UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DEPLOYMENT ALGORITHMS FOR OUTDOOR IOT NETWORKS OVER 2.5D TERRAIN

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Kerry Scott Veenstra

June 2022

The Dissertation of Kerry Scott Veenstra
is approved:

Professor Katia Obraczka, Chair

Professor J. J. Garcia-Luna-Aceves

Professor Chen Qian

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures vi

List of Tables ix

Abstract x

Acknowledgments xii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Terminology . . . 2
  - 1.2.1 Non-contact Sensors . . . 2
  - 1.2.2 Visibility, Viewshed, Coverage . . . 3
  - 1.2.3 Wonderful Life Utility . . . 5
  - 1.2.4 Commsheds . . . 8
- 1.3 Problem Formulation . . . 9
- 1.4 Contributions . . . 12
- 1.5 Organization of Dissertation . . . 13

2 Evaluation Framework 14
- 2.1 Terrain . . . 14
  - 2.1.1 Terrain Representation . . . 15
  - 2.1.2 Terrain Benchmarks . . . 16
  - 2.1.3 Terrain Classification . . . 18
- 2.2 Terrain-Aware Framework For IoT . . . 19
  - 2.2.1 TAFFI/Java Operation . . . 19
  - 2.2.2 TAFFI/C++ Operation . . . 21
  - 2.2.3 Benchmark Algorithms . . . 22
- 2.3 Implementation Details . . . 22

2.3.1 TAFFI/Java Implementation . . . 22
2.3.2 TAFFI/C++ Implementation . . . 24
2.4 Discussion . . . 25
2.4.1 Evaluation of Terrain Classifiers . . . 25
2.4.2 Reproducibility . . . 27
2.5 Contributions . . . 28

3 Outdoor IoT Deployment Algorithms 30
3.1 Related Work . . . 30
3.2 Taxonomy of Algorithms . . . 32
3.2.1 Iterative Improvement Algorithms . . . 32
3.2.2 Population-Based Algorithms . . . 33
3.3 Simulated Annealing . . . 35
3.4 Grid Partition . . . 39
3.5 EMNA/global,steps . . . 45
3.6 Evaluation Methodology . . . 48
3.6.1 Benchmark Algorithms . . . 48
3.6.2 Communication, Exploration, and Sensing Radii . . . 56
3.6.3 Parameters . . . 59
3.7 Results . . . 60
3.7.1 Algorithm Performance . . . 61
3.7.2 Exploring the Exploration Radius . . . 65
3.8 Discussion . . . 67

4 Outdoor IoT Recovery Algorithms 69
4.1 Related Work . . . 69
4.1.1 Proactive Methods . . . 70
4.1.2 Reactive Methods . . . 71
4.1.3 Hybrid Methods . . . 72
4.2 Terrain Aware Recovery with Commshed Intersections . . . 74
4.2.1 Pre-Failure Planning . . . 77
4.2.2 Failure Response . . . 80
4.3 Evaluation Methodology . . . 81
4.3.1 Experimental Evaluation Infrastructure . . . 81
4.3.2 Benchmark Networks . . . 83
4.3.3 Experiments . . . 85
4.4 Results . . . 86

5 Prior Accomplishments 91

6 Conclusions **93**
6.1 Contributions Revisited 93
6.2 Future Work 94

A Additional Plots **97**

B Custom C++ Programs **102**
B.1 Generating Synthetic Terrains 103
B.2 Centralized Deployment Algorithms 104
B.3 Generating Recovery Test Cases: CGA 105
B.4 Generating Recovery Test Cases: Topology 106
B.5 Recovery Algorithm TARCI 107

C Perimeter and Area of Intersecting Circles **109**

D Size of the Grid Partition **113**

Bibliography **116**

List of Figures

1.1 Viewshed of a two-node network on 2.5D terrain. Lighter areas represent regions with higher elevation, and darker areas represent regions with lower elevation. Blue squares represent the positions of nodes. Yellow areas indicate the positions of targets that have CLOS (clear line of sight) visibility to at least one of the nodes. 4

1.2 On the left is a *Height Map* of terrain. Lighter regions represent higher elevations, such as hills and plateaus, and darker regions represent lower elevations, such as valleys. On the right is a *Coverage Map* showing the range-limited cumulative visibility of the terrain. Lighter regions identify potential node positions that would provide greater coverage (range-limited cumulative visibility). 5

1.3 On the left is a *Coverage Map*. On the right is a corresponding *Wonderful-Life Utility Map* that results from placing a node at the position of the blue square. Brighter parts of the WLU Map indicate positions where placing a second node would improve network coverage the most. 6

1.4 This three-node example shows how changes in Wonderful Life Utility (WLU) are the same as changes in global utility. The total area of the figure represents the system's global utility (g). The two open circles represent the coverage of the two outermost nodes, and the central, dark region represents the Wonderful Life Utility of the central node. If the central node were to move, the change in the total area would be the same as the change in the area of the dark region, or $\Delta g = \Delta\text{WLU}$. 7

1.5 Commshed of a network. The gray region indicates positions that are within distance R_C of at least one node of the network. A new node that is placed anywhere in the commshed can communicate with the network. (For clarity, this figure uses a simple 2D disk radio-propagation model, but commsheds can be defined using any radio-propagation model.) 9

1.6 Commsheds of a partitioned network. The central, roughly triangular gray region indicates the intersection of all of the partitions' commsheds. A new node that is placed anywhere in the three-way intersection can communicate with all three partitions and will repair the network. 10
1.7 Example of network recovery through node mobility. 11

2.1 Selected subtiles of SRTM tile N37W123. 17

3.1 Placing a node at a low elevation in a valley can lead to good coverage. . . 39
3.2 Placing a node at a high elevation on a plateau can lead to poor coverage. . 39
3.3 Nodes must consider local elevation in order to avoid view-blocking depressions. 40
3.4 Illustration of the Grid Partition algorithm (see text). 42
3.5 Demonstration of the effect of the reference mean on an EDA algorithm's covariance computation. Fig. 3.5a marks the prior generation's mean with a + and plots line segments from the current generation of fittest individuals to their mean, which is used in their covariance computation. The resulting $\text{EMNA}_{\text{global}}$ covariance matrix is indicated by a 50% isodensity ellipse. Notice that the search tends to deviate from the direction of improvement. Fig. 3.5b plots the same individuals as in Fig. 3.5a, but line segments indicate that the alternate covariance computation uses the *prior* generation's mean +, a strategy employed by CMA-ES[21]. The 50% isodensity ellipse of $\text{EMNA}_{\text{global,steps}}$ is centered on the mean of the fittest individuals ×, with the ellipse's shape determined by the alternate covariance matrix. The ellipse shows how the $\text{EMNA}_{\text{global,steps}}$ search better continues in the direction of improvement. 46
3.6 Results of algorithms on test terrains vs. centralized Greedy Adding—10 nodes. 60
3.7 Results of algorithms on test terrains vs. ACV of terrain—10 nodes. 61
3.8 Cumulative visibility compared to Greedy Adding vs. 10, 20, and 30 nodes. 64
3.9 Effect of the value of n0 on Grid Partition results. 65
3.10 Comparison of Range-Limited Cumulative Visibility and the number of uncached Fitness Computations in the distributed algorithms that were tested. 68

4.1 Conceptual example of using commshed intersections to identify candidate positions for a recovery node. For clarity this example uses the 2D disk model, but one could extend the example to 2.5D by replacing the disk model with a terrain-based range-limited CLOS model, similar in appearance to the viewsheds shown in Fig 1.1. 74
4.2 Change in coverage boxplot. 88
4.3 Change in coverage vs distance. 89
4.4 Change in coverage vs num positions. 90

A.1 Results of algorithms on test terrains vs. centralized Greedy Adding—20 nodes. 98
A.2 Results of algorithms on test terrains vs. ACV of terrain—20 nodes. 99
A.3 Results of algorithms on test terrains vs. centralized Greedy Adding—30 nodes. 100
A.4 Results of algorithms on test terrains vs. ACV of terrain—30 nodes. 101

C.1 The perimeter of a circle-circle intersection: $p = s + s_E$. 110
C.2 Area arithmetic used for computing the area of a lens-shaped circle-circle intersection. The areas of the circular sectors in (a) and (b) are easy to compute and are summed into (c), but with the darker region double counted. We obtain the area of the lens-shaped region in (f) by subtracting the two triangular regions in (e). 111

List of Tables

2.1 Terrain Classifications as Predictors of Algorithm Performance. 26

3.1 **Control Parameters.** I ran simulation using these values of control parameters. When sweeping values of m, I set $R_E = 15$ and $n_0 = 10$. When sweeping values of R_E, I set $m = 10$ and $n_0 = 10$. When sweeping values of n_0, I set $m = 10$ and $R_E = 15$. 59
3.2 Distributed 2.5D IoT Deployment Algorithms Performance: Cumulative Visibility and Number of Fitness Computations. Results are averages over all 21 benchmarks. For comparison, visibility results appear as a percentage of Centralized Greedy Adding. 62

4.1 Message Types 77
4.2 Control Parameters 86
4.3 Pre-Failure Planning CPU Time 87

Abstract

Deployment Algorithms for Outdoor IoT Networks Over 2.5D Terrain

by

Kerry Scott Veenstra

Outdoor IoT networks are used for a range of applications including surveillance and environmental monitoring. The nodes of such networks sense environmental data and, through wireless networks that interconnect them, transmit these measurements to data collection sites known as sinks. Much research in this area focuses on 2D deployments where nodes must remain coplanar and 3D deployments where nodes can move freely through the air or underwater. However outdoor deployments where nodes remain on the surface of terrain cannot be modeled as either 2D or 3D because the models fail to account for obstructions to communication and coverage that are caused by the terrain itself. Such "2.5D deployments" require terrain-aware deployment tools and algorithms that are not yet well developed. In this dissertation I present my work on such tools and algorithms.

First, I developed two novel terrain-aware network deployment algorithms. Grid Partition accounts for visibility over 2.5D terrain, but it also improves performance by using elevation as a visibility proxy that vastly reduces the number of more costly visibility computations. The other algorithm, $\text{EMNA}_{\text{global,steps}}$, improves on $\text{EMNA}_{\text{global}}$, an estimation-of-distribution algorithm, by computing the covariance matrix of the solution population's distribution by referencing the mean of the previous generation instead of the

mean of the current generation's mean. Comprehensive experiments show that the proposed algorithms perform as well as or better than traditional optimization meta-heuristics, measured as network sensor coverage. In addition, the core idea behind one of the algorithms, Grid Partition, can serve as a meta-heuristic itself and is used in one of my recovery algorithms.

Next, I developed and evaluated a class of terrain-aware network recovery algorithms which guide networks in self-repair following a node failure. Terrain-Aware Recovery with Commshed Intersections (TARCI) consists of four algorithm variants. Experiments show that the variant that incorporates Grid Partition as a meta-heuristic (TARCI-GP) achieves nearly the same quality of results as a variant that includes an exhaustive step while requiring two orders of magnitude less CPU time. In addition, I show that my terrain-aware recovery algorithms perform better than recovery algorithms that are intended for 2D deployments.

Finally, to help with the development of my terrain-aware algorithms, I created a terrain-aware experimental framework. The Terrain-Aware Framework For IoT (TAFFI) simulates Java- and C++-based algorithms, both distributed and centralized. The framework provides a library of line-of-sight algorithms and a suite of benchmark terrains. To ensure that the benchmarks include a wide range of landform variation, another contribution of this dissertation is the design of two terrain classifiers that ensure that the benchmark terrains used in the experiments represent a wide variety of terrain landforms.

Acknowledgments

I'd like to thank my advisor Katia Obraczka for our numerous illuminating conversations, Jorge Cortés of UCSD for introducing me to Potential Games and Wonderful Life Utility, Sam Mansfield for our joint work on Average Cumulative Visibility, and Professor J. J. Garcia-Luna-Aceves and Professor Chen Qian for offering their time to serve on my reading commmittee.

Chapter 1

Introduction

1.1 Motivation

Outdoor IoT deployments have unique requirements. In areas where terrain is non-planar, the target region of the IoT deployment can block inter-device communication and reduce remote sensing coverage. Although the related problem of Wireless Sensor Network (WSN) deployment has been well researched[52], most outdoor WSN deployment approaches often constrain nodes to a region represented by a two-dimensional (2D) plane, which cannot model non-planar terrain. Even if a plane includes obstructions, which is considered in the *Art Gallery Problem*[43], a plane with obstructions is a poor model for non-planar terrain. For example, a mobile node in a plane can move in only two dimensions to avoid the effects of a blocking obstruction. On the other hand, an outdoor IoT node has the further option of moving up a hill to increase its elevation and let it see *over*

an obstruction. So WSN deployment algorithms for 2D are not appropriate for use with deployments over outdoor IoT terrain.

