Grid Partition: an Efficient Greedy Approach for Outdoor Camera IoT Deployments in 2.5D Terrain

Kerry Veenstra
Baskin School of Engineering
University of California
1156 High Street
Santa Cruz, CA, 95064
United States
veenstra@ucsc.edu

Katia Obraczka
Baskin School of Engineering
University of California
1156 High Street
Santa Cruz, CA, 95064
United States
katia@soe.ucsc.edu

***Abstract*—In this paper, we introduce Distributed Grid Partition, a distributed greedy deployment algorithm for outdoor IoT camera networks. The proposed algorithm optimizes visual network coverage over 2.5D terrain. The main idea behind Distributed Grid Partition is that each deployment node tries to find the best vantage point in its neighborhood that will maximize the network's overall visual coverage. It does so by using information from its immediate neighbors. In order to achieve a favorable cost-performance trade-off, Distributed Grid Partition uses height as a proxy for visual coverage, or fitness, avoiding expensive fitness computations. In addition, each node's contribution to network fitness is determined without knowledge of the overall network using the concept of "Wonderful Life Utility". Our experimental results show that Distributed Grid Partition results in deployments with superior coverage-cost performance when compared to other distributed optimization algorithms as well as a centralized greedy set cover heuristic.**

I. Introduction

Outdoor camera IoT networks, i.e., wireless camera networks for outdoor deployment, have a wide range of applications, including surveillance of outdoor areas, environmental monitoring, and event tracking. *Deployment* of IoT camera nodes (that is, choosing their positions) on the surface of a 2D plane has been well researched, using *line of sight* (LOS) between cameras and targets to determine camera/target *visibility*. Less is known about deployment in 3D regions, where nodes are constrained to the surface of terrain, and landforms block the cameras' LOS views. We refer to such deployments as *2.5D*.

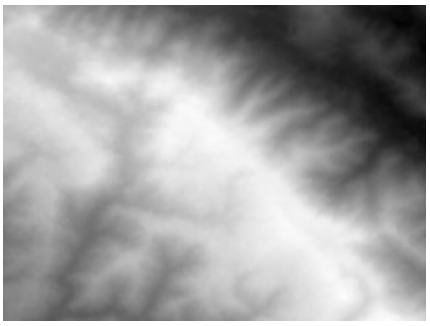
Deployment algorithms often attempt to maximize *coverage*, which is a measure of the number of visible target positions, i.e., positions at which an event can be detected by the network. We say that a target is *covered* if one or more of the network's cameras can see the target with an unobstructed LOS view. In the case of visual sensors, besides LOS, the camera's "range" must be considered, as well. Visual target resolution becomes limited as the distance to a target increases, and so one should limit the maximum camera-to-target range to some *sensor radius*. Combining visibility, i.e., unobstructed LOS view, and sensor radius leads us to the concept of *range-limited visibility*.

Range-limited visibility can thus be used to guide the deployment of visual sensor networks. For any camera node (or for any network of camera nodes), the set of all covered targets forms a *viewshed*. A common and frequent goal in visual sensor network deployments is to maximize the number of targets that comprise the viewshed—or in the case of continuous terrain, to maximize the area of the viewshed. The area of a viewshed is called its *cumulative visibility* [1]. Accounting for the constraint of maximum camera-target range yields the measure of *network fitness* that we use in this work: *range-limited cumulative visibility*.

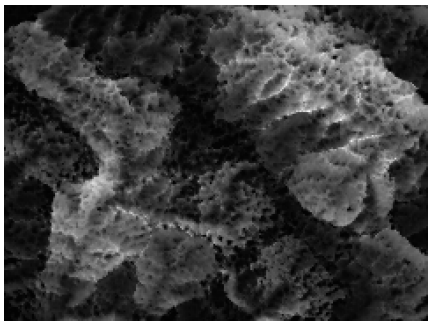
Often terrain is modeled using a grid of heights called a *height map*, where each (x, y) or $(\text{latitude}, \text{longitude})$ position has an associated terrain elevation. When using a grid-based height map, a straightforward visibility algorithm will treat each grid position as entirely visible or as entirely obstructed [2]. As such, the range-limited cumulative visibility of a deployment is given by the total number of grid positions that have visibility from at least one of the network's cameras. As discussed in more detail in Section II, the problem of achieving maximum coverage, or more specifically, maximum range-limited cumulative visibility, can be formulated as an optimization problem which aims at placing the camera nodes such as to maximize the visual sensor network's overall coverage.

Greedy algorithms often are used to solve optimization problems due to their simplicity and efficiency [3, Ch. 16]. As such, greedy algorithms usually can deliver adequate cost-performance tradeoff, i.e., they are able to yield adequate results with reasonable computational cost.

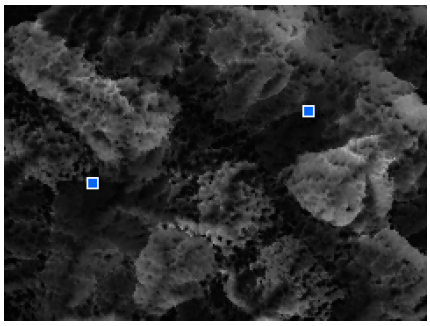
In this work we present a new distributed, greedy deployment algorithm for camera-based IoT networks. The proposed algorithm uses a fixed number of camera nodes, each of which uses information about their immediate neighbors' positions to decide its best vantage point, i.e, the position that will maximize the camera network's overall coverage. We evaluate our algorithm's effectiveness using the Cooja IoT simulator/emulator in a variety of terrain scenarios and compare the algorithm's performance against other greedy-based deployment algorithms.



(a) **Height map of terrain** (h_i). Lighter regions represent higher elevations, such as hills and plateaus, and darker regions represent lower elevations, such as valleys.



(b) **Range-limited cumulative visibility** (v_i) of the terrain of Fig. 1a. Lighter regions identify potential viewpoints that have greater range-limited cumulative visibility.



