) moving itself probabilistically using the Simulated Annealing Metropolis loop (updating its own location in its own solution vector) and then (2) accepting all of its neighbors' own self-reported locations with 100% certainty.

Each node running our algorithm needs to estimate its contribution to the objective function $f(\mathbf{s})$. For a node to answer this question, our distributed algorithm uses a local objective function called the "wonderful life utility," or WLU.

The WLU for a node compares the global utility of the system with the global utility of an alternate world in which the node doesn't exist. The WLU is the node's individual contribution to global utility. (The local utility function's name is inspired by a scene in the Frank Capra movie *It's a Wonderful Life* [6] during which James Stewart's character learns what his home town would have been like if he hadn't been born.) For example, in Fig. 1 the value of WLU($\mathbf{s}$) is represented by a shaded region. The area of this region is the contribution that node s makes to overall coverage. As one can see from the figure, as long as the other nodes are momentarily immobile, changes in a node's WLU equal changes in overall coverage.

Changes in a node's WLU are the same as changes in the global objective function $f(\mathbf{s})$ as long as no two nodes change state simultaneously [2], [34]. That is

$$\mathrm{WLU}(\mathbf{s}') - \mathrm{WLU}(\mathbf{s}) = f(\mathbf{s}') - f(\mathbf{s}) = \Delta \tag{3}$$

One can see from (3) that an algorithm that bases its decisions on *changes* in an objective function can guide its operation with the local utility function WLU($\mathbf{s}$) instead of the global objective function $f(\mathbf{s})$. While this choice does not affect the operation of a centralized algorithm, with the addition of a few constraints, one can use WLU($\mathbf{s}$) to construct a distributed algorithm whose average results closely match those of the centralized algorithm.

1) Communication: A node can compute its WLU accurately as long as it has three values: its own location, the locations of its neighbors, and an understanding of visibility over terrain. Depending on the circumstances, the node does not need to know the locations of *all* of its neighbors.

Considering communication between nodes, we must account for the fact that radio signals have vastly different propagation properties than visible light. On one hand, under appropriate conditions, radio waves (such as VHF) can travel around and through obstacles that are opaque to visible light [28, §16.2.2]. On the other hand, even with CLOS, radio propagation can be affected by fading due to multi-path effects [17, §7.3]. Clearly radio propagation is more complicated than can be represented by the disk model. However we use the disk model for radio communication for simplicity, and we require CLOS for sensing. (As part of future work, we plan to use a more realistic radio propagation model.)

Consider first when all nodes are immobile. To properly compute their WLUs, two neighboring nodes must be aware

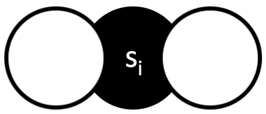

Fig. 1. The Wonderful Life Utility (WLU) of a node is its individual contribution to overall utility. In this example, the overall utility of three nodes working together is the total area of three overlapping disk models. The WLU of node s_i is represented by the central black region, which shows what would be uncovered were node s_i to move to a completely different location.

when they cover the same region. If the nodes are separated by twice the sensing radius R_s, then (assuming CLOS) they both can *just* cover the point midway between them. To be aware of each other's positions, the nodes' communication radius R_c must be at least twice R_s:

$$R_c \geq 2R_s$$

Now let us consider a mobile node (assuming the same sensing and radio-propagation models). If a node is mobile and it is evaluating a potential move, then for an accurate WLU computation, the required communication radius must be increased by the maximum possible distance that the node could move away from its current position before communicating with its neighbors again. We call this distance the nodes' *exploration radius* R_e.

$$R_c \geq 2R_s + R_e$$

The prior inequality assumes immobile neighbors. Since two nodes that are out of communication range may move simultaneously toward each other, the required communication distance must be increased by another R_e. This gives us the constraint relation in its most general form:

$$R_c \geq 2R_s + 2R_e \tag{4}$$

We can rearrange (4) to create a constraint on R_e:

$$R_e \leq \tfrac{1}{2}R_c - R_s \tag{5}$$

This constraint on the exploration radius is sufficient to prevent nodes that are not communicating (that is, which are separated by a distance greater than R_c) from inadvertently covering the same area of interest after moving.