Movement of WSN nodes in three dimensions also has been researched by considering nodes that can fly or can float in 3D regions of air or water[26][44]. Although an IoT node such as a quadcopter (which can fly through the air over terrain) can increase its elevation to see over obstructions, such capabilities require energy expenditure just to remain immobile, and so this dissertation does do not consider such nodes.

Instead, the problems addressed in this dissertation are modeled by restricting IoT nodes to a 2.5D surface, where "2.5D" means a surface whose third dimension is a function of the first two[66]. One can define such a surface by coordinates using a height function of two variables: $(x, y, h(x, y))$. A 2.5D surface can represent nearly any terrain form, except for rare cases where terrain has more than one height at position (x, y), such as a cave or an overhang, which this dissertation doesn't consider.

1.2 Terminology

1.2.1 Non-contact Sensors

This dissertation considers nodes that have *non-contact sensors* that detect events at a distance but require an unobstructed view. Examples of such sensors are ultrasonic microphones, infrared imagers, and cameras. In this dissertation we can think of all such sensors as cameras.

1.2.2 Visibility, Viewshed, Coverage

One says that a target is *visible* if one or more of the network's cameras can see the target with a clear *line-of-sight* (or *CLOS*) view. In the case of non-contact sensors, besides CLOS, the camera's "range" must be considered, as well. For instance, when someone is so far away that they become a single pixel on the camera sensor, it's not possible to identify who they are. So we limit the maximum camera-to-target range to some *sensor radius*. Combining visibility, i.e., CLOS, and sensor radius leads one to the concept of *range-limited visibility*. In this dissertation, all visibility is assumed to be range-limited.

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 visible targets forms a *viewshed*[7]. 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 number of targets of a viewshed or its area is called its *cumulative visibility*[7] or its *coverage*[52]. Accounting for the constraint of maximum camera-target range yields the measure of *network fitness* that I use in this work: *range-limited cumulative visibility*. Through the rest of this dissertation, all cumulative visibility is assumed to be range-limited. The terms cumulative visibility and coverage will be used interchangeably.

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

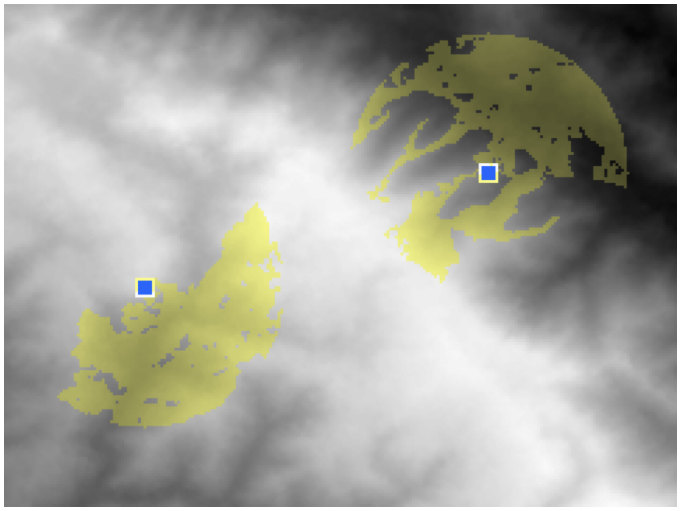


Figure 1.1: Viewshed of a two-node network on 2.5D terrain. Lighter areas represent regions with higher elevation, and darker areas represent regions with lower elevation. Blue squares represent the positions of nodes. Yellow areas indicate the positions of targets that have CLOS (clear line of sight) visibility to at least one of the nodes.

grid-based height map, a straightforward visibility algorithm will treat each grid position as entirely visible or as entirely obstructed[65]. As such, the 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 1.3, the problem of achieving maximum coverage, or maximum 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.

To illustrate the concepts of viewsheds and cumulative visibility, Fig. 1.1 shows a 2.5D 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

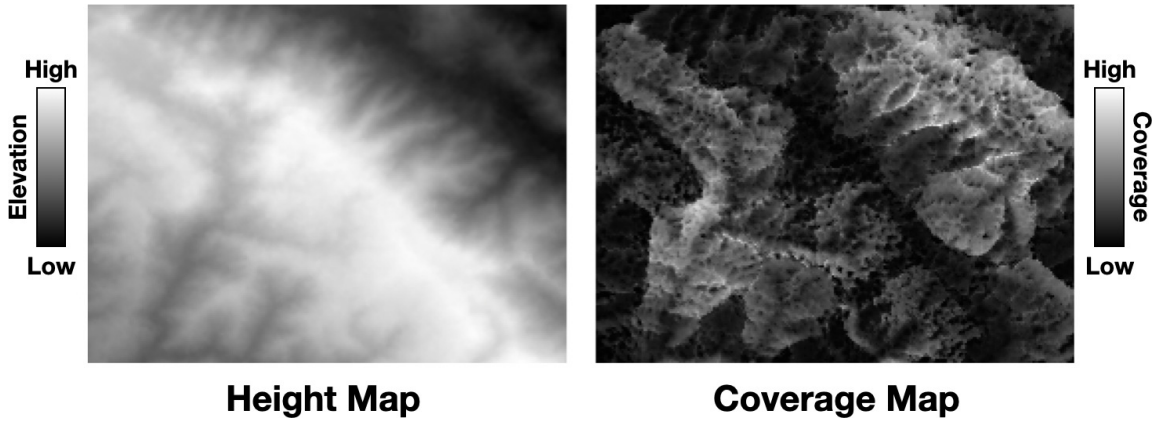


Figure 1.2: On the left is a *Height Map* of terrain. Lighter regions represent higher elevations, such as hills and plateaus, and darker regions represent lower elevations, such as valleys. On the right is a *Coverage Map* showing the range-limited cumulative visibility of the terrain. Lighter regions identify potential node positions that would provide greater coverage (range-limited cumulative visibility).

as dark lines. The two blue squares represent the node positions of a two-node network, and the yellow regions comprise the all of the positions that are visible to at least one of the nodes, defining the viewshed of the network. The total area of the yellow region is the network's cumulative visibility or coverage.

1.2.3 Wonderful Life Utility

Consider Fig. 1.2. On the left is a height map. Now imagine that for each position on the height map one computes the yellow viewshed region for one node at that position. The area of each position's viewshed would be that position's cumulative visibility. Then once the cumulative visibility of every position is computed, all of these values can be plotted on a *coverage map*, as shown on the right side of the figure. One can use such a

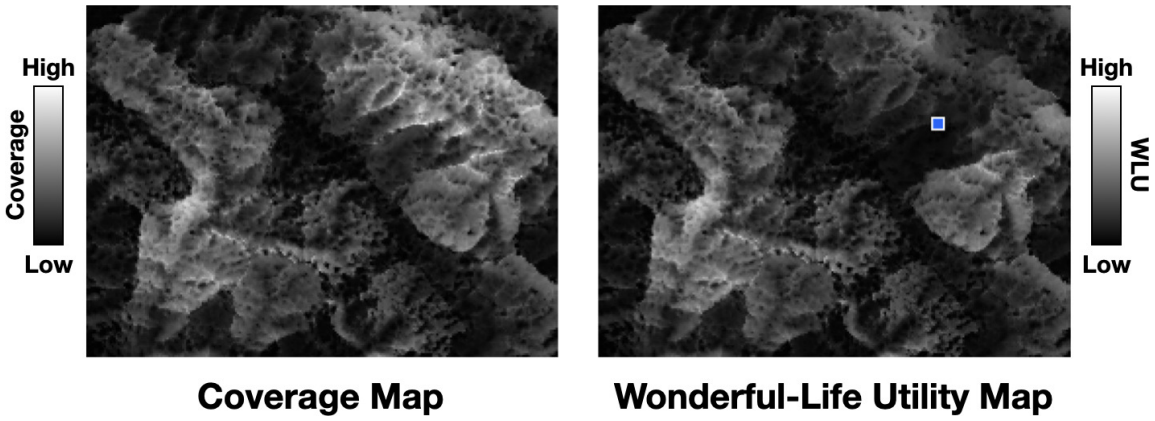


Figure 1.3: On the left is a *Coverage Map*. On the right is a corresponding *Wonderful-Life Utility Map* that results from placing a node at the position of the blue square. Brighter parts of the WLU Map indicate positions where placing a second node would improve network coverage the most.

coverage map to choose the best position for a node. That is, if one places a node at the brightest position on the map, then one will have a deployment with the greatest coverage.

However, to place a second node, one wants to know the *additional* or *incremental* coverage that the second node provides. To this end, I use the *Wonderful Life Utility* (or *WLU*), a concept from potential games[72]. Using WLU, 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 difference is the node's WLU, that is, its 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.)

As an example, the left side of Fig. 1.3 shows the coverage for all positions of

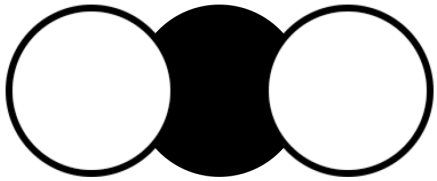


Figure 1.4: This three-node example shows how changes in Wonderful Life Utility (WLU) are the same as changes in global utility. The total area of the figure represents the system's global utility (g). The two open circles represent the coverage of the two outermost nodes, and the central, dark region represents the Wonderful Life Utility of the central node. If the central node were to move, the change in the total area would be the same as the change in the area of the dark region, or $\Delta g = \Delta\text{WLU}$.

the map, and the right side shows the WLU for all positions after a node is placed at the position indicated by the blue square. That is, the right side shows the *additional* coverage that would result from placing a *second* node on the map given a first node in the indicated position. Comparing the maps, one can see that WLU is low near the first node. This is because placing a second node near the first node just provides multi-coverage and does little to improve overall coverage. On the other hand, far from the first node, the WLU map is seen to be the same as the coverage map because placing a second node far from the first avoids multi-coverage and hence will improve overall coverage.

A characteristic of WLU is that 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. As demonstrataed in Fig 1.4, we know that:

$$\Delta g = \Delta\text{WLU}. \tag{1.1}$$

Hence, as long as (1.1) is true, a node can use changes in its WLU to compute the effect

that its movement has on network fitness.

1.2.4 Commsheds

Similar to light waves, microwave signals can pass through the air, and they cannot pass through soil. And so just as visibility leads to the concept of a covered region of terrain called a viewshed, communication constraints lead to an analogous communication concept of a covered region of terrain called a *communication viewshed* or a *commshed*[13]. A commshed is a region of 2.5D terrain within which a node can communicate with whichever node defined the commshed.

While this dissertation uses a CLOS radio-propagation model, using other models, such as those that incorporate Fresnel zones[18], does not affect the overall algorithms which treat commsheds as sets of positions. I address these ideas further in Future Work in Chapter 6.

Just as a maximum sensor range leads to a range-limited viewshed, a maximum communication range leads to a range-limited commshed. In the rest of this dissertation, any mention of a commshed is assumed to be range-limited.

Since commsheds are sets of positions, the commsheds of a network's nodes can be joined using the *union* set operation to construct the commshed of the network, which is the set of positions where a new node could communicate with at least one of the preexisting nodes of the network (Fig. 1.5).

In addition to commshed unions, a key concept in this dissertation is *commshed*

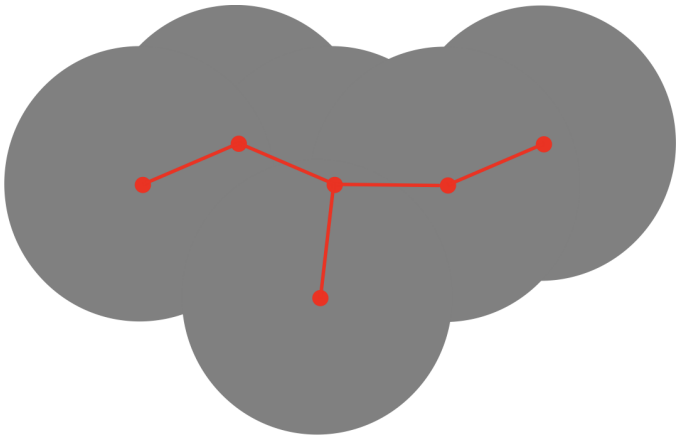


Figure 1.5: Commshed of a network. The gray region indicates positions that are within distance R_C of at least one node of the network. A new node that is placed anywhere in the commshed can communicate with the network. (For clarity, this figure uses a simple 2D disk radio-propagation model, but commsheds can be defined using any radio-propagation model.)

intersections, specifically the intersection of the commsheds that are defined by a partitioned network's partitions. The commsheds of a network's partitions can be joined using the *intersection* set operation yielding the set of positions where placing a node will rejoin the partitions and make the network whole again (Fig 1.6). Commshed intersections will appear again in Chapter 4 on network recovery.