(c) **Wonderful Life Utility** (w_i) of the terrain after placing two nodes. Lighter regions identify viewpoint positions that provide the greatest *improvement* in range-limited cumulative visibility.

Fig. 1. Height map, cumulative visibility, and Wonderful Life Utility of example terrain. Comparing Fig. 1b and Fig. 1c, the darkening of the regions near the viewpoints' positions shows that there is less benefit from adding a third viewpoint close to either of the first two.

Finally, since computing cumulative visibility is computationally expensive, in addition to fitness caching, our solution uses an easy-to-compute fitness proxy to reduce the number of cumulative-visibility computations.

II. Problem Formulation

Generally speaking, the deployment of outdoor IoT camera networks is an optimization problem which aims to maximize some measure of network fitness by adjusting node positions subject to practical constraints, such as the number of camera nodes. Often the measure of network fitness is some form of coverage, which measures the network's ability to perform its functions, e.g., survey the region, detect events, etc [4]. As discussed in Section I, one measure of coverage in camera IoT networks is cumulative visibility, i.e., the total *area of terrain* that is collectively visible by the IoT camera network (when the terrain has a continuous representation) or the total *number of terrain positions* that are collectively visible by the IoT camera network (when the terrain representation is discretized). Since practical sensors have limited range, a more practical measure of coverage is range-limited cumulative visibility, where not only must the target be visible from the position of the sensor, but the target must be within the sensor's sensing range.

To illustrate the concept of cumulative visibility and range-limited cumulative visibility, Fig. 1a shows a terrain height map where bright regions of the map indicate positions of the terrain that have greater elevation. Ridges appear as bright linear regions while valleys appear as dark lines.

Fig. 1b shows the range-limited cumulative visibility corresponding to the height map of Fig. 1a for a sensor range of 50 pixels. In this figure, bright regions indicate positions that can see the greatest areas of the surrounding terrain. Compare the dark region of Fig. 1a, which indicates a large valley, to the corresponding region of Fig. 1b, which is bright and indicates positions with superior range-limited cumulative visibility. The concave-up shape of a valley means that many positions on a valley's wall can see a large portion of the opposite wall.

Paradoxically, the bright, central region of Fig. 1a, which indicates an area of greatest elevation on the height map, corresponds to a dark region in Fig. 1b. This correspondence shows that higher node elevation does not necessarily lead to greater visibility. Indeed, a sensor that is located in the middle of a high plateau will be unable to see regions below the plateau without first moving to the plateau's edge. In addition, while distant mountains might be seen from positions anywhere on the plateau, the range limitation of a sensor may prevent the sensor from usefully resolving such distant targets, hurting visibility.

We desire to design a distributed algorithm that relies only on knowledge of nearby nodes and their positions (instead of requiring global knowledge). Consequently we need a means of computing the effect that an individual node's position has on the fitness of the entire network. We use the *Wonderful Life utility* (WLU), a concept from potential games [5]. Using WLU, a node compares the global utility of the system with the global utility of an alternate world in which it doesn't exist. The difference is the node's WLU, that is, its individual contribution to global utility.

Changes in a node's WLU are the same as changes in the global objective function g as long as no two nodes change state simultaneously. That is

$$\mathrm{WLU}' - \mathrm{WLU} = g' - g \tag{1}$$

For example, Fig. 1c shows the WLU for all positions of the map after two nodes are placed at the positions indicated by blue squares. Brighter regions indicate greater WLU. Comparing the range-limited cumulative visibility of Fig. 1b and the WLU of Fig. 1c, one can see that the WLU in regions far from the placed nodes is identical to the range-limited cumulative visibility of those regions. This observation is expected because placing an additional node far from the first two will increase the fitness of the network by the range-limited cumulative visibility at the additional node's position. However, looking at regions close to the placed nodes, one

can see that the WLU is much lower than the corresponding range-limited cumulative visibility. This result is also expected since there is little benefit to locating an additional node near either of the placed nodes.

The camera network deployment problem we set out to explore can be formulated as follows: maximize line-of-site visual coverage on 2.5D terrain as guided by node WLU through a one-time deployment in which nodes are initially placed at random (instead of incremental deployment) and node-to-node communication is one-hop. Our goal is to design a distributed deployment algorithm that cooperatively determines the positions of a fixed number of homogenous, mobile, omnidirectional camera nodes [4] on the 2.5D terrain.

III. Distributed Grid Partition

The proposed Distributed Grid Partition algorithm is inspired by landform classification [6]. Two of the simplest landform classes, *depressions* and *saddles that are primarily concave up*, define regions with superior cumulative visibility. Nodes positioned within these two classes of landforms will tend to contribute effectively to network fitness. Alternatively, the two landform classes of *hills* and *saddles that are primarily convex up* define regions with inferior cumulative visibility, and nodes positioned within these regions should not be expected to contribute effectively to network fitness.

While these landform classes can help predict superior node positions, one must not ignore the physical size of the landform. For instance, a node that is in a large valley normally would be expected to improve cumulative visibility, but if the node lands in a small depression, then its view of the surrounding terrain will be completely blocked.

Following the above observations, the Grid Partition algorithm attempts to find positions on landforms that will have large viewsheds, such as valleys, while also positioning camera nodes on small hills. By doing so, the algorithm tries to reduce the number of fitness computations by avoiding positions that are inherently unfit.

To start, Grid Partition is a distributed deployment algorithm that runs on every node of the network. Each node alternates several times between transmitting and moving, terminating its search after a maximum number of iterations or after discovering over several iterations that it is unable to find an improved position. During one iteration of the algorithm's outer loop, a node broadcasts its current physical position and recent position reports that it has received from other nodes. Then the node searches for a new position with better WLU than its current position. Since the WLU computation is computationally expensive, the node limits the number of WLU computations in two ways. Firstly, it considers only candidate positions that are in a circle of radius r centered on its current position. The effect of this first limit is create a circular *exploration region*, as illustrated in Fig 2a. Secondly, the node partitions the exploration region into n_0 squares (Fig 2b) and consults terrain height data to find the highest position within each square (Fig 2c). Choosing a highest local position helps the node avoid local visual obstructions. Then the node computes the WLU at each of the identified positions (Fig 2d) and identifies a set of candidate positions with the best WLU. The final step of an iteration (Fig. 2e) is for the node to choose a new position by computing the centroid of the best candidate positions that were found in the previous step. The process is repeated by looping back to Fig. 2b.