2) Our Algorithm: Our version of distributed Simulated Annealing, as it runs on a single sensor node, is below. It differs from the centralized algorithm in two ways. First, instead of selecting a random node k to move (centralized algorithm line 3), each node always selects itself. Second, while both algorithms follow the same Metropolis procedure (lines 4–13 of both algorithms), the centralized algorithm uses always up-to-date neighbor positions during all iterations of the Metropolis loop, but the distributed algorithm must use neighbor positions that are communicated outside of the loop in lines 14–15.

```
1:  Share this sensor's location with neighbors.
2:  Initialize state s, cycle count $M$, and temp. $T$.
3:  repeat
4:     for $i = 1$ to $M$ do
5:        Select a random potential move for this sensor.
6:        Create a potential next state $s'$ from this move.
7:        $\Delta \leftarrow \text{WLU}(s') - \text{WLU}(s)$.
8:        if $\Delta > 0$ then
9:           $s \leftarrow s'$
10:       else
11:          $s \leftarrow s'$ with probability $e^{\Delta/T}$
12:       end if
13:    end for
14:    Share this sensor's location with neighbors.
15:    Incorporate other sensors' locations into s.
16:    time ← time + $M$
17:    $M \leftarrow \beta M$ $\{0 < \beta < 1\}$
18:    Reduce T. {see text}
19: until time ≥ time_max or $T = 0$
20: Return the best s seen
```

VI. Results

A. Experimental Methodology

To evaluate our use of range-limited cumulative visibility as an objective function, we created a custom simulation testbed in C++. The testbed reads terrain data from DEM files, executes one of several deployment algorithms for a set of nodes over a region of terrain, and then reports the cumulative visibility of the computed solution $\mathbf{s}$. In addition, the testbed lets us write a sequence of bitmap images that we use to animate an algorithm's operation.

The DEM terrain data comes from two sources. The first is SRTM radar topography data for two areas near our university. We use topography of a forested valley (identified as "Real") and topography of flatter urban region (identified as "RealB"). We use a subregion of each area with 240×180 elevation values. (To simplify our algorithms' distance computations, we treat these SRTM samples as square although actually they represent elevations of non-square regions that subtend one second of arc in latitude and longitude.)

The second source of DEM data are four mathematically defined synthetic terrains. These represent a flat surface, a set of irregular steps, and two terrains that are defined by sinusoids:

$$\begin{aligned} h_{\text{wavy}_1}(x,y) &= 1000\left[\sin^2(\frac{\pi}{360}x) + \sin^2(\frac{\pi}{240}y)\right] \\ h_{\text{wavy}_2}(x,y) &= 1000\left[\sin^2(\frac{\pi}{180}x) + \sin^2(\frac{\pi}{120}y)\right] \end{aligned}$$

where

$$\begin{aligned} x &= \{0,\ldots,239\} \\ y &= \{0,\ldots,179\} \end{aligned}$$

In all cases, the objective function $f(\mathbf{s})$ measures the number of elevation points that are visible to one or more nodes. Given a subregion of 240×180 elevation points, the maximum value of the objective function is the total number of elevation points, or 43,200.

B. Centralized Algorithms

We ran our centralized algorithm on our testbed with each of the six terrains. We compared its results to those of greedy algorithms Gradient Ascent [15, ch. 5] and Pattern Search [9]. Each algorithm/terrain combination was run ten times. We initialized each run with a different set of ten random node locations.

The results (Fig. 2) show that Simulated Annealing finds better deployments over all tested terrains than either Gradient Ascent and Pattern Search. Since Simulated Annealing can avoid getting stuck in local optima, we expect that is why it outperforms the greedy-style algorithms. That said, we are surprised at the performance of Pattern Search on the synthetic terrains.