1.3 Problem Formulation

Using the terminology of the prior section, this section describes the three related topics of this dissertation.

The first topic is the camera network *deployment* problem of a fixed number of nodes over outdoor terrain. This problem requires maximizing line-of-site visual coverage.

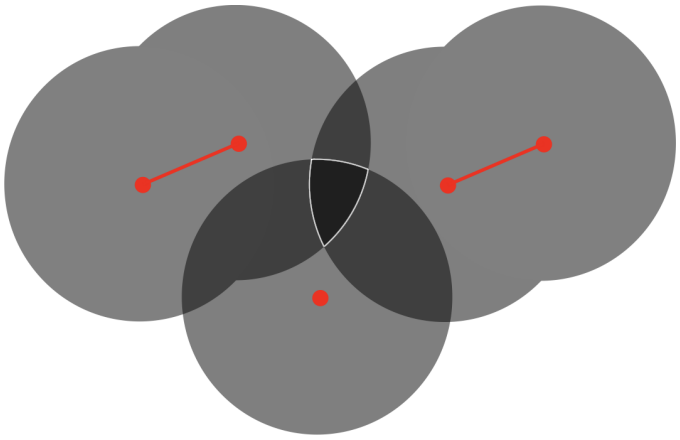
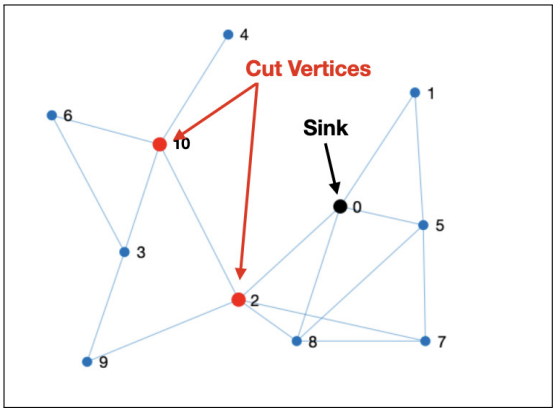


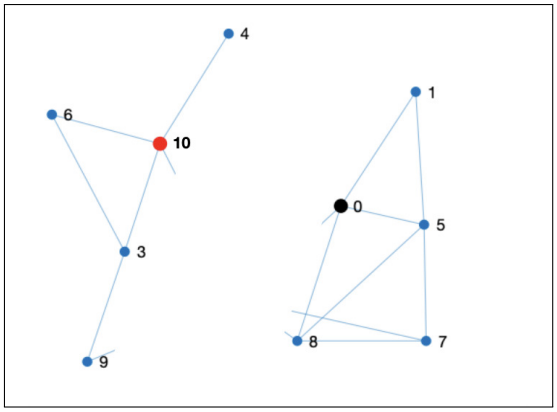
Figure 1.6: Commsheds of a partitioned network. The central, roughly triangular gray region indicates the intersection of all of the partitions' commsheds. A new node that is placed anywhere in the three-way intersection can communicate with all three partitions and will repair the network.

Initially, a homogeneous network of mobile and omnidirectional camera nodes[52] is placed at random over terrain. Each node can determine its position, through GPS or another means, and has access to a digital height map of terrain. Then each node guides itself using the height data to compute its WLU and to improve overall coverage through a one-time deployment. Node-to-node communication is modeled using a disk model and is one-hop. To address this problem, I design distributed deployment algorithms. Work on deployment is described in Chapter 3.

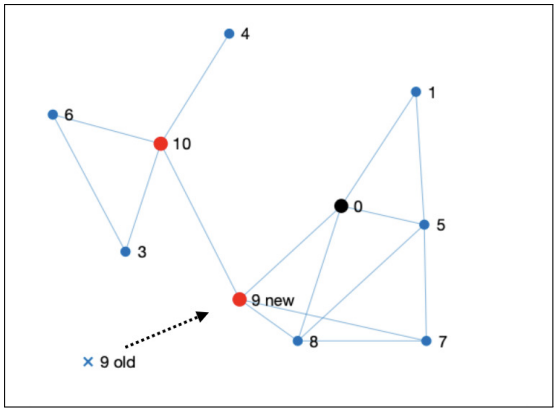
The second topic is the problem of *network recovery* over outdoor terrain. This problem requires mobile nodes to reposition themselves in order to reconnect a network that has been partitioned due the loss of a single node. An additional goal is to mitigate lost coverage or even to improve coverage. For an example of network recovery see Fig. 1.7.



(a) A network with two cut vertices.



(b) Node 2 fails, partitioning the network and isolating nodes 3, 4, 6, 9, and 10 from the sink.



(c) Node 9 repositions to restore communication between the two partitions.

Figure 1.7: Example of network recovery through node mobility.

To address this problem, I design and evaluate algorithms that guide the necessary node movement. Work on network recovery is described in Chapter 4.

The third topic of this dissertation is the creation of the research framework that was used to carry out reproducible experiments on the algorithms mentioned above. Work on the TAFFI research framework is described in Chapter 2.

1.4 Contributions

My contributions include:

- Distributed Deployment Algorithms
  - The distributed Grid Partition deployment algorithm, which has superior coverage-cost performance compared with other distributed optimization algorithms[60].
  - A distributed simulated annealing deployment algorithm[59].
  - The distributed $\text{EMNA}_{\text{global,steps}}$ population-based, Estimation of Distribution (EDA) algorithm that better directs its search by adopting an alternative, more suitable covariance-matrix computation approach[62].
  - A simplified least-squares-based gradient-descent algorithm.
  - The exploration of the effect of benchmarks and algorithm-control parameters on distributed outdoor IoT deployment algorithm performance.
- Family of Recovery Algorithms
  - The TARCI network recovery algorithm, whose performance compares will with an exhaustive algorithm for about a hundredth of the cost[63].
- Experimental Framework for 2.5D Terrain
  - The TAFFI framework with its experimental infrastructure, command-line job management, library of line-of-sight algorithms, library of terrains, and selection

of deployment algorithms[62].

- Terrain Classifiers
  - The use and evaluation of Average Cumulative Visibility as a novel terrain classifier (with Sam Mansfield)[37].
  - The use and evaluation of the Greedy Adding set-cover heuristic as a novel terrain classifier[62].

1.5 Organization of Dissertation

The rest of this dissertation is organized as follows: Chapter 2 describes my TAFFI research framework, Chapter 3 describes my contributions to 2.5D deployment algorithms, and Chapter 4 covers my contributions to 2.5D recovery algorithms. Chapter 5 lists additional accomplishments of mine made at UCSC and Chapter 6 concludes the dissertation and includes a discussion of future work.

Chapter 2

Evaluation Framework

I created TAFFI (Terrain-Aware Framework for IoT)[62] to allow easy and repeatable evaluation of distributed network algorithms over 2.5D terrain. This chapter describes the framework's components and is organized into the following sections: Section 2.1 describes TAFFI's source of benchmark terrain data, the terrain classifiers that I used during benchmark selection, and TAFFI's terrain representation. Section 2.2 contains detailed descriptions of TAFFI's currently supported benchmark algorithms. Section 2.3 contains TAFFI's implementation details.

2.1 Terrain

Evaluating outdoor IoT algorithms requires modeling 2.5D terrain. One needs to choose a terrain representation and create a useful selection of terrain benchmarks. To help ensure effective algorithm evaluation, the benchmarks that are selected should represent

a variety of landmass forms without redundancy. This goal requires the use of terrain classifiers.

2.1.1 Terrain Representation

Practical terrain data is discontinuous, as it is a set of distinct elevation samples. This sort of data is called a Digital Elevation Model or DEM[7]. 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)[46], 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 my 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)[47], 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[17].

Although the TIN terrain representation is used often, high-resolution DEM data is available in *raster grid* format, with individual height data points for each point on a longitude/latitude grid. Converting such data into a TIN representation would require

sub-sampling the raster grid data, with consequential loss of terrain detail. Such loss of detail would add undesired errors to the visibility computations.

One could avoid any loss of detail by using the original raster grid as the TIN grid, but then that grid choice would eliminate any storage benefit that the TIN representation normally affords. Consequently, I have selected a raster grid terrain representation that aligns with my source of terrain benchmark data (discussed next). Another advantage of the raster grid representation for terrain data is that it serves the Wang/Robinson/White dynamic-programming visibility algorithm that I use[65].

2.1.2 Terrain Benchmarks

In my research, I have used two sources of terrain data for benchmarks: synthetic terrains and Shuttle Radar Topography Mission (SRTM) terrains. Although I abandoned synthetic terrains early in the research, I will briefly describe them in order to support that decision. Synthetic terrains are mathematically defined, such as a flat plane, a random surface, and sinusoids[59]. An example of a terrain that is described by sinusoids is below. This height equation is intended to be used over the terrain tile $\{0 \leq x \leq 239, 0 \leq y \leq 179\}$:

$$h_{\text{wavy}_2}(x, y) \quad = \quad 1000 \left[\sin^2(\frac{\pi}{180}x) + \sin^2(\frac{\pi}{120}y)\right]$$

Although synthetic terrains are easy to define, early on I reluctantly acknowledged that these arbitrary terrain forms have no basis in evaluating algorithms for real-world deployments, when actual, real-world terrain data is available. Hence, benchmarks used in this dissertation

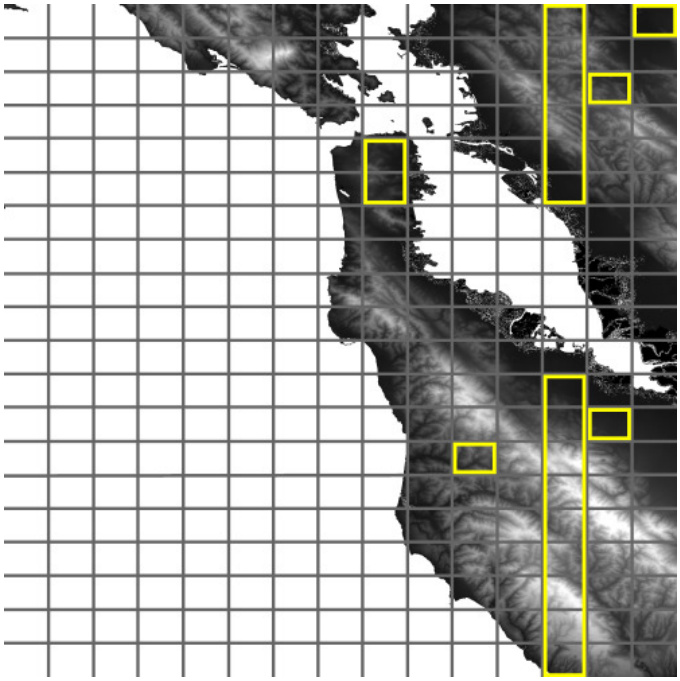


Figure 2.1: Selected subtiles of SRTM tile N37W123.

are sourced from real-world terrain data.

I used data from NASA's 2005 Shuttle Radar Topography Mission (SRTM)[16]. This mission collected elevation data for much of the landmass of the Earth and provides this information in datasets as digital elevation models (DEMs). The DEMs provide the height for positions on the Earth using latitude-longitude coordinates that have a precision of 1 arc second (about 30 meters in the latitude direction).

The SRTM datasets are organized into "tiles" of 3600×3600 arcsecs (1 degree of longitude by 1 degree of latitude). I selected tile N37W123 (Fig. 2.1) because it includes regions that represent a wide variety of landform shapes. Since the full N37W123 tile is impractically large, I divided it into 300 subtiles, each 240×180 arcsec, and consider each

subtile separately for inclusion as a terrain benchmark. The figure indicates which subtiles are used for benchmarks; the next section justifies the selection.

2.1.3 Terrain Classification

To help ensure that my benchmark set includes a broad but practical selection of landmass forms, I evaluated each subtile using two metrics. The first metric is the coverage provided by the centralized greedy set-cover heuristic *Greedy Adding*[8]. I began to use this heuristic as a novel terrain-classification metric in [60]. The second metric, which I used earlier in my joint work with Mansfield, is a subtile's *Average Cumulative Visibility*[37] (ACV). To compute the ACV of a subtile, one averages the cumulative visibility of all of its 43 200 positions. As explained below, I have observed that the Greedy Adding terrain metric-classifier predicts the performance of the best algorithms that I have tested, while the ACV classifier better predicts the performance of the remaining algorithms. Section 2.4.1 discusses this observation further.