Algorithm 1 Outer loop used in distributed algorithms.

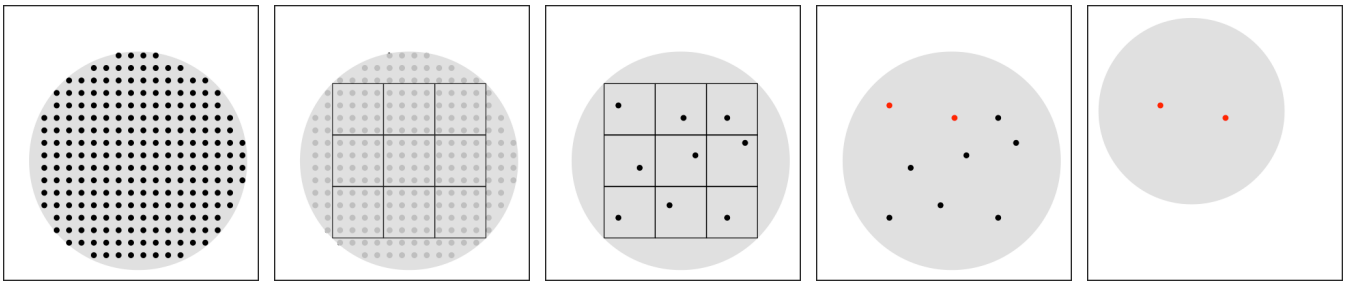
1: $x_E \leftarrow$ initial node position
2: $r_E \leftarrow R_E$
3: **for** L times **do**
4: Move to a new position x based on the chosen algorithm
5: Communicate x_E to other nearby nodes.
6: Decrease r_E {see Section V}
7: **if** improvement in the network fitness is inadequate **then**
8: **exit for loop**
9: **end if**
10: **end for**

We present our Distributed Grid Partition algorithm in detail below. Note that every participating network node runs the algorithm independently from the other nodes and uses only local information obtained from its immediate neighbors. For clarity, we describe the algorithm in two parts, namely the algorithm's outer and inner loops. We should point out that the outer loop is common to the other greedy algorithms considered in the comparative performance evaluation we conducted, whose results are presented in Section VI.

A. Outer Loop

As previously pointed out, the outer loop (Algorithm 1) is executed by all of the distributed algorithms considered in our study. These algorithms are described in Section V. In Step 4, the corresponding node placement algorithm is invoked.

In **Steps 1–2**, the node is initialized with an initial position and an exploration radius. The exploration radius is described in Section III-B below. **Steps 3–10** are the outer loop which runs until the node's fitness ceases improving for several cycles or until a maximum number of loops L is executed. The maximum loop count helps ensure that the algorithm will eventually terminate. **Step 4** calls the greedy algorithm that determines the next position for the node and moves the node to its new position. **Step 5** broadcasts the node's new position to all its neighbors. The node includes in its broadcast any current positions of its neighbors that it is aware of. In other words, this step performs one-hop signaling, which also includes the most recently received neighbor positions. We assume that at any point a node may receive updated position information from its neighbors, although in our simulations, a node will not act on received positions until it returns to Steps 4 and 5. **Step 6** decreases the exploration radius r_E, which is described in Section V below. **Steps 7–9** terminate the loop if there is inadequate fitness improvement, that is, if there is no improvement for a given number of cycles. Each node uses this strategy because the inability to find a better position may be resolved when one of the node's neighbors



(a) Exploration region. (b) Grid partition. (c) Highest positions. (d) Best positions. (e) Move, shrink, loop.

Fig. 2. Illustration of the Grid Partition algorithm (see text).

Algorithm 2 Inner loop of Grid Partition.

1: $C_E \leftarrow$ circle centered at x_E with radius r_E
2: $C \leftarrow C_E$
3: $x \leftarrow x_E$
4: $r \leftarrow r_E$
5: $x_{\text{best}} \leftarrow x$
6: **for** M times **do**
7: **loop**
8: Using a square grid, partition $C_E \cap C$ into n regions $R_1 \ldots R_n$, where $n \approx n_0$. {see text}
9: $x_i \leftarrow$ highest position within R_i **for** i in $1 \ldots n$.
10: Compute the fitness of each position $x_1 \ldots x_n$.
11: Sort positions $x_1 \ldots x_n$ by decreasing fitness.
12: $x_{\text{best}} \leftarrow x_1$ **if** x_1 is better than x_{best}
13: **if** the grid in step 8 is smaller than the DEM grid **then**
14: **exit loop**
15: **end if**
16: $x \leftarrow$ mean of positions $x_1 \ldots x_{\lfloor Fn \rfloor}$
17: $r \leftarrow Ar$
18: $C \leftarrow$ circle centered at x with radius r
19: **end loop**
20: **end for**
21: **return** x_{best}

eventually moves and allows a subsequent improvement in the node's WLU.

B. Inner Loop

The inner loop implements a greedy algorithm to compute the node's next position (Algorithm 2). This part of the algorithm accepts an initial position x_E, an exploration radius r_E, and an effort parameter n_0. An additional parameter, F, indicates the fraction of the candidate positions that will guide repositioning of the exploration region. Finally, parameter A controls the rate at which the exploration region shrinks.

The r_E parameter is used to prevent a node from traveling too far before receiving position updates from its neighbors. Since this is a distributed algorithm in which each node guides its decisions using its WLU, we want to avoid the situation where a node fails to recognize that its WLU has been substantially affected by a change in a neighbor's coverage due to the neighbor's movement.