C. Distributed Algorithm

We ran our distributed algorithm over the same terrains and under the same conditions that we used for testing the centralized algorithms. The performance of our version of Distributed Simulated Annealing (Fig. 3) improves with increased communication radius R_c, which is expected. Comparing the results of our Distributed Simulated Annealing algorithm with the centralized Simulated Annealing algorithm, we see that communication is important in achieving good results. Notice that as the communication radius of the distributed algorithm increases to values that are nearly equal to the dimension of the test range, the results become comparable to those of the centralized algorithm.

We also ran the algorithm for the case where 20 sensor nodes are used. We observe that the algorithm exhibits similar behavior in terms of how its performance varies with R_c.

In our test runs, while varying the maximum communication radius R_c we set the maximum exploration radius R_e correspondingly by treating (5) as an equality. To determine whether our results were affected more by constraining communication or by constraining exploration, we also ran a set of tests that increase R_e beyond the limit specified in inequality (5). The results (Fig. 4) show that limiting the exploration radius has little effect on the final results obtained by the distributed algorithm.

While initially puzzled with this result, we now believe that we understand it. Our motivation for limiting R_e is to ensure that changes in WLU always equal changes in global utility (3). While this restriction ensures that the Simulated Annealing algorithm never makes a decision based on an incorrectly computed value of the objective function, the algorithm already tolerates missteps. Following each Metropolis procedure, after neighboring nodes communicate their new locations, changes in WLU once again will match changes in the objective function. Then at the beginning the next Metropolis loop, any nodes that have inadvertently moved to cover the same topography will tend to part since the updated WLU will favor separation. Indeed, since Simulated Annealing algorithms benefit from—and actually rely on—randomness, we should not be surprised to see that restricting movement is not beneficial (such restrictions even could extend convergence times).

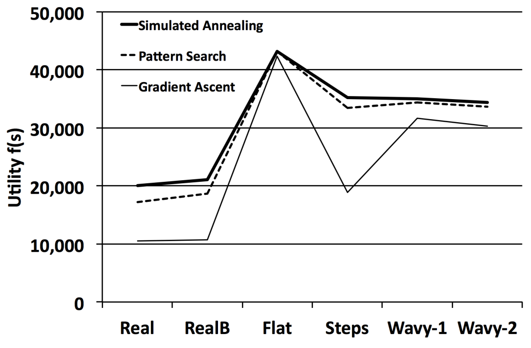

Fig. 2. Results for centralized algorithms. Each data point is computed as an average over ten random starting positions for ten nodes. For the Real and RealB terrains, the gradient ascent results show a standard deviation that is 13%–15% of the mean, pattern search results show a standard deviation that is 3%–4% of the mean, and simulated annealing results show a standard deviation that is 1%–3% of the mean. Note that all algorithms find the optimal solution for flat terrain. The optimal results for other terrains are unknown.

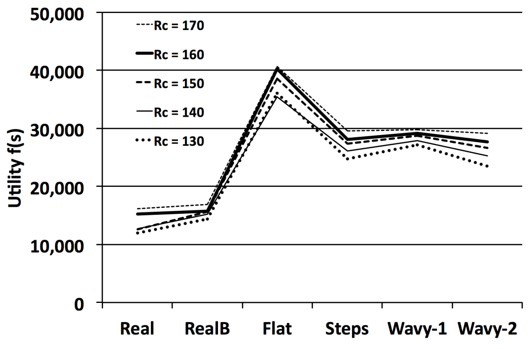

Fig. 3. Results for Distributed Simulated Annealing. Each data point is computed as an average over ten random starting positions for ten nodes. For all terrains, results show a standard deviation of about 10% of the mean with some variation as low as 5% and as high as 21%. We expected to see a larger communications radius R_c help the algorithm find a placement with greater utility.

VII. Conclusions and Future Work

We have proposed a novel distributed deployment algorithm for sensor networks whose nodes reside upon 2.5D terrain. To the best of our knowledge, our distributed algorithm is the first to use a distributed simulated annealing algorithm for sensor network deployment in 2.5D terrain. In addition, to our knowledge, this is the first time cumulative visibility is used to guide sensor deployment over 2.5D terrain. In order to limit communication requirements, our algorithm uses the Wonderful Life Utility as a local utility function. The WLU measures a node's individual contribution to global utility, making it ideal for a distributed algorithm.