After classifying all of the subtiles using Greedy Adding, I selected 21 benchmark subtiles that span a broad range of classifier values, as confirmed by the horizontal distribution of datapoints in my tests of a variety of deployment algorithms from Chapter 3 (see Fig. 3.6 for results with 10 nodes). Then I confirmed that my selected benchmark subtiles also span a broad range of ACV values, as confirmed by the horizontal distribution of datapoints in the results of 3.7 for 10 nodes. (Distributions for 20 and 30 nodes are similar. See Appendix A.)

2.2 Terrain-Aware Framework For IoT

This section discusses the use of the TAFFI/Java framework for evaluating distributed deployment algorithms. Then it compares TAFFI/Java with the use of the TAFFI/C++ framework for evaluating centralized deployment and recovery algorithms. Section 2.3 presents the implementation details.

2.2.1 TAFFI/Java Operation

Controlled by the TAFFI/Java framework, each simulated node runs an Outer Loop in which it alternates between *communicating* with its neighbors, i.e., the nodes within communication range, and *moving*. The communication phase consists of broadcasting a node's current physical position along with the most recent position reports that have been received from other nodes. The movement phase consists of using the optimization algorithm of choice to compute a new position and then moving to it. If a node is unable to find an improved position over several iterations, the Outer Loop will terminate. To avoid getting into an infinite execution loop, the Outer Loop will terminate after a maximum number of iterations. The Outer Loop of TAFFI/Java is detailed in Algorithm 2.1 and is described below.

Each node of the network executes this Outer Loop independently. In **Steps 1–2**, the node is given an initial position and an initial *exploration radius* equal to R_E. The exploration radius is used to prevent a node from traveling too far before receiving position

Algorithm 2.1 Outer Loop of TAFFI/Java that is executed independently by each node in distributed algorithms.

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

updates from its neighbors. This feature is motivated by each node's use of WLU as a fitness function, i.e., I want to avoid the situation where $\Delta g \neq \Delta$WLU due to a change in a neighboring node's coverage. (See Section 2.2.1 for a complete explanation of the exploration radius and exploration region.)

Steps 3–10 are the outer loop which runs until the node's fitness ceases improving for C cycles or until a maximum number of loops L is executed. The maximum loop count helps ensure that the algorithm eventually will terminate. **Step 4** implements the movement phase by calling the optimization algorithm of choice which determines the next position for the node and moves the node to its new position. **Step 5** decreases the exploration radius to sequentially focus the search on more promising regions. For my current experiments, the exploration radius decreases linearly each iteration by $\frac{R_E}{13}$. However, Simulated Annealing, which focuses the search using its temperature parameter T, leaves its exploration radius at R_E. **Steps 6–8** terminate the loop if there is inadequate fitness improvement, that is, if there is no improvement in the best fitness seen for C consecutive cycles. Each node

waits C cycles before terminating because the inability to find a better position may be resolved when one of the node's neighbors moves and allows a subsequent improvement in the node's WLU. In my experiments, I found that $C = 4$ is adequate. Once all nodes terminate, the simulation ends. Assuming that the simulation continues, **Step 9** implements the communication phase during which the node broadcasts its new position to all of 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. Since nodes are not synchronized, any node may receive position updates from its neighbors at any time, although in my simulations, nodes store such updates until they are first used at the beginning of Step 4.

2.2.2 TAFFI/C++ Operation

TAFFI/C++ is better suited for evaluating centralized algorithms since, unlike TAFFI/Java which runs Java models under a simulator and scheduler, TAFFI/C++ runs native compiled C++ programs. That difference makes passing control parameters on the command-line of a TARCI/C++ program more straightforward than passing control parameters to a TARCI/Java simulation through a Cooja configuration file.

But aside from those differences, both TAFFI frameworks are similar: both frameworks schedule jobs on the command line, both frameworks provide a library with visibility algorithms, and both frameworks provide a library of benchmark terrains. Although it is possible to force TARCI/Java to run a centralized algorithm (essentially running the

entire centralized simulation during the initialization of node 1), I found that writing a C++ command-line program was more straightforward.

2.2.3 Benchmark Algorithms

To help with the evaluation of deployment algorithms, TAFFI/Java includes several distributed benchmark algorithms for comparison. Details of the distributed covered in Chapter 3, but I list them here: Grid Partition, $\text{EMNA}_{\text{global,steps}}$, Simulated Annealing, Gradient Descent, and Pattern Search. In addition, several centralized benchmark algorithms are included: Random Placement, Regular Grid, and Greedy Adding.

2.3 Implementation Details

2.3.1 TAFFI/Java Implementation

While a Cooja user can describe a node's operation using emulated microcontroller binaries or using compiled C code with a JNI interface[20], instead I chose to write "application-level" nodes in the simulator's native Java. Such application-level nodes are more straightforward to use and avoid the memory limitations of Cooja's MSP430 microcontroller emulator.

My `AppMoteMobile` Java subclass extends Cooja's `AbstractApplicationMote` class, but it also adds mobility. To add mobility, my `AppMoteMobile` class creates two interlocked `MoteTimeEvent` objects: `moveEvent` and `transmitEvent`. These private objects

are created by the `AppMoteMobile` constructor. Each call to the node's `execute()` method schedules a moveEvent. When the moveEvent fires, it computes the node's new position, moves the node, and then schedules a transmitEvent. A transmitEvent transmits a packet. After each transmission, the `sentPacket()` callback calls the node's `execute()` method, which then schedules another moveEvent, completing the cycle. The moveEvent performs Steps 4–8 of Algorithm 2.1; the transmitEvent performs step 9.

I use several software tools to manage the tens of thousands of simulation runs that evaluated the algorithms for this dissertation. First, I run simulations under GNU Make[53], which helps me avoid duplicate runs. Since I use multiple compute servers, each server has its own Make file that defines only that server's Make targets and then includes a common Make file, which contains the dependencies and actions necessary to run all simulations. Due to the inherent operation of Make, only the simulations that are identified in each server's Make file are run.

Second, to pass control parameters to simulations, I wrote a command-line utility, `params`, that creates Cooja simulation configuration (`.csc`) files from templates. The `params` utility accepts the name of a `.csc` file template (which will be the basis for the generated configuration file), an index into a compiled-in list of random initial node locations (for simulation reproducibility), and an optional list of name/value pairs for controlling simulation parameters. The name/value pairs can be placed directly on the command line, or they can be included within one or more referenced text files. This flexibility helps me efficiently run a large set of simulation runs for my experiments.

Third, I use a parameterized Bash[48] script to prepare each simulation's `.csc` file, run the simulation, and compress the resulting log file using Gzip[19].

Fourth, I use MATLAB[54] for data analysis and for plot generation. MATLAB can directly generate the Encapsulated PostScript files that are used by LaTeX[29] to generate the plots in manuscripts (such as this one). I also use MATLAB to generate data table text files, which I include in LaTeX documents manually.

Fifth, prior to running MATLAB, I use GNU Awk[49] scripts to merge data from the tens of thousands of Cooja simulation log files into a handful of automatically generated MATLAB script (`.m`) files, one script file for each plot, or for each set of related plots, or for each data table text file. GNU Make manages the GNU Awk and MATLAB executions and helps ensure that tables and plots are updated when necessary.

2.3.2 TAFFI/C++ Implementation

Although using TAFFI/C++ means that a centralized C++ program is written instead of a distributed Java program, and that control parameters are passed directly on the C++ program's command line rather than indirectly through a Cooja configuration file using `params`, the rest of job management and data analysis with TAFFI/C++ is the same as with TAFFI/Java.

2.4 Discussion

2.4.1 Evaluation of Terrain Classifiers

In Section 2.1.3, I introduced two terrain classifiers, namely Greedy Adding and ACV. I used these classifiers to choose the terrain benchmarks for my experiments, desiring that the selected benchmarks present a wide range of difficulty. Here I evaluate the effectiveness of these classifiers as predictors of difficulty.

I used MATLAB's `fitlm()` function to fit an algorithm's cumulative visibility results (see Chapter 3) for each benchmark (averaged over 10 random seeds) to each benchmark's corresponding classifier value. Then for each regression model generated, I retrieved the r^2 coefficient of determination from the model's `Rsquared.Ordinary` property. The r^2 value indicates the approximate portion of variability in an algorithm's results that is predicted by the terrain classifier. An r^2 value of 1.00 or -1.00 means that the classifier perfectly predicted variability in the algorithms result (as a linear function); an r^2 value of 0.00 means that the classifier failed to predict variability in the algorithm's result. Table 2.1 reports the r^2 Coefficient of Determination for algorithm results relative to both classifiers for 10, 20, and 30 nodes.

Considering first the Distributed Grid Partition algorithm, the table shows relatively high r^2 values for the Greedy Adding terrain classifier, meaning that using this classifier will tend to choose benchmarks with a wide range of difficulty for Grid Partition. (I note that since Grid Partition performs nearly as well as Greedy Adding in my tests,

Table 2.1: Terrain Classifications as Predictors of Algorithm Performance.

Algorithm	r^2 **using Greedy Adding**			r^2 **using ACV**		
	10 Nodes	**20 Nodes**	**30 Nodes**	**10 Nodes**	**20 Nodes**	**30 Nodes**
Dist. Grid Partition	**0.92**	**0.89**	**0.87**	0.20	0.26	0.50
Dist. Simulated Annealing	**0.83**	0.65	0.12	0.30	0.05	0.02
Dist. Pattern Search	0.61	0.78	**0.92**	0.64	0.77	0.69
Dist. $\text{EMNA}_{\text{global,steps}}$	0.31	0.74	**0.88**	**0.85**	**0.84**	**0.88**
Dist. Gradient Descent	0.09	0.50	0.73	**0.94**	**0.93**	**0.96**
Regular Grid	0.14	0.41	0.57	0.69	**0.85**	**0.89**
Random Placement	0.07	0.37	0.62	**0.99**	**0.99**	**0.98**

seeing a uniformly high r^2 is not a surprise.) The table also shows that the benchmarks selected using Greedy Adding present a wide range of difficulty for Simulated Annealing when it is run with 10 and 20 nodes. Greedy Adding can predict how terrain will affect Pattern Search and how terrain will affect $\text{EMNA}_{\text{global,steps}}$ with 20 and 30 nodes. These results suggest that although there are features of algorithms that are influenced by terrain, also there are algorithm features that are affected by the number of nodes deployed. This is an area for future work.

Examining the ACV (Average Cumulative Visibility) half of the table, I see that using the ACV classifier to choose benchmarks will result in a wide range of difficulty for $\text{EMNA}_{\text{global,steps}}$, Gradient Descent, Regular Grid, and Random Placement. The $\text{EMNA}_{\text{global,steps}}$ algorithm, which performs well with more nodes, is well served by ACV, but since the remaining three of these algorithms are among the worst that I tested, evaluating their performance by terrain using ACV is merely interesting.

The correlation of Random Placement with ACV is unsurprising since the coverage of several randomly placed nodes would tend to sum to a multiple of average coverage, and ACV *is* an average. The correlation of Gradient Descent with ACV also should not be surprising since I know that due a noisy fitness function, Gradient Descent ends its searches early after encountering local optima, meaning that Gradient Descent finds solutions that are only slightly better than its initial random placements. $\text{EMNA}_{\text{global,steps}}$ is strongly correlated with ACV, but also it is strongly correlated with Greedy Adding in simulations with 30 nodes. It is curious that a single algorithm shows correlation with both metrics under different circumstances. Since both terrain classifiers predict the performance of some algorithms tested, I used both.

2.4.2 Reproducibility

Reprodicibility of results is important for quality research, but it is difficult to achieve. To help working toward this end, nearly all of the experiments of this dissertation were performed non-interactively, through the command-line based TAFFI framework. In addition, nearly all of the plots (Embedded Postscript files for LaTeXand PNG image files for PowerPoint) were created from MATLAB scripts. This unwavering focus on using command-line tools through scripts and Make files helps ensure that results can be repeated merely by moving the original results files and rerunning the necessary Make files. (This sounds easy, and it is, but in many cases regenerating the results will take several days of CPU time, due to the large number of experiments conducted. Be aware!)

2.5 Contributions

Regarding the framework: *TAFFI* (Terrain-Aware Framework for IoT) inherently considers the topography of a target deployment region by having the modeled camera nodes select their positions in order to maximize an outdoor IoT network's overall visual coverage. TAFFI also incorporates an experimental infrastructure that eases the development and analysis of distributed IoT algorithms that aim at the deployment and repair of outdoor IoT camera networks. When one uses TAFFI, one creates models in either Java or C++. TAFFI/Java is built upon the Cooja-Contiki IoT network simulator/emulator[45] and is best suited for modeling distributed algorithms and their networking protocols. TAFFI/C++ is built upon GCC and is best for modeling centralized algorithms. Both frameworks integrate a line-of-sight radio-communications model and provide a line-of-sight camera/target visibility library for IoT device models. And both frameworks add flexible command-line job management to simplify the creation of experiments and to help ensure repeatability, although I note that the command-line job management is *especially* useful with TAFFI/Java because it lets users avoid Cooja's GUI-based means for individually defining multiple experiments.