Steps 1–5 initialize the algorithm. C_E is the initial exploration region, centered at x_E with radius r_E. The node will not leave this region. C is the current exploration region, centered at x with radius r. The current exploration region will shift and shrink as the algorithm proceeds, focusing its attention on regions around better positions. x_{best} is the best position seen so far, which at initialization is the node's current position. In **Steps 6–20** the algorithm searches for the best next position for the node. This loop runs M times, where M is a tuning parameter of the algorithm. We want to avoid values of M that are too small, which will hurt solution quality, or values that are too large, which will hurt algorithm performance. For our experiments, $M = 10$ yields a good compromise. **Steps 7–19** form an inner loop that searches within the exploration region that is centered on x_E. **Step 8** of the algorithm partitions the exploration region using a square grid. The result of this step is a square grid with approximately n_0 complete squares within the exploration region. Changing the value of n_0 will not lead to incorrect results, but too large a value will negatively affect the run time of the algorithm, while too small a value will negatively impact solution quality. In Section VI we characterize the effect of changing the value of n_0. **Steps 9–12** evaluate the fitness of the highest position within each grid square and record the best position seen so far. Recall that the fitness of a position is its WLU, that is, the contribution that the node would provide to the range-limited cumulative visibility of the network if the node moved to that position. **Steps 13–15** terminate the loop when the algorithm degenerates to exhaustive search of the exploration region. **Steps 16–18** shift the current exploration region to the mean of the top $\lfloor Fn \rfloor$ positions just evaluated and shrinks the exploration region. This reduction in the exploration area while maintaining the value of n_0 increases the density of the evaluated positions. **Step 21** returns the best position found.

IV. Related Work

The placement of nodes with the goal of covering targets is a well-known problem with applications in a variety of other disciplines. For instance, in the *Art Gallery Problem* [7] the floorspace of an art gallery is "guarded" by locating cameras or guards. Variations of the problem include the inclusion of "holes" in the floorspace (which can be considered columns

in the gallery), and guarding the outside of the building rather than guarding the inside. Since a floorspace is 2D, this problem is different than ours. A 3D version of the Art Gallery problem considers the visibility of a polyhedron's *interior volume*, but once again we are concerned with a different problem: the visibility of a polyhedron's upper *surface*.

In operations research, the Location Set Covering Problem (LSCP) strives to minimize the number of facilities needed to satisfy 100% of a 2D area's demand [8]. LSCP can be formulated as a zero-one linear-programming problem with one variable for each node's candidate position and one equation for each target whose coverage is desired [9]. As LSCP assumes an unbounded number of nodes, we do not consider it further.

Related to LSCP, the Maximal Covering Location Problem (MCLP), strives to identify positions for a fixed number of facilities that will maximize the total demand that is satisfied [10]. This problem can be addressed by a Greedy Set Cover heuristic that stops after placing all available nodes. We include the Set Cover heuristic in our evaluation as an upper bound in our evaluation of Grid Partition.

Coverage in the area of wireless sensor networks (WSNs) has received considerable attention from the research community. In particular, the work in [11] provides a thorough survey of the state-of-the-art in coverage over terrain. The various approaches reviewed represent terrain using three different schemes: contrived mathematical equations, regular matrix terrain, and triangulated irregular networks (TIN). In our work, we represent terrain as a regular matrix for simplicity. Using a TIN terrain representation is one direction of future work we plan to explore. WSN coverage models can be binary or probabilistic. While we use a binary coverage model in our evaluation, a probabilistic model could be used instead. WSN research of terrain coverage has used several optimization strategies, including simulated annealing, genetic algorithms, and the CMA-ES variety of Evolution Strategy [12]. Greedy heuristics, while sometimes used, are less common.

While there has been much work on WSN coverage, our work is different because it is one of the few that uses a greedy heuristic, and it includes an easy-to-compute fitness proxy to improve algorithm performance. In addition, use of WLU as a fitness function lets our algorithm make sound judgements without needing to have a global understanding of the network.

While Distributed Grid Partition shares range-limited cumulative visibility with some existing work [11], including ours [13], it introduces the idea of using terrain height as an easy-to-compute fitness proxy that reduces the number of fitness computations. In addition, to the best of our knowledge, Distributed Grid Partition is the first to use the WLU concept to reduce communication and computation cost.

V. Experimental Methodology

A. Simulation Platform, Datasets, and Parameters

To evaluate our Distributed Grid Partition approach, we implemented the algorithm on the Cooja-Contiki IoT simulator/emulator [14]. The Cooja simulator supports user-definable nodes in both homogeneous and heterogeneous wireless sensor networks. Cooja nodes can communicate using standard or user-provided protocols and propagation models. One describes a node's operation using emulated microcontroller binaries, compiled C code via a JNI interface, or native Java code.

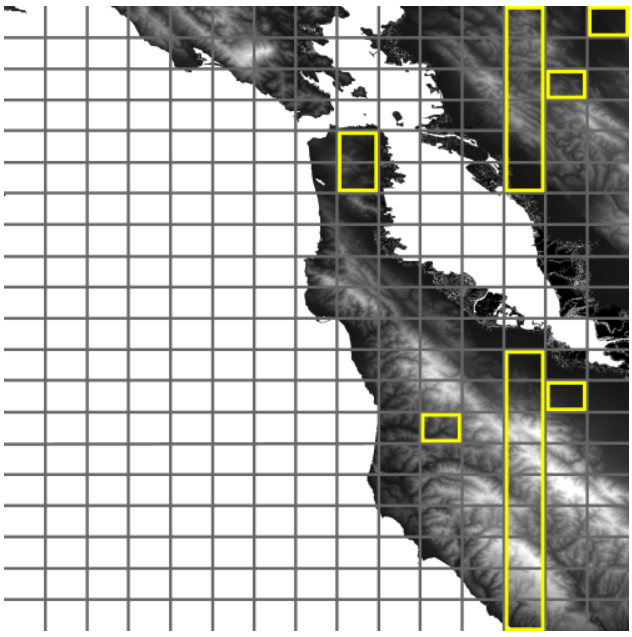


Fig. 3. Selected subtiles of SRTM tile N37W123.