Our results show that a centralized simulated annealing algorithm outperforms Pattern Search and Gradient Ascent approaches. Results also show that our version of distributed simulated annealing performs well, degrading gracefully as communication radius is reduced.

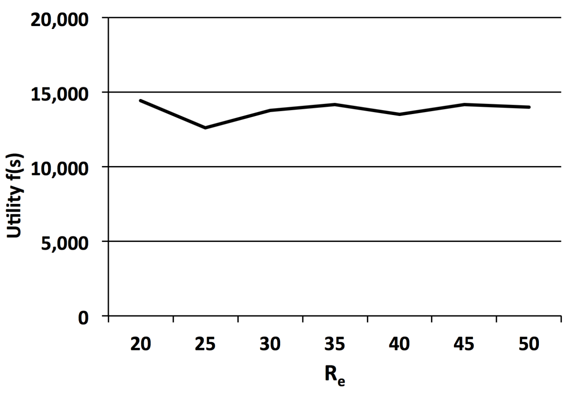

Fig. 4. There appears to be no trend in the influence of the exploration radius R_e on the final objective function found by the distributed deployment algorithm. Upon reflection, we believe that this result is expected: simulated annealing algorithms automatically correct for missteps. In this case, any inadvertent coverage overlap caused by a large exploration radius is tends to be corrected during the following executions of the Metropolis procedure.

Our inclusion of an exploration radius R_e was intended to help the algorithm avoid unintended node-to-node interference when communication is limited, but our tests show that this feature apparently is unnecessary.

In the future we want to incorporate a more realistic communications model than the Boolean disk, and we would like to include connectivity in the utility model. In addition, we want to see the effect of removing the movement restriction R_e completely. At present we are porting our algorithm into a sensor network simulator which will help us explore more realistic communication models. In this platform, we also will compare our algorithm to other distributed heuristic algorithms such as gradient descent.

Acknowledgment

The authors would like to thank Jorge Cortés of UCSD for introducing us to Potential Games and WLU.

References

[1] M. Arshad and M. C. Silaghi. Distributed simulated annealing. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, pages 35–47. IOS Press, 2004.

[2] G. Arslan, J. R. Marden, and J. S. Shamma. Autonomous vehicle-target assignment: A game-theoretical formulation. *Transactions of the ASME*, 129:584–596, September 2007.

[3] M. Batalin and G. Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675, 2007.

[4] P. Bose, D. G. Kirkpatrick, and Z. Li. Efficient algorithms for guarding or illuminating the surface of a polyhedral terrain. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 217–222. Carleton University Press, 1996.

[5] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding polyhedral terrains. *Computational Geometry: Theory and Applications*, 7:173–185, 1997.

[6] F. Capra. It's a Wonderful Life. Film, 1946. Liberty Films.

[7] K. Chang. *Introduction to geographic information systems*. McGraw-Hill, 6th edition, 2012.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, 3rd ed. edition, 2009.

[9] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, February 1991.

[10] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. A. K. Peters, Ltd., Wellessley, Massachusetts, 2006.

[11] T. G. Farr, P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, D. Seal, S. Shaffer, J. Shimada, J. Umland, M. Werner, M. Oskin, D. Burbank, and D. Alsdorf. The shuttle radar topography mission. *Reviews of Geophysics*, 45(2), 2007.

[12] L. D. Floriani and P. Magillo. Algorithms for visibility computation on digital terrain models. In *SAC*, pages 380–387, 1993.

[13] N. Heo and P. Varshney. An intelligent deployment and clustering algorithm for a distributed mobile sensor network. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 2003.

[14] A. Howard, M. J. Matarić, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. In *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, volume 13, pages 113–126, 2002.