To help standardize experimental setups, TAFFI includes a benchmark suite of actual terrains. To ensure that the selected terrains span a wide range of landmass forms, I chose them based on two novel terrain-classification metrics: Greedy Adding and Average Cumulative Visibility. I show that, together, these two metrics suggest best-case deployment

algorithm performance, and as such, they helped me ensure that the benchmark suite includes terrains that range from "flatter" to "rougher."

To facilitate evaluating the performance of distributed IoT deployment techniques TAFFI includes several distributed deployment algorithms.

Chapter 3

Outdoor IoT Deployment Algorithms

In this chapter I describe my outdoor IoT deployment algorithms. This chapter is organized as follows: Section 3.1 discusses related work, and Section 3.2 presents a taxonomy of optimization algorithm that I considered. Then Sections 3.3 and 3.4 present my iterative-improvement algorithms Distributed Simulated Annealing and distributed Grid Partition. Next Section 3.5 presents my estimation of distribution algorithm $\text{EMNA}_{\text{global,steps}}$. After briefly discussing my evaluation methodology in Section 3.6, I present my results in Section 3.7 and conclude with discussion in Section 3.8.

3.1 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* the floorspace of an art gallery is "guarded" by locating cameras or guards[43]. 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 mine. A 3D version of the Art Gallery problem considers the visibility of a polyhedron's *interior volume*, but once again I am 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 provide 100% coverage of a 2D area[56]. Although LSCP can be formulated as a zero-one linear-programming problem (with two variables for each node's two coordinates and one equation for each target whose coverage is desired[55]), since LSCP assumes an unbounded number of nodes, I 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. This problem can be addressed by a greedy set cover heuristic that stops after placing all available nodes, as exemplified by the Greedy Adding algorithm analyzed by Church[8]. As LSCP is similar to my problem, I include centralized Greedy Adding in my evaluation as a likely upper bound.

Coverage in the area of wireless sensor networks (WSNs) has received considerable attention from the research community. In particular, the work in[76] provides a thorough survey of the state-of-the-art in using centralized algorithms to provide coverage over terrain. Many algorithms consider network connectivity constraints, which is part of my future

work.

Finally, many researchers have looked at the centralized camera network deployment problem over 2.5D terrain. To cite two examples, Lv et al.[35] use Simulated Annealing with a centralized algorithm, and Akbarzadeh et al.[2] compare four different centralized placement algorithms for camera nodes: regular grid, simulated annealing, the L-BFGS method, and CMA-ES. However, this dissertation considers distributed deployment algorithms, rather than the centralized algorithms of the related work.

3.2 Taxonomy of Algorithms

When one is presented with a complex optimization problem, often it is not trivial to find an algorithm that guarantees an optimal solution in reasonable time. Instead one can use heuristics to find an adequate solution. Heuristics can be classified in different ways, such as whether they are inspired by nature or whether they make use of search history. Others have found it useful to classify heuristics into those that iteratively improve a single solution and those that use an evolving population of multiple solutions[5].

3.2.1 Iterative Improvement Algorithms

An iterative-improvement algorithm repeatedly adjusts a single candidate solution during its search for a global optimum. The algorithm's decisions are based on the fitness of the current solution and any prior solutions seen. I want to evaluate how a selection of

iterative-improvement algorithms respond to the WLU fitness function, which means that I should choose algorithms that do well in a distributed context with only a two-variable fitness function. (Below we will see that two-variable fitness functions are inappropriate for some well-known heuristics.) In addition, the algorithms must use my chosen terrain representation (see Section 2.1.1). Later in this chapter, I consider these iterative heuristics: the stochastic optimization algorithm Simulated Annealing (Section 3.3) and the greedy algorithm Grid Partition (Section 3.4).

3.2.2 Population-Based Algorithms

Population-based algorithms use a "population" of candidate solutions in a search for a global optimum. Such an algorithm's decisions are based on the fitness of all of the candidate solutions of the population. Some examples of population-based heuristics that I considered are Estimation of Distribution (EDA), Ant Colony Optimization, and Swarm Intelligence.

An EDA is a form of Evolutionary Algorithm in which the various tuning parameters (controlling crossover and mutation, for instance) are eliminated and replaced with multivariate probability distributions[30]. CMA-ES is a well-known EDA heuristic that adds recombination weights, step-size control, and learning rates[21]. Although CMA-ES performs well with higher-dimension fitness functions, its author reports that it will be outperformed when a fitness function has only two dimensions[22], as is the case with my distributed algorithm.

I considered using an Ant Colony Optimization (ACO) heuristic. Such a heuristic can be used for local search algorithms, but like Estimate of Distribution Algorithms (EDAs), it has shown success with higher-dimension fitness functions. For example, an ACO algorithm for a 75-city Traveling Salesman Problem is considered a "small" instance[14]. For this reason, although ACO might be appropriate for a centralized algorithm that finds positions for 75 nodes, I do not consider it as a potential solution to my distributed optimization problem that has a two-variable fitness function.

I also considered the population-based Swarm Intelligence heuristic. In one well-known Swarm Intelligence technique, Particle Swarm Optimization (PSO), numerous potential solutions, or "particles," move in the solution space while randomly adjusting their velocities based on their and other particles' best known solutions[33]. Given a particle i at position $\mathbf{x}_i$ with velocity $\mathbf{v}_i$, the stochastic weighted average (3.1) updates the velocity of the particle making it tend toward its personal best position $\mathbf{p}_i$ as well as toward the best position $\mathbf{p}_g$ of the particle's neighbors or "group." φ_1 and φ_2 are random vector weights, and the operator $\otimes$ is component-wise multiplication.

$$\mathbf{v}_i^{\text{NEW}} \leftarrow w\mathbf{v}_i + \varphi_1 \otimes (\mathbf{p}_i - \mathbf{x}_i) + \varphi_2 \otimes (\mathbf{p}_g - \mathbf{x}_i) \tag{3.1}$$

$$\mathbf{x}_i^{\text{NEW}} \leftarrow \mathbf{x}_i + \mathbf{v}_i^{\text{NEW}} \tag{3.2}$$

Particles using velocity-update equation (3.1) or its variants tend to "swarm" around a common solution. Although PSO is an intriguing technique, practical applications cited in [33] have more than just two dimensions.

Finding no existing population-based algorithms that are reported to perform well on two-variable optimization problems, I nonetheless remain intrigued. For the purpose of my evaluation, I developed an EDA algorithm that is inspired by one of CMA-ES's simpler predecessors. My algorithm, $\text{EMNA}_{\text{global,steps}}$ is described later in this chapter (Section 3.5).

3.3 Simulated Annealing

Simulated Annealing is a general and well-known 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[34][38] and even the physical design of VLSI circuits[69].

A Simulated Annealing algorithm generates a sequence of incremental candidate solutions. At each point in the sequence, the algorithm determines whether the candidate solution should be accepted or rejected. The criteria for acceptance depends on the change in the system's fitness, comparing a candidate solution to the solution that was most recently accepted. Given a prior accepted solution $\mathbf{p}$ and a randomly generated candidate solution $\mathbf{q}$, the probability of accepting $\mathbf{q}$ is computed from the potential change in fitness.

$$\Delta = \text{WLU}(\mathbf{q}) - \text{WLU}(\mathbf{p})$$

yielding

$$P(\text{accept } \mathbf{q}) = \begin{cases} 1 & \text{if } \Delta \geq 0 \\ e^{\Delta/T} & \text{if } \Delta < 0 \end{cases} \tag{3.3}$$

A difference between Simulated Annealing and a greedy algorithm is that there is a chance that a Simulated Annealing algorithm will move away from a locally optimal position. The control parameter T in (3.3) is analogous to a system's slowly decreasing temperature during annealing, and therefore, at lower temperatures, the probability of accepting a solution that does not improve fitness decreases[1]. While $T > 0$, an inferior candidate solution may be accepted as a "stepping stone" to a superior solution. The rate at which T decreases greatly affects the performance of the algorithm. I discuss this "annealing schedule" in more detail below.

In the classic Simulated Annealing algorithm implementation, there are two nested loops. The outer loop manages the temperature and the number of iterations, and the inner loop performs a search at a given temperature[51, p. 54]. In my implementation, TAFFI/Java performs the control portion of the outer loop and limits the total number of iterations. Then during each iteration of the outer loop, TAFFI/Java calls Algorithm 3.1, which performs the outer loop's remaining tasks and the tasks of the inner loop.

My implementation is shown in Algorithm 3.1. The algorithm uses as parameters starting position $\mathbf{p}_E$, exploration radius R_E, number of iterations M, temperature T, and

[1]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"[69, p. 5].

Algorithm 3.1 Distributed Simulated Annealing. Inputs: starting position $\mathbf{p}_E$, exploration radius R_E, number of iterations M, and temperature T. Control parameter and its evaluated value: $\beta = 0.9$.

1: $\mathbf{p} \leftarrow \mathbf{p}_E$
2: **for** $i = 1$ **to** M **do**
3: Create a random potential move $\mathbf{q}$ for this node within R_E of $\mathbf{p}$.
4: $\Delta \leftarrow \text{WLU}(\mathbf{q}) - \text{WLU}(\mathbf{p})$.
5: **if** $\Delta > 0$ **then**
6: $\mathbf{p} \leftarrow \mathbf{q}$
7: **else**
8: $\mathbf{p} \leftarrow \mathbf{q}$ with probability $e^{\Delta/T}$
9: **end if**
10: **end for**
11: $M \leftarrow \beta M$ $\{0 < \beta < 1\}$
12: Reduce T {see text}
13: Return M, T, and the best $\mathbf{p}$ seen

control parameter β, which is used to reduce the number of candidate solutions evaluated. I use the value of β that is commonly found in Simulated Annealing references.

In Algorithm 3.1, the **for** loop of **Steps 2–10** (called the Metropolis loop) generates and evaluates a succession of M candidate solutions at temperature T. **Step 11** reduces M, helping to control execution time. **Step 12** reduces T, an important part of a Simulated Annealing algorithm, as discussed below. After M iterations, **Step 13** returns to TAFFI/Java the best position found, but it also returns updated values of parameters M and T that will be passed back during the next call from TAFFI/Java.

Step 12 determines the algorithm's annealing schedule. Early implementations of Simulated Annealing started with a large value of T that forces acceptance of all candidate solutions[51, p. 54]. Their loops periodically multiplied T by a constant between zero and one, reducing T monotonically. 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 optimal value, degenerating to a greedy algorithm. But if one changes T too slowly, execution time of the algorithm may extend unnecessarily.

To avoid both quenching and unnecessarily long run times, adaptive cooling schedules have been developed. For my Distributed Simulated Annealing algorithm, I adopt an efficient yet general cooling schedule[51, p. 72]. This schedule attempts to control T such that consequential increases in the fitness function are limited to the value of the fitness function's standard deviation. (See [51, p. 72] for the inspiration behind this choice.)

Another important aspect of a Simulated Annealing algorithm is the generation of candidate solutions. First, candidate solutions should be incremental changes to the current solution rather than large jumps. Second, the mechanism that generates candidate solutions should allow the possibility of returning to any prior solution. For my algorithm, a candidate position is selected from a uniform circular distribution of radius R_E that is centered on the current position, and so any move potentially can be reversed.

The Simulated Annealing algorithm is evaluated, along with other deployment algorithms, in Section 3.6.

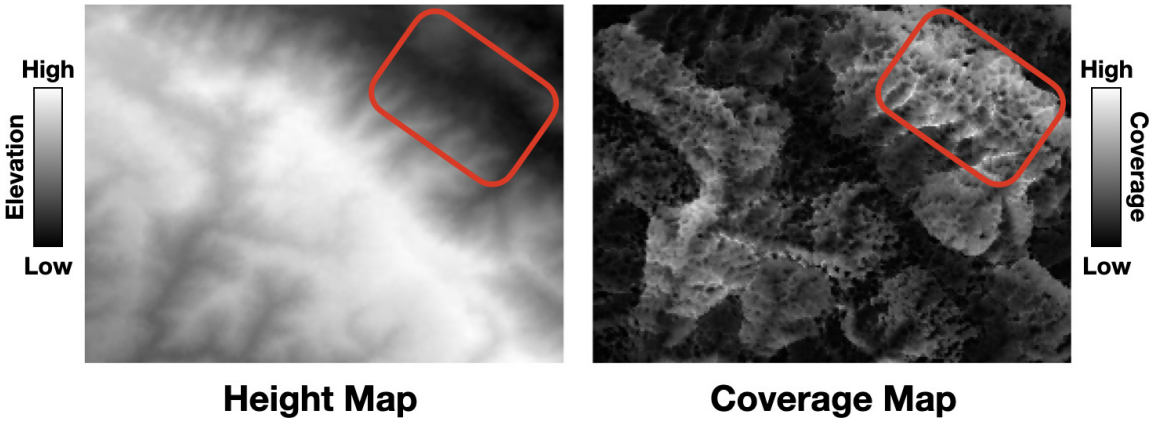


Figure 3.1: Placing a node at a low elevation in a valley can lead to good coverage.