We chose to use native Java code, which Cooja calls an "application level" node, because it avoids the memory limitations of ordinary Cooja nodes that run emulated MSP430 binaries. Specifically, the model of our node uses the *Abstract-Application* Mote class.

We use Cooja's radio medium which employs a simple model, i.e., limits packet reception based on propagation distance, to simulate communication between nodes over the wireless medium. As future work, we plan to investigate more realistic propagation models.

For sample deployment regions, we use terrain datasets from NASA's 2005 Shuttle Radar Topography Mission (SRTM) [15]. This mission collected elevation, or height, data for much of the landmass of the Earth and provides this information in datasets called *digital elevation models* or DEMs [1]. The DEMs provide the height for positions on the Earth using latitude-longitude coordinates that have a precision of 1 arc second, which is about 30 meters (or 100 feet) in the latitude direction.

The datasets are organized into "tiles" of 3600×3600 arcsecs (1 degree of longitude by 1 degree of latitude). We selected tile N37W123, which is illustrated in Fig. 3, because it includes regions that represent a wide variety of landform shapes. To evaluate the effect of landform shapes on the algorithms' performance, we divided the full tile's 13 million elevation samples into smaller, more manageable 240×180 arcsec subtiles. We selected the 21 subtiles indicated in the figure, as they represent a variety of terrains. Our strategy for

Algorithm 3 Pattern Search.

1: $C_E \leftarrow$ circle centered at x_E with radius r_E
2: $x \leftarrow x_E$
3: $r \leftarrow r_E$
4: **while** $r > r_{\text{min}}$ **do**
5: $x_1 \leftarrow x + (0, r)$
6: $x_2 \leftarrow x - (0, r)$
7: $x_3 \leftarrow x + (r, 0)$
8: $x_4 \leftarrow x - (r, 0)$
9: Compute the fitness of $x, x_1 \ldots x_4$.
{Assign fitness 0 to any x_i that falls outside C_E.}
10: **if** x is more fit than all x_i **then**
11: $r \leftarrow r/2$
12: **else**
13: $x \leftarrow$ fittest of $x_1 \ldots x_4$
14: **end if**
15: **end while**
16: **return** x

TABLE I
ALGORITHM PARAMETERS AND THEIR VALUES

Parameter	Value(s)	Parameter	Value(s)
A	0.9	M	10
F	0.25	n_0	$5, 10, 21$
L	10	R_E	$10, 15, 21, 34, 51, 76$

selecting subtiles is validated later in Section VI.

As mentioned above, the coordinates of the SRTM data are measured in arc seconds of latitude and longitude, but our algorithm uses Cartesian coordinates in arbitrary distance units. One normally would perform a coordinate conversion through a resampling of the DEM height data, which would introduce some interpolation errors in the heights. However, since we are not performing field verification, Earth-accurate coordinates are not necessary. Instead, we treat the SRTM data as if it had been measured at the Earth's equator and interpret each DEM sample as representing a flat 33×33 meter patch of terrain.

For each algorithm considered in our study, we ran numerous simulations per terrain subtile. Our experiments deploy 10 camera nodes, but our algorithm can accommodate any number. Each data point reported in Section VI is an average over ten different sets of random starting positions for the nodes.

We also evaluate the effect of R_E, the initial value of the exploration radius, on the distributed algorithms, including Distributed Grid Partition. Recall that r_E defines the region a camera node will explore as it tries to place itself such that it maximizes its contribution to the camera network's overall coverage. We use six different values of R_E, namely 10, 15, 21, 34, 51, and 76 arcsec. In Step 6 of the outer loop (Algorithm 1) r_E sequentially decreases in value to focus the algorithm on more promising regions. For our current experiments, we use the following sequence of r_E values: $0.92R_E$, $0.85R_E$, $0.77R_E$, $0.69R_E$, $0.62R_E$, $0.54R_E$, $0.46R_E$, $0.38R_E$, and $0.31R_E$.

We set the sensor radius to 50 arcsec since this value allows 100% coverage of a 240×180 arcsec subtile with proper placement on a flat terrain. We set the communication range to 130 arcsec, leaving the consideration of more realistic radio propagation models for future work. We set both of the maximum-loop limits L and M to 10, and we set $A = 0.9$ and $F = 0.25$. For a subset of experiments, we evaluate three different values of n_0: 5, 10, and 21. The parameter values used in the experiments reported in Section VI below are summarized in Table I. As future work, we will explore the sensitivity of the algorithm to the values of parameters $L, M, A,$ and F.

B. Other Deployment Algorithms

Since Distributed Grid Partition is a greedy algorithm, we compared it with two other greedy algorithms: Pattern Search and Gradient Descent. For our comparative performance study, we implemented distributed versions of both algorithms, which are described below.

We also implemented a centralized Set Cover algorithm which is also described in this section. We use Centralized Set Cover as our performance upper bound because it is centralized, and because it does not attempt to limit the number of fitness computations, as Grid Partition does.

Finally, in addition to the algorithms just mentioned, we ran simulations for "strawman" deployment approaches, namely Random Placement, with 100 random seeds per subtile, as well as placing nodes on a regular triangular grid.

1) Distributed Pattern Search: We implemented the classic Pattern Search algorithm [16] [17, §9.3]. See Algorithm 3. The algorithm accepts an initial position x_E and an exploration radius r_E. In its termination condition, it uses a value r_{min}, which is the horizontal resolution of the terrain's height map.

Steps 1–3 initialize the algorithm, accepting a starting exploration region that is defined by x_E and r_E. The value of r determines when the algorithm will terminate. **Steps 4–15** loop until $r \leq r_{\text{min}}$. **Steps 5–8** compute four candidate next-positions: $x_1 \ldots x_4$. These positions are r away from the current position x in all four cardinal directions. **Step 9** computes the fitness of all of the positions under consideration, using the special fitness value 0 for any position that falls out of the exploration region. Using this special value ensures that the algorithm always chooses positions that are inside the exploration region. **Steps 10–14** either move x to the fittest position or, if x already is in the fittest position, cut r in half. **Step 16** returns the best position.