[15] S. L. S. Jacoby, J. S. Kowalik, and J. T. Pizzo. *Iterative Methods for Nonlinear Optimization Problems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[16] C.-W. Kang and J.-H. Chen. Multi-objective evolutionary optimization of 3d differentiated sensor network deployment. In *GECCO'09*, 2009.

[17] C. A. Levis, J. T. Johnson, and F. L. Teixeira. *Radiowave Propagation: Physics and Applications*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2010.

[18] F. Y. Lin and P. Chiu. A simulated annealing algorithm for energy-efficient sensor network design. In *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 183–189, April 2005.

[19] T. V. Mikosch, S. I. Resnick, and S. M. Robinson, editors. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer Science+Business Media, LLC, 2nd edition, 2006.

[20] G. Molina and E. Alba. Wireless sensor network deployment using a memetic simulated annealing. In *International Symposium on Applications and the Internet*, pages 237–240, August 2008.

[21] H. Mousavi, A. Nayyeri, N. Yazdani, and C. Lucas. Energy conserving movement-assisted deployment of ad hoc sensor networks. *IEEE Communications Letters*, 10:269–271, 2006.

[22] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7:308–313, 1965.

[23] C. D. Ortiz, J. M. Puig, C. E. Palau, and M. Esteve. 3d wireless sensor network modeling and simulation. In *SensorComm 2007: International Conference on Sensor Technologies and Applications*, pages 307–312. IARIA, October 2007.

[24] T. K. Peucker and D. H. Douglas. Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing*, 4:375–387, 1975.

[25] T. K. Peucker, R. J. Fowler, J. J. Little, and D. M. Mark. The triangulated irregular network. In *Proceedings, American Society of Photogrammetry: Digital Terrain Models (DTM) Symposium*, pages 516–540. American Society for Photogrammetry & Remote Sensing, May 1978.

[26] E. Rodriguez, C. S. Morris, and J. E. Belz. A global assessment of the SRTM performance. *Photogrammetric Engineering & Remote Sensing*, 72:249–260, 2006.

[27] E. Rodriguez, C. S. Morris, J. E. Belz, E. C. Chapin, J. M. Martin, W. Daffer, and S. Hensley. An assessment of the SRTM topographic products. Technical Report JPL D-31639, Jet Propulsion Laboratory, Pasadena, California, 2005.

[28] R. Rudd. Short-range and Indoor propagation. In L. Barclay, editor, *Propagation of Radiowaves, 3rd ed.*, pages 308–310. The Institution of Engineering and Technology, 2013.

[29] S. M. Sait and H. Youssef. *Iterative Computer Algorithms with Applications in Engineering*. IEEE Computer Society, Los Alamitos, CA, 1999.

[30] B. Wang. *Coverage Control in Sensor Networks*. Springer, 2010.

[31] J. Wang, G. J. Robinson, and K. White. Generating viewsheds without using sightlines. *Photogrammetric Engineering & Remote Sensing*, 66(1):87–90, January 2000.

[32] Y.-C. Wang, C.-C. Hu, and Y.-C. Tseng. Efficient placement and dispatch of sensors in a wireless sensor network. *IEEE Transactions on Mobile Computing*, 7:262–274, 2007.

[33] Wikipedia. "2.5D (machining)". http://en.wikipedia.org/wiki/2.5D_(machining).

[34] D. H. Wolpert and K. Tumer. An introduction to collective intelligence. In J. M. Bradshaw, editor, *Handbook of Agent Technology*. AAAI Press/MIT Press, 2000.

[35] D. Wong, H. Leong, and H. Liu. *Simulated Annealing for VLSI Design*. The Springer International Series in Engineering and Computer Science. Springer, 1988.

[36] M. H. Wright. Direct search methods: once scorned, now respectable. In *Numerical Analysis 1995*, pages 191–208. Addison Wesley Longman Limited, 1995.

[37] C.-H. Wu, K.-C. Lee, and Y.-C. Chung. A delaunay triangulation based method for wireless sensor network deployment. *Computer Communications*, 30(14–15):2744–2752, 2007.