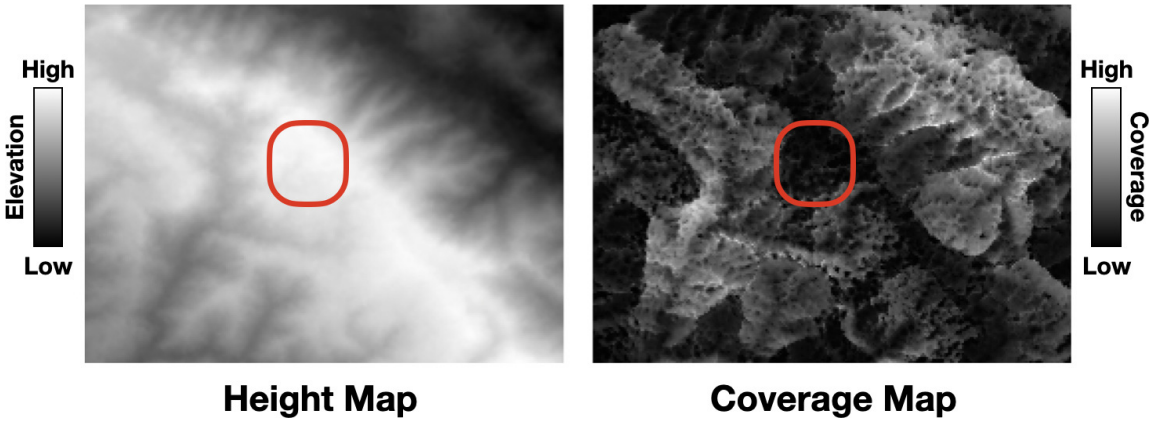


Figure 3.2: Placing a node at a high elevation on a plateau can lead to poor coverage.

3.4 Grid Partition

My distributed Grid Partition algorithm[60][61] is inspired by landform classification[36] and landform scale. Since one might ask the question, "Why not just position nodes on all of the mountaintops?", below I show that globally high positions do not necessarily provide good coverage, but locally high positions may (on landforms with a

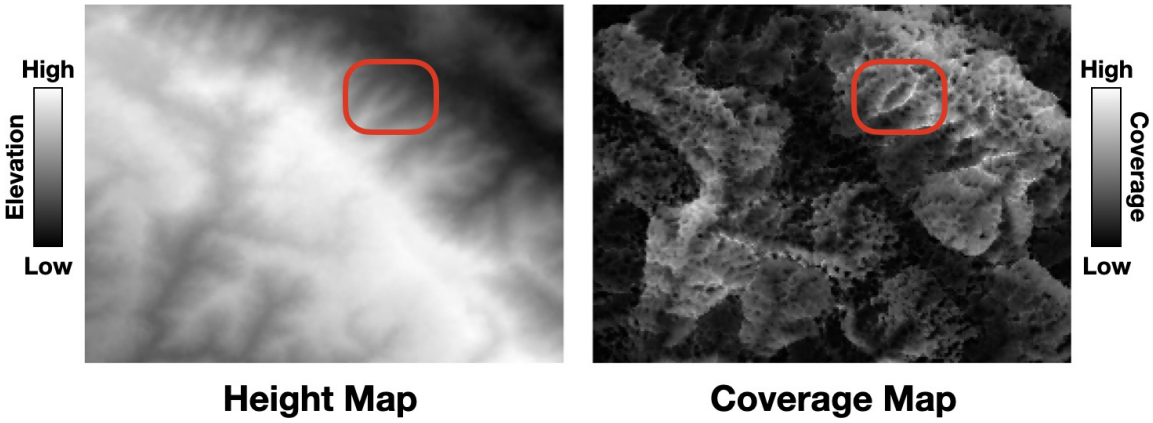


Figure 3.3: Nodes must consider local elevation in order to avoid view-blocking depressions.

smaller scale).

First consider Fig. 3.1. The dark region of the height map on the left indicates the low elevation of a large valley. The corresponding region on the coverage map on the right side is bright and indicates positions with superior coverage. So the concave-up shape of a valley causes many positions on a valley's wall to see a large portion of the opposite wall, improving coverage, even when elevation is relatively low.

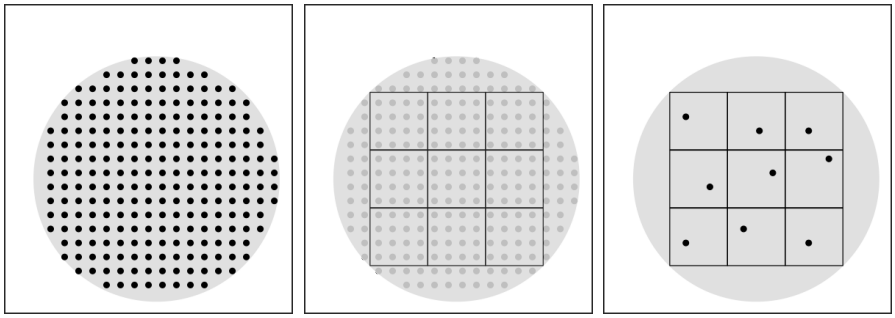
Next consider Fig. 3.2. The bright, central region of the coverage map on the left indicates the region of greatest elevation. Comparing to the coverage map on the right, the corresponding region is dark, which indicates poor coverage. This correspondence shows that higher node elevation does not necessarily lead to greater visibility. There are two reasons for this: (1) A sensor that is located away from the edge a high plateau will be unable to see regions below the plateau without first moving to the edge. (2) 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* distant targets. So overall height does not guarantee coverage.

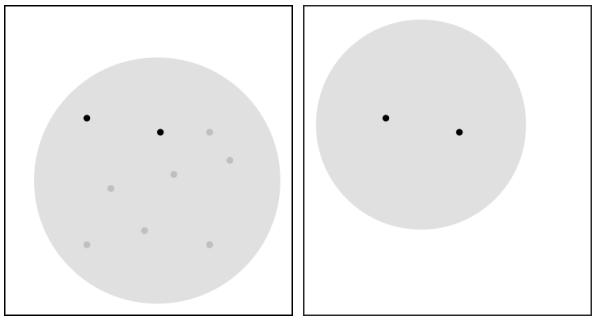
Finally, let's consider the effect of landform scale. The height map on the left of Fig. 3.3 shows that the valley wall has small ridges and depressions. Comparing to the corresponding region of the coverage map on the right, the small ridges have great coverage and the depression between the ridges has poor coverage. Consequently, we see that the scale of the landform is important. Choosing the highest elevations over too large a scale can hurt coverage, while choosing the highest elevations over a smaller scale can help it.

Following the above observations, the Grid Partition algorithm attempts to find positions on landforms that will have large viewsheds, such as valleys, while also maximizing coverage using terrain elevation data to position camera nodes on small rises, ridges, or hills. The key concept of Grid Partition is that terrain elevation data can identify locally superior node positions, making it possible to limit the total number of coverage computations needed.

Using Fig. 3.4 I summarize the Grid Partition algorithm. Grid Partition considers only candidate positions that are in an exploration region that is defined by a circle of radius r centered on the node's current position (see Fig 3.4a). The algorithm partitions the exploration region into n_0 squares (Fig 3.4b) and consults terrain height data to find the highest position within each square (Fig 3.4c). (Choosing the highest local position helps the node avoid local visual obstructions.) Then the algorithm computes the WLU at each of the identified positions and identifies a subset of candidate positions with the



(a) Exploration region (b) Grid partition. (c) Highest positions.



(d) Best positions. (e) Move and shrink.

Figure 3.4: Illustration of the Grid Partition algorithm (see text).

best WLU (Fig 3.4d). The final step of an iteration (Fig. 3.4e) is for the node to choose a new candidate position by computing the centroid of the best candidate positions that were found in the previous step, and also to decrease r. The process is repeated by looping back to Fig. 3.4a, but with the updated candidate position and exploration radius.

Algorithm 3.2 contains the details of this greedy algorithm, which I walk through here. At invocation, the algorithm accepts an initial position $\mathbf{p}_E$ and exploration radius r_E. Parameter A controls the rate at which the exploration region shrinks. Parameter F is the fraction of the candidate positions that will guide repositioning of the exploration

Algorithm 3.2 Grid Partition. Inputs: starting position $\mathbf{p}_E$ and exploration radius r_E. Control parameters and their evaluated values: $A = 0.9$, $F = 0.25$, $M = 10$, and $n_0 = \{5, 10, 21\}$.

1: $\mathbf{C}_E \leftarrow$ circle centered at $\mathbf{p}_E$ with radius r_E
2: $\mathbf{C} \leftarrow \mathbf{C}_E$
3: $\mathbf{p} \leftarrow \mathbf{p}_E$
4: $r \leftarrow r_E$
5: $\mathbf{p}_{\text{best}} \leftarrow \mathbf{p}$
6: **for** M times **do**
7: **loop**
8: Using a square grid, partition $\mathbf{C}_E \cap \mathbf{C}$ into n regions $R_1 \dots R_n$, where $n \approx n_0$. {see text}
9: $\mathbf{p}_i \leftarrow$ highest position within R_i **for** i in $1 \dots n$.
10: Compute the fitness of each position $\mathbf{p}_1 \dots \mathbf{p}_n$.
11: Sort positions $\mathbf{p}_1 \dots \mathbf{p}_n$ by decreasing fitness.
12: $\mathbf{p}_{\text{best}} \leftarrow \mathbf{p}_1$ **if** $\mathbf{p}_1$ is better than $\mathbf{p}_{\text{best}}$
13: **if** the grid in step 8 is smaller than the DEM grid **then**
14: **exit loop**
15: **end if**
16: $\mathbf{p} \leftarrow$ mean of positions $\mathbf{p}_1 \dots \mathbf{p}_{\lfloor Fn \rfloor}$
17: $r \leftarrow Ar$
18: $\mathbf{C} \leftarrow$ circle centered at $\mathbf{p}$ with radius r
19: **end loop**
20: **end for**
21: **return** $\mathbf{p}_{\text{best}}$

region. I chose these two parameters together: a smaller value of F focuses the algorithm on moving the exploration region to superior positions, while a larger value of A ensures that consecutive exploration regions will overlap sufficiently, leading to comparison with several of the prior iteration's candidate positions. Parameter M limits the number of loop iterations. I chose a value for M that, along with A, eventually reduces the area of the exploration region to 12% of its initial value, which seems an adequate amount of focus. Parameter n_0 determines the number of grid squares the algorithm will use, and hence

the number of fitness-function evaluations that will be performed during each iteration, essentially acting as an "effort" parameter.