2) Distributed Gradient Descent: We implemented *Gradient Descent*, whose goal is to minimize negative fitness. Gradient descent is an iterative optimization algorithm that chooses a sequence of step directions based on an objective function's derivative [17, ch. 3]. Although our WLU fitness function lacks a derivative, it is possible to use gradient descent with a derivative approximation. We describe the approximation and our algorithm below.

A straightforward method of approximating a derivative is to use finite differences, where one evaluates a function at two points and then solves for a slope [17, §8.1]. During an initial

Algorithm 4 Gradient Descent.

1: $C_E \leftarrow$ circle centered at x_E with radius r_E
2: $x_{\text{next}} \leftarrow x_E$
3: **repeat**
4: $x \leftarrow x_{\text{next}}$
5: Fit a plane to the fitness of the 25 positions around x.
6: **if** the plane is level **then**
7: **exit repeat-until loop**
8: **end if**
9: $x_{\text{next}} \leftarrow$ position adjacent to x in the direction of the plane's steepest upward slope
10: **until** x_{next} is outside C_E or x_{next} already has been visited
11: **return** x

test of this method, we discovered that noise in the cumulative visibility function passes into the derivative approximation and interferes with the ability of the gradient descent algorithm to find a good solution.

So instead, to smooth its derivative approximation around a point, our algorithm uses least-squares estimation [18, p. 363]. A plane is fit to a set of 25 WLU values $f_i = f(x_{i,0}, x_{i,1})$ arranged in a 5×5 grid that is centered on the position of interest. Then an optimization step is taken in the direction of the plane's steepest ascent (b_0, b_1) by solving the linear equation

$$\begin{bmatrix} \sum x_{i,0} f_i \\ \sum x_{i,1} f_i \end{bmatrix} = \begin{bmatrix} \sum x_{i,0}^2 & \sum x_{i,0} x_{i,1} \\ \sum x_{i,0} x_{i,1} & \sum x_{i,1}^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \quad (2)$$

The 5×5 arrangement of points simplifies (2) when we use a spacing that matches the DEM data. Our algorithm considers the positions shifted so that $x_{i,0} \in \{-2, -1, 0, 1, 2\}$ and $x_{i,1} \in \{-2, -1, 0, 1, 2\}$. Then $\sum x_{i,0} x_{i,1} = 0$, which we can substitute into (2), and also $\sum x_{i,0}^2 = \sum x_{i,1}^2$, which we use for further simplification. Because our algorithm relies on the gradient (b_0, b_1) for the *direction* of steepest ascent and not its magnitude, we can scale both b_0 and b_1 by $\sum x_{i,0}^2$ without affecting the direction. This change uses a simpler gradient (b'_0, b'_1) that yields the same ascent direction.

$$\begin{bmatrix} \sum x_{i,0} f_i \\ \sum x_{i,1} f_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b'_0 \\ b'_1 \end{bmatrix} \quad (3)$$

We then can solve directly for (b'_0, b'_1).

$$(b'_0, b'_1) = \left(\sum x_{i,0} f_i, \sum x_{i,1} f_i \right) \quad (4)$$

Finally, using two-argument arctangent, we compute the direction of the steepest ascent of WLU.

$$\theta = \texttt{atan2} \left(\sum x_{i,1} f_i, \sum x_{i,0} f_i \right) \quad (5)$$

See Algorithm 4. The Gradient Descent algorithm accepts an initial position x_E and an exploration radius r_E. The algorithm uses (5) in Step 9.

Steps 1–2 initialize the algorithm. C_E is the initial exploration region, centered at x_E with radius r_E. The node will not leave this region. x_{next} is set to a value that readies the repeat-until loop. **Steps 3–10** form a loop that moves the node along the computed WLU gradient. The loop will terminate either when the slope of the computed WLU gradient is 0, or when the algorithm attempts to revisit a position that it already had visited in a prior loop iteration. (The derivative approximation uses 25 points in a 5×5 grid. We evaluated using 9 points in an earlier version of this algorithm and were not satisfied with the results.) **Step 11** returns the best position found.

Algorithm 5 Centralized Set Cover.

1: **for** every node i **do**
2: Place node i at the position that best improves network fitness.
3: **end for**

3) Random Placement: For the Random Placement algorithm, ten nodes are placed according to a uniform random distribution. This exercise is repeated 100 times and the average of the range-limited cumulative visibility is reported.

4) Regular Grid: For the Regular Grid algorithm, ten nodes are placed in a triangular grid on a subtile. The range-limited cumulative visibility is computed for that deployment.

5) Centralized Set Cover: The deployment problem that we address differs from the set cover optimization problem in that instead of minimizing the number of nodes needed to achieve 100% coverage, our deployment problem is to maximize coverage using a limited number of nodes.

We implemented the Greedy Adding algorithm [10], which is a Centralized Greedy Set Cover heuristic [3, §35.3] that has been modified to stop after placing all nodes of the network (Algorithm 5). Fig. 4 shows a demonstration of the set cover algorithm as it finds positions for ten nodes.

VI. Experimental Results

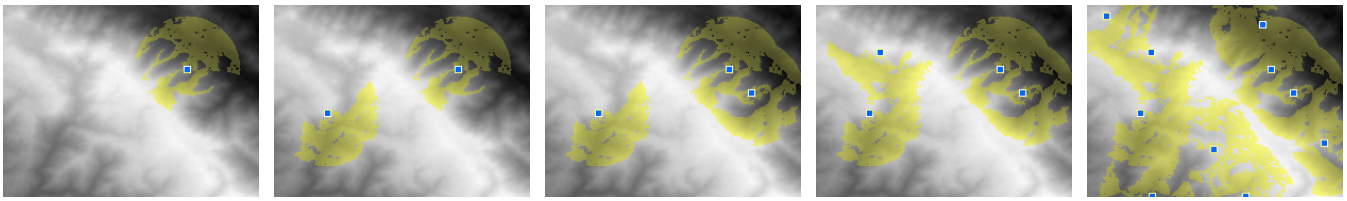
In this section, we show performance results of our Distributed Grid Partition deployment algorithm compared against Centralized Set Cover and distributed versions of Pattern Search and Gradient Descent. We also compared all algorithms to random- and triangular grid placement.