Steps 1–5 initialize the algorithm. $\mathbf{C}_E$ is the initial exploration region, centered at $\mathbf{p}_E$ with radius r_E. The node will not leave this region. $\mathbf{C}$ is the current exploration region, centered at $\mathbf{p}$ with radius r. The current exploration region will shift and shrink as the algorithm proceeds, focusing its attention on regions around better positions. $\mathbf{p}_{\text{best}}$ 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 (unless it is terminated early by Steps 13–15). **Steps 7–19** form an inner loop that searches within the exploration region that is centered on $\mathbf{p}_E$. **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 3.7 I characterize the effect of changing the value of n_0. The actual number of complete grid squares in the partition is n, a value that is used in several subsequent steps. (Derivation of the equation that computes the spacing of the grid can be found in Appendices C and D.) **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

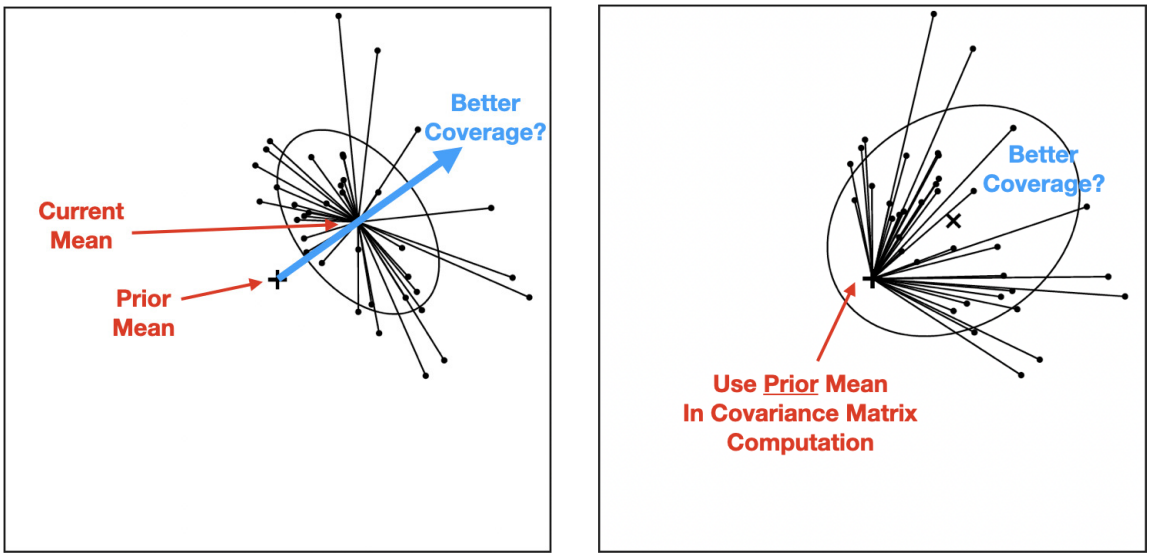
algorithm degenerates to exhaustive search of the exploration region. **Steps 16–18** shift the current exploration region to the mean of the top $\lfloor Fn \rfloor$ positions just evaluated and shrinks the exploration region. This reduction in the exploration area while maintaining the value of n_0 increases the density of the evaluated positions. **Step 21** returns the best position found.

Grid Partition is evaluated, along with other deployment algorithms, in Section 3.6.

3.5 EMNA/global,steps

My $\text{EMNA}_{\text{global,steps}}$ algorithm is a modification of $\text{EMNA}_{\text{global}}$[30]. While $\text{EMNA}_{\text{global}}$ uses the traditional definition of covariance to update a multivariate Gaussian distribution, $\text{EMNA}_{\text{global,steps}}$ adopts the strategy of CMA-ES, which is to compute the covariance matrix by referencing the mean of the previous generation instead of the current generation's mean. As explained in [21], this approach measures the variances of successful individuals' *steps*, continuing the search in the direction of improvement and helping to avoid premature convergence. Fig. 3.5 illustrates the benefit of this approach using an example.

My implementation of $\text{EMNA}_{\text{global,steps}}$ is shown in Algorithm 3.3. As input, the algorithm accepts a starting position $\mathbf{p}_E$ and an exploration radius r_E. It also accepts four parameters: n is the number of *individuals* (candidate solutions) per generation, M determines the number of generations, F is the fraction of individuals that will be used



(a) $\text{EMNA}_{\text{global}}$ computes a covariance matrix normally, by referencing the current generation's mean.

(b) $\text{EMNA}_{\text{global,steps}}$ computes a covariance matrix by referencing the prior generation's mean.

Figure 3.5: Demonstration of the effect of the reference mean on an EDA algorithm's covariance computation. Fig. 3.5a marks the prior generation's mean with a + and plots line segments from the current generation of fittest individuals to their mean, which is used in their covariance computation. The resulting $\text{EMNA}_{\text{global}}$ covariance matrix is indicated by a 50% isodensity ellipse. Notice that the search tends to deviate from the direction of improvement. Fig. 3.5b plots the same individuals as in Fig. 3.5a, but line segments indicate that the alternate covariance computation uses the *prior* generation's mean +, a strategy employed by CMA-ES[21]. The 50% isodensity ellipse of $\text{EMNA}_{\text{global,steps}}$ is centered on the mean of the fittest individuals ×, with the ellipse's shape determined by the alternate covariance matrix. The ellipse shows how the $\text{EMNA}_{\text{global,steps}}$ search better continues in the direction of improvement.

as parents of the next generation, and c is the magnitude of the initial covariance matrix. I needed to limit values for M and n to control the algorithm's execution time. The magnitude of the value that I chose for F is commonly used in population algorithms.

Step 1 defines an exploration region $\mathbf{C}_E$. All randomly generated positions will be restricted to this area. **Steps 2 and 3** define the initial mean and covariance matrix of

Algorithm 3.3 $\text{EMNA}_{\text{global,steps}}$. Inputs: starting position $\mathbf{p}_E$ and exploration radius r_E. Control parameters and their evaluated values: number of generations $M = 50$, number of individuals $n = 15$, fraction of individuals to be used as parents $F = \frac{1}{3}$, and magnitude of initial covariance matrix $c = \frac{1}{2}r_E$.

1: $\mathbf{C}_E \leftarrow$ circle centered at $\mathbf{p}_E$ with radius r_E
2: $\mathbf{p} \leftarrow \mathbf{p}_E$
3: $\mathbf{cov} \leftarrow \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix}$
4: $\mathbf{p}_1 \dots \mathbf{p}_n \leftarrow n$ random positions from bivariate Gaussian$(\mathbf{p}, \mathbf{cov})$ that also are inside $\mathbf{C}_E$
5: $\mathbf{p}_{\text{best}} \leftarrow \mathbf{p}_1$
6: **for** M times **do**
7: **loop**
8: Compute the fitness of each position $\mathbf{p}_1 \dots \mathbf{p}_n$.
9: Sort positions $\mathbf{p}_1 \dots \mathbf{p}_n$ by decreasing fitness.
10: $\mathbf{p}_{\text{best}} \leftarrow \mathbf{p}_1$ **if** $\mathbf{p}_1$ is better than $\mathbf{p}_{\text{best}}$
11: $\mathbf{cov} = \frac{1}{\lfloor Fn \rfloor} \sum_{i=1}^{\lfloor Fn \rfloor} (\mathbf{p}_i - \mathbf{p})(\mathbf{p}_i - \mathbf{p})^T$ {Compute the covariance using the *prior* generation's mean.}
12: $\mathbf{p} \leftarrow$ mean of positions $\mathbf{p}_1 \dots \mathbf{p}_{\lfloor Fn \rfloor}$
13: $\mathbf{p}_1 \dots \mathbf{p}_n \leftarrow n$ random positions from bivariate Gaussian$(\mathbf{p}, \mathbf{cov})$ that also are inside $\mathbf{C}_E$
14: **end loop**
15: **end for**
16: **return** $\mathbf{p}_{\text{best}}$

a bivariate Gaussian distribution. Initially I used $c = r_E$ in defining the initial value of the covariance matrix, but the resulting wide distribution was too similar to a disk model with radius r_E and failed to create a distribution with sufficient focus. I found an acceptable distribution when I reduced c to $\frac{1}{2}r_E$. **Step 4** generates the initial population of n random positions[27, §3.4.1 C(5)]. **Step 5** initializes $\mathbf{p}_{\text{best}}$ with an arbitrary generated position. **Steps 6–15** loop through M generations. Within the loop, **Steps 8–9** isolate the Fn fittest positions to use as parents of the next generation. **Step 10** saves the very best position seen so far. **Steps 11 and 12** update the covariance matrix and the mean of the bivariate

Gaussian distribution using the Fn parents. (As Step 11 is a matrix equation, the T superscript is the matrix transpose operator.) The order in which **cov** and **p** are computed ensures that the parents' covariance matrix uses the mean of *all* of the individuals, instead of the mean of just the parents. This choice measures the variance of the parents' *steps* away from the prior generation's mean, as desired. **Step 13** creates the next generation of random positions. Finally, **Step 16** returns the best position seen.

$\text{EMNA}_{\text{global,steps}}$ is evaluated, along with other deployment algorithms, in Section 3.6.

3.6 Evaluation Methodology

To evaluate my algorithms, I compared them against a benchmark suite of several traditional optimization algorithms. Details of the baseline algorithms that I include for comparison purposes are described in Section 3.6.1. The algorithms are parameterized, but since the values of parameters R_C, R_S, and R_E are tightly related, Section 3.6.2 explains. The table in Section 3.6.3 shows the values of the remaining parameters.

3.6.1 Benchmark Algorithms

Below are descriptions of the benchmark algorithms.

Gradient Descent

If the domain of the fitness function WLU($\mathbf{p}$) is continuous, and its gradient ∇WLU($\mathbf{p}$) is defined (and possibly its Hessian matrix ∇^2WLU($\mathbf{p}$) also is defined), then one can use gradient-based optimization techniques[25, Ch. 5][40, Ch. 3]. However, as I outlined in Section 2.1, my fitness function's domain is discrete, and so I must use gradient estimation techniques, as are used in fields such as volume graphics[15].

In the case of discrete fitness function optimization, a simple method for estimating gradients is to approximate each partial derivative using *central differences*. With this method, one computes the slopes of the fitness function using nearby data points. An estimate of the gradient vector at (x, y) using the central-differences method is shown below.

$$\nabla \mathrm{WLU}(x, y) \approx \frac{1}{2} \begin{bmatrix} \mathrm{WLU}(x + \Delta, y) - \mathrm{WLU}(x - \Delta, y) \\ \mathrm{WLU}(x, y + \Delta) - \mathrm{WLU}(x, y - \Delta) \end{bmatrix}$$

If I were to use the central-differences method to estimate gradients, then I could employ gradient descent or quasi-Newton methods to find the value of $\mathbf{p}$ that maximizes the fitness function WLU($\mathbf{p}$)[25].

However, during an initial test of this method, I discovered that abrupt changes in the cumulative visibility function pass into the derivative approximation and interfere with the ability of the gradient descent algorithm to find a good solution. One way to smooth the estimated gradient is to use least-squares fitting, specifically to fit a plane to a set of points immediately around the node and use the plane's slope as the estimated gradient.

To that end, I implemented an algorithm using a derivative approximation via least-squares estimation. A plane is fit to a set of w^2 WLU values $\mathrm{WLU}_i = \mathrm{WLU}(x_i, y_i)$ for $i \in \{1 \ldots w^2\}$, with the values arranged in a $w \times w$ grid that is centered on the position of interest. I evaluated $w = 3$ in an earlier version of this algorithm but was dissatisfied with the noise level that remained, however I was satisfied with the amount of remaining noise when $w = 5$.

While it is straightforward to use least-squares fitting to estimate gradients, I note that the regular-grid terrain representation lets one simplify the fitting algorithm to be as simple as a weighted average, as follows: One can find the direction of the plane's steepest ascent (b_x, b_y) by solving this linear system[70, p. 363]:

$$\begin{bmatrix} \sum x_i \mathrm{WLU}_i \\ \sum y_i \mathrm{WLU}_i \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & \sum x_i y_i \\ \sum x_i y_i & \sum y_i^2 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \end{bmatrix} \tag{3.4}$$

Next, one can simplify (3.4) through two modifications. First, scale the values of x_i and y_i to integers by using a spacing that matches the raster grid of the DEM data. Second, the algorithm translates the coordinates to be centered at the origin so that $x_i \in \{-2, -1, 0, 1, 2\}$ and $y_i \in \{-2, -1, 0, 1, 2\}$. These modifications result in $\sum x_i y_i = 0$ and $\sum y_i^2 = \sum x_i^2$, which one can substitute into (3.4).

$$\begin{bmatrix} \sum x_i \mathrm{WLU}_i \\ \sum y_i \mathrm{WLU}_i \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & 0 \\ 0 & \sum x_i^2 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \end{bmatrix} \tag{3.5}$$

Also, since my algorithm relies on the gradient (b_x, b_y) for the *direction* of steepest ascent but not its magnitude, one can scale both b_x and b_y by $\sum x_i^2$ without affecting the direction.

Algorithm 3.4 Gradient Descent. Inputs: starting position $\mathbf{p}_E$ and exploration radius r_E. Control parameter and its evaluated value: $w = 5$.

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

This change results in a simpler gradient (c_x, c_y) that yields the same ascent direction.

$$\begin{bmatrix} \sum x_i \text{WLU}_i \\ \sum y_i \text{WLU}_i \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_x \\ c_y \end{bmatrix} \tag{3.6}$$

Then one can solve directly for weighted averages c_x and c_y.

$$c_x = \sum x_i \text{WLU}_i \tag{3.7}$$

$$c_y = \sum y_i \text{WLU}_i \tag{3.8}$$

Then using two-argument arctangent, one can compute the direction of the steepest ascent of WLU.

$$\theta = \texttt{atan2}\left(\sum y_i \text{WLU}_i, \sum x_i \text{WLU}_i\right) \tag{3.9}$$

My implementation of Gradient Descent is illustrated in Algorithm 3.4. As input, the algorithm accepts an initial position $\mathbf{p}_E$, an exploration radius r_E, and the parameter w. **Steps 1–2** initialize the algorithm: $\mathbf{C}_E$ is the initial exploration region, centered at $\mathbf{p}_E$

with radius r_E. The node will not leave this region. $\mathbf{p}_{\text{next}}$ 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 algorithm uses (3.7) and (3.8) in **Step 5** and (3.9) in **Step 9**. The loop will terminate when any of three conditions is met: the slope of the computed WLU gradient is 0, the search reaches the edge of the exploration region, or the algorithm attempts to revisit a position that it already had visited in a prior loop iteration. **Step 11** returns the best position found.