Since the results of Distributed Grid Partition, Distributed Pattern Search, and Distributed Gradient Descent all depend on the initial node positions, we ran these algorithms under the same conditions: for each subtile we ran each algorithm ten times using ten different initial node placements.

Random placement was evaluated using 100 seeds, and the results for each subtile were averaged over these 100 runs. Placement on a regular grid was performed only once per tile since the algorithm has no random aspect in its operation.

We start by presenting the results of our comparative study followed an analysis of the algorithms' parametric sensitivity.

Fig. 5 compares the coverage provided by all algorithms considered in our study. Each data point represents the range-limited cumulative visibility as determined by the indicated algorithm for one of the 21 terrain subtiles. The horizontal axis is the range-limited cumulative visibility that Centralized Set Cover computes for each subtile. We note that the uniform horizontal distribution of data points suggests that the 21 subtiles that we selected represent a sufficient variety of terrain



(a) Placing the first node. (b) Adding the second node. (c) Adding the third node. (d) Adding the fourth node. (e) All ten nodes.

Fig. 4. Choosing node positions using Greedy Set Cover for ten nodes. In height map (a), the colored terrain shows the coverage that is provided by the first node, which is placed at the position with the largest viewshed. Height map (b) shows the improved coverage that is provided after the algorithm positions the second node without moving the first. This is the position that provides the second node with the greatest WLU ("wonderful-life utility"). Height maps (c) and (d) show improvements in coverage that are seen after the algorithm positions the third and fourth nodes, respectively. Height map (e) shows the final coverage that the algorithm achieves after it positions all ten nodes.

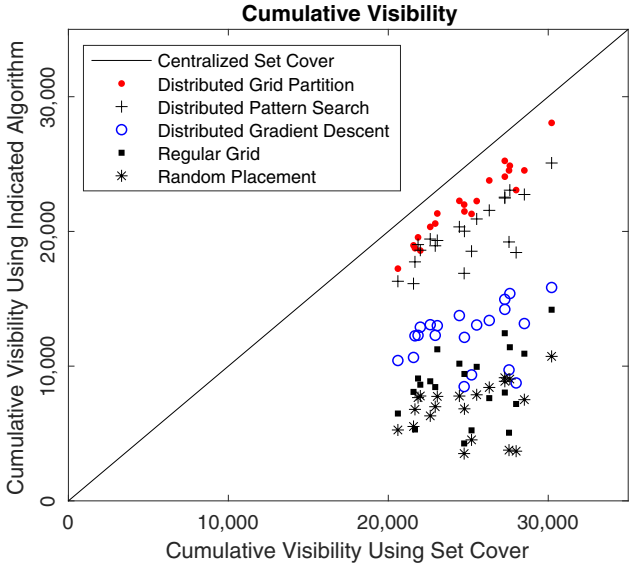


Fig. 5. Results of algorithms compared to the Centralized Set Cover Heuristic.

in the $20,000$ to $30,000$ cumulative visibility range. The vertical axis is the range-limited cumulative visibility that the other algorithms computed for each subtile. The 45° line indicates the results for Centralized Set Cover. Considering the algorithms evaluated, as expected, Centralized Set Cover is the best. Our Distributed Grid Partition algorithm comes in second, appearing as red dots immediately under the 45° line. The figure shows the results of the remaining algorithms clearly are ranked from best to worst as Distributed Pattern Search, Distributed Gradient Descent, Regular Grid, and Random Placement.

Table II provides a quantitative comparison between each algorithm and Set Cover by computing each algorithm's result relative to Set Cover for each subtile and then averaging these relative results over all subtiles.

In these experiments, all three distributed algorithms use $R_E = 51$. Remember that this parameter helps prevent nodes from moving too far during a single iteration of the distributed algorithms' outer loop. Having too small a value of R_E will hurt an algorithm's results, as a node is unable to travel a sufficient distance to discover its best position. On the other hand, too large a value of R_E could let a pair of nodes

TABLE II
AVERAGE ALGORITHM RESULTS COMPARED TO SET COVER

Set Cover	Grid Partition	Pattern Search	Gradient Descent	Regular Grid	Random Placement
100%	90%	82%	58%	44%	36%

"overshoot" their best positions as they move toward each other, each node being unaware of the other node's effect on its own WLU. Once discovering the overshoot, the nodes will need to retreat, which increases the algorithm's execution time.

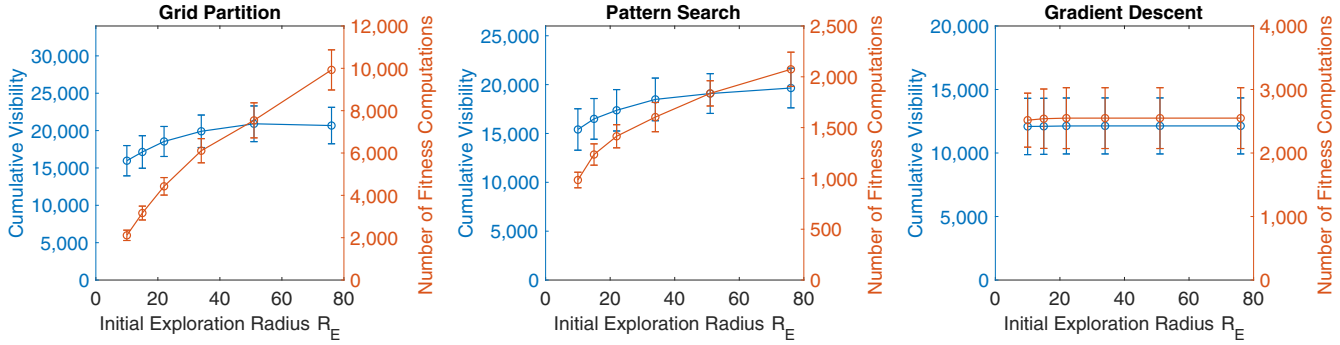
We explored the sensitivity of Distributed Grid Partition, Distributed Pattern Search, and Distributed Gradient Descent to different values of $R_E \in \{10, 15, 21, 34, 51, 76\}$. Fig. 6 shows the average range-limited cumulative visibility for 210 runs at each value of R_E. (Ten runs on each of the 21 subtiles.) As a measure of execution time, the plots also show the number of uncached fitness computations for the different values of R_E.

Fig. 6a shows that, for these terrain samples, Distributed Grid Partition is well tuned with $R_E = 51$ since either increasing or decreasing R_E hurts the average result.

Similarly, Fig. 6b shows that Distributed Pattern Search also performs well at $R_E = 51$, since an increase in R_E yields only a small improvement in the average range-limited cumulative visibility but a more pronounced increase in execution time.

Finally, Fig. 6c illustrates that Distributed Gradient Descent is essentially unaffected by the value of R_E. Since Gradient Descent performs best when its gradient is smooth, we suspect that Gradient Descent's insensitivity to R_E is because the algorithm gets "stuck" when it responds to "noise" in the gradient approximation.

Recall that the n_0 parameter controls the amount of effort performed by Grid Partition. Specifically, n_0 is the number of grid squares that the algorithm tries to partition the exploration region into, where each square will cause a fitness computation. So a larger value of n_0 will cause the algorithm to examine the WLU of more positions, resulting in longer run times. A smaller value for n_0 will reduce run time but may hurt coverage by examining too few positions. Fig. 7 shows that our chosen value $n_0 = 10$ provides adequate cost-performance trade-off for the sample terrains examined.



(a) Grid Partition (b) Pattern Search (c) Gradient Descent

Fig. 6. Comparison of Range-Limited Cumulative Visibility and the number of uncached Fitness Computations in the distributed algorithms that were tested.

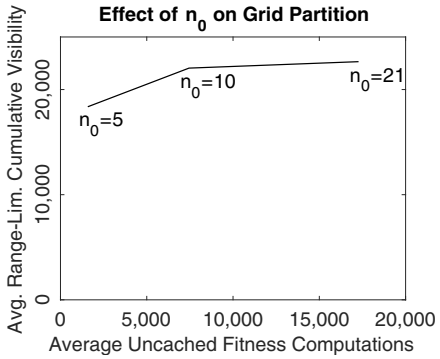


Fig. 7. Effect of the value of n_0 on Grid Partition results.

VII. Conclusion and Future Work

In this paper we proposed a novel distributed greedy algorithm we call Distributed Grid Partition. Extensive experimentation shows that the proposed algorithm yields superior performance when compared to other distributed algorithms as well as centralized Set Cover. Using the WLU lets nodes estimate the effect of their actions on network fitness, and using an easy-to-compute fitness proxy helps reduce the number of fitness computations needed.

We measured Grid Partition's sensitivity to primary control parameters, and as one direction of future work, we plan to extend the sensitivity analysis to confirm the effects of other parameters of the algorithm. We also plan to consider the effect of more realistic radio propagation models.

References

[1] K.-t. Chang, *Introduction to geographic information systems*, 6th ed. McGraw-Hill, 2012.

[2] J. Wang, G. J. Robinson, and K. White, "Generating viewsheds without using sightlines," *Photogrammetric Engineering & Remote Sensing*, vol. 66, no. 1, pp. 87–90, January 2000.

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

[4] M. Senouci and A. Mellouk, *Deploying Wireless Sensor Networks: Theory and Practice*. London: ISTE Press Ltd, 2016.

[5] H. P. Young, *Individual Strategy and Social Structure: an Evolutionary Theory of Institutions*. 41 William St, Princeton, NJ 08540: Princeton University Press, 1998.

[6] R. MacMillan and P. Shary, "Landform and landform elements in geomorphometry," in *Geomorphometry: Concepts, Software, Applications*, ser. Developments in Soil Science, T. Hengl and H. Reuter, Eds. Amsterdam: Elsevier, 2009, vol. 33, ch. 9, pp. 227–254.

[7] J. O'Rourke, *Art Gallery Theorems and Algorithms*, ser. International Series of Monographs on Computer Science. New York: Oxford University Press, 1987, vol. 3.

[8] C. Toregas, R. Swain, C. ReVelle, and L. Bergman, "The location of emergency services facilities," *Operations Research*, vol. 19, pp. 1363–1373, 01 1971.

[9] C. Toregas and C. Revelle, "Binary logic solutions to a class of location problem," *Geographical Analysis*, vol. 5, pp. 145 – 155, 09 2010.

[10] R. Church and C. ReVelle, "The maximal covering location problem," *Papers of the Regional Science Association*, vol. 32, pp. 101–118, 12 1974.

[11] M. Zafer, M. R. Senouci, and M. Aissani, "Terrain partitioning based approach for realistic deployment of wireless sensor networks," in *Computational Intelligence and Its Applications*, A. Amine, M. Mouhoub, O. Ait Mohamed, and B. Djebbar, Eds. Springer, 2018, pp. 423–435.

[12] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[13] K. Veenstra and K. Obraczka, "Guiding Sensor-Node Deployment Over 2.5D Terrain," in *Proceedings of IEEE International Conference on Communications*, June 2015, pp. 6719–6725.

[14] F. Österlind, "A sensor network simulator for the contiki os," Swedish Institute of Computer Science, Tech. Rep. T2006:05, 2006.

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

[16] R. Hooke and T. A. Jeeves, "Direct search solution of numerical and statistical problems," *JACM*, vol. 8, no. 2, pp. 212–229, 1961.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research and Financial Engineering. Springer Science+Business Media, LLC, 2006.

[18] T. H. Wonnacott and R. J. Wonnacott, *Introductory Statistics for Business and Economics*, 2nd ed. John Wiley & Sons, Inc., 1977.