Pattern Search

An alternative optimization scheme that requires no derivative computations is *Direct Search*[71]. In Direct Search methods, one or more sets of parameters are compared using a fitness function, and then an iterative step is taken to update the parameters. This process repeats until the state of the system converges or until the number of iterations exceeds some threshold.

A simple Direct Search algorithm is Pattern Search, which was used by Fermi and Metropolis, as mentioned in [11]. Pattern Search compares the fitness of a candidate solution with the fitness of nearby candidate solutions that one obtains after adjusting individual components of the candidate solution by a step $\pm\Delta$. For my distributed algorithm with

Algorithm 3.5 Pattern Search. Inputs: starting position $\mathbf{p}_E$ and exploration radius r_E. Parameters: minimum step $r_{\min}$ = size of DEM grid.

```
1: C_E ← circle centered at p_E with radius r_E
2: p ← p_E
3: r ← r_E
4: while r > r_min do
5:    p_1 ← (x, y + r)
6:    p_2 ← (x, y − r)
7:    p_3 ← (x + r, y)
8:    p_4 ← (x − r, y)
9:    Compute the fitness of p, p_1 ... p_4.
      {Assign fitness 0 to any p_i that falls outside C_E.}
10:   if p is more fit than all p_i then
11:      r ← r/2
12:   else
13:      p ← fittest of p_1 ... p_4
14:   end if
15: end while
16: return p
```

only two parameters (x, y), the four nearby candidate solutions are

$$(x \pm \Delta, y)$$

$$(x, y \pm \Delta)$$

In each iteration, classic Pattern Search accepts the first candidate solution that improves the fitness function. If, during an iteration, all four candidate solutions are evaluated without finding an improvement, then Δ is reduced by one half, and the process continues. Eventually the algorithm reaches a stopping condition (for instance a maximum iteration count or a minimum value for Δ).

I implemented the classic Pattern Search algorithm[23][40, §9.3], except that given the small number of necessary function evaluations, I modified the algorithm to evaluate

all four candidate positions before choosing the best one. (The classic algorithm accepts the first of the candidates that improves the fitness.) Algorithm 3.5 shows my Distributed Pattern Search implementation.

The algorithm accepts an initial position $\mathbf{p}_E$, an exploration radius r_E, and a parameter $r_{\min}$, 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 $\mathbf{p}_E$ and r_E. The value of r determines when the algorithm will terminate, as explained in the next step. **Steps 4–15** loop until $r \leq r_{\min}$. **Steps 5–8** compute four candidate positions: $\mathbf{p}_1 \ldots \mathbf{p}_4$. These positions are r away from the current position $\mathbf{p}$ in all four cardinal directions. **Step 9** computes the fitness of all of the positions under consideration, using the fitness value of 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 p to the fittest position or, if $\mathbf{p}$ already is in the fittest position, cut r in half. **Step 16** returns the best position.

Simplex Method

I also considered using the Direct Search based Simplex Method by Nedler and Mead[39][40, §9.5]. The Simplex Method, when optimizing the positions of a single node ($m = 1$), maintains a collection of $2m + 1 = 3$ candidate solutions. During each iteration, all of the candidate solutions are compared, and the algorithm attempts to move the worst one to a superior position. When the algorithm cannot improve the worst candidate solution, it

assumes that the candidate solutions surround a maximum, and it moves the two worst candidates toward the third candidate.

Although not identical to Pattern Search, the Simplex Method with $m = 1$ is quite similar, using three candidate solutions (instead of four), and using similar update rules. Consequently, I decided not to use it as a second Direct-Search based algorithm.

Random Placement

For the **Random Placement** algorithm, 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.

Regular Grid

For the **Regular Grid algorithm**, nodes are placed in a triangular grid on a subtile, a configuration that provides complete coverage on a plane. The range-limited cumulative visibility is computed for that deployment.

Greedy Adding

I implemented the **Greedy Adding** algorithm[8], which is a centralized Greedy-Set-Cover heuristic[9, §35.3] that has been modified to stop after placing all nodes of the network (Algorithm 3.6).

Algorithm 3.6 Centralized Greedy Adding.

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

3.6.2 Communication, Exploration, and Sensing Radii

Since all of the distributed algorithms that I tested use WLU as a fitness function, and since a node that computes its WLU needs to know the viewsheds of its neighbors, in addition to the exploration radius (discussed in Section 2.2), I must also consider the node communication radius (or communication range). A node can compute its WLU accurately as long as it has three values: its own position, the positions of its neighbors, and an understanding of visibility over terrain. However, depending on circumstances that I describe below, a node does not need to know the positions of *all* of its neighbors for accurate WLU computation.

Before understanding this, I must account for the fact that that radio signal propagation can be vastly different than propagation of 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[50, §16.2.2]. On the other hand, even with clear line of sight (CLOS), radio propagation can be affected by fading due to multi-path effects[18][32, §7.3]. Although I acknowledge that radio propagation is more complicated than CLOS, to simplify my evaluation I use range-limited CLOS to model radio communication, as described below. (Other radio propagation regimes will be considered as part of future work.)

I use communication radius R_C to specify the maximum distance between two nodes that allow them to be in direct communication. I show below that with planar terrain, there is a relationship between R_C, the exploration radius R_E, and the sensing range R_S (the maximum CLOS distance to a detectable event).

First consider the case when all nodes are immobile. To properly compute their WLUs, two neighboring nodes must know, based on their sensing range, 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. Then, 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 single mobile node (assuming the same sensing and radio-propagation models). If a mobile node is surrounded by immobile neighbors, and it is evaluating a potential move, then for the node to have neighbor-position data for an accurate WLU computation, the required communication radius must be increased by the maximum possible distance that the node could move toward a neighbor before communicating with the neighbor again. I call this distance the node's *exploration radius* R_E.

$$R_C \geq 2R_S + R_E$$

The prior inequality assumes immobile neighbors, but I must consider that neighbors also can be mobile. 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 the constraint relation in its most general form:

$$R_C \geq 2R_S + 2R_E \tag{3.10}$$

I can rearrange (3.10) to create a constraint on R_E:

$$R_E \leq \tfrac{1}{2}R_C - R_S \tag{3.11}$$

This constraint on the exploration radius is sufficient to prevent nodes that are on planar terrain and that are separated by a distance greater than R_C from inadvertently covering the same area of interest after moving. Consequently, in the case of planar terrain, nodes that limit their exploration range using (3.11) always will have an accurate view of WLU.

However, nodes on 2.5D terrain could be unable to communicate and then have overlapping viewsheds, for example if the nodes are on opposite sides of a hill. And so even nodes that limit their exploration using (3.11) are not certain to make accurate WLU computations. That said, it still could be beneficial to limit node exploration, and so I evaluated the sensitivity of algorithm performance to different values of R_E. Given the 240×180 size of the benchmark tiles, I chose a sensing radius of $R_S = 50$, which allows a triangular grid deployment of 10 nodes to completely cover flat terrain. Then I chose a communications radius $R_C = 130$. This value is larger than $2R_S$ so that nodes with nearly overlapping coverage may be able to communicate, but it is not so large as to cause all node broadcasts to be global. Using (3.11) I compute $R_E = \frac{1}{2}R_C - R_S = \frac{1}{2} \times 130 - 50 = 15$. To evaluate the importance of R_E, I ran simulations with $R_E \in \{10, 15, 21, 34, 51, 76\}$.

Table 3.1: **Control Parameters.** I ran simulation using these values of control parameters. When sweeping values of m, I set $R_E = 15$ and $n_0 = 10$. When sweeping values of R_E, I set $m = 10$ and $n_0 = 10$. When sweeping values of n_0, I set $m = 10$ and $R_E = 15$.

Algorithm	**Parameter Purpose**	**Name**	**Value**
Framework	Termination Condition	C	4
Framework	Iteration Count	L	10
Framework	Number of Nodes	m	$\{10, 20, 30\}$
Framework	Communications Radius	R_C	130
Framework	Init. Exploration Radius	R_E	$\{10, 15, 21, 34, 51, 76\}$
Framework	Sensing Radius	R_S	50
Grid Partition	Reduce Exploration Region	A	0.9
Grid Partition	Frac. of Positions to Average	F	0.25
Grid Partition	Iteration Count	M	10
Grid Partition	Partition Count	n_0	$\{5, 10, 21\}$
Simulated Annealing	Reduce Iteration Count	β	0.9
Pattern Search	Minimum Step	$r_{\min}$	1
$\text{EMNA}_{\text{global,steps}}$	Mag. of Init. Cov. Matrix	c	$\frac{1}{2}r_E$
$\text{EMNA}_{\text{global,steps}}$	Frac. of Indiv. as Parents	F	$\frac{1}{3}$
$\text{EMNA}_{\text{global,steps}}$	Number of Generations	M	50
$\text{EMNA}_{\text{global,steps}}$	Number of Individuals	n	15

3.6.3 Parameters

In addition to the parameter values mentioned in the prior section, I evaluated all algorithms using 10, 20, and 30 nodes. All simulations are run using 10 different random seeds and on the 21 benchmark terrain subtiles discussed in Section 2.1, meaning that each algorithm is simulated 210 times for each parameter set. Table 3.1 summarizes the parameter values that I used.

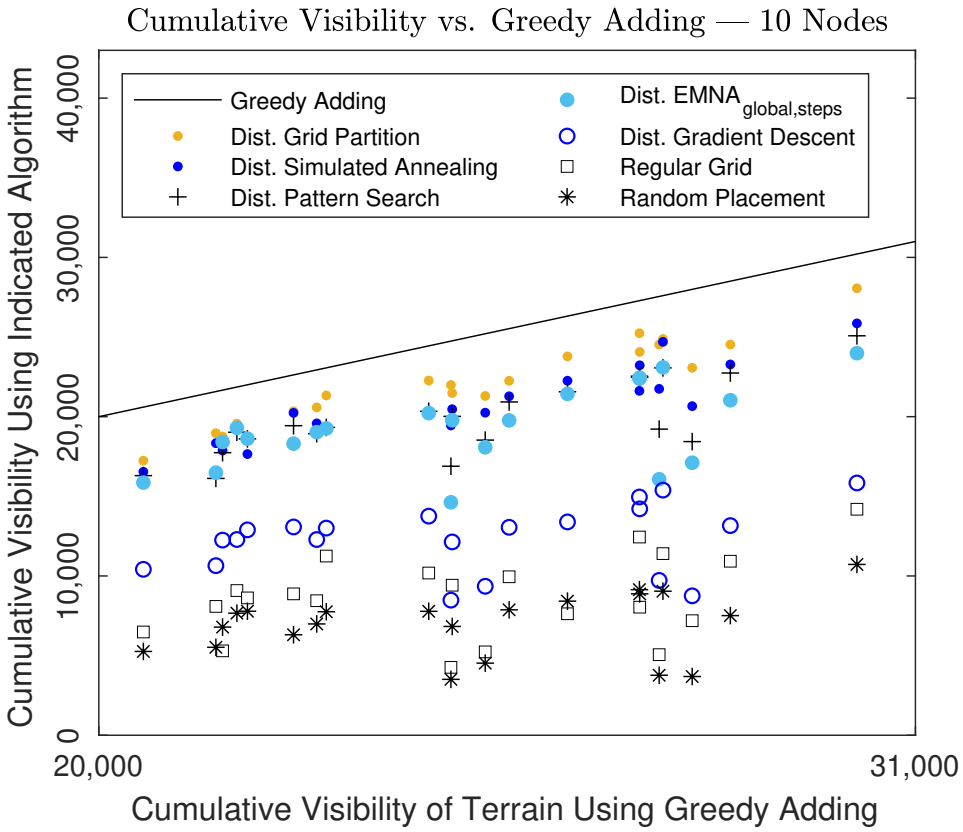


Figure 3.6: Results of algorithms on test terrains vs. centralized Greedy Adding—10 nodes.

3.7 Results

In this section, I present the performance of the algorithms, compare the effectiveness of the two terrain classifiers Greedy Adding and ACV (described in Section 2.1), and discuss the effect of different values of the Exploration Radius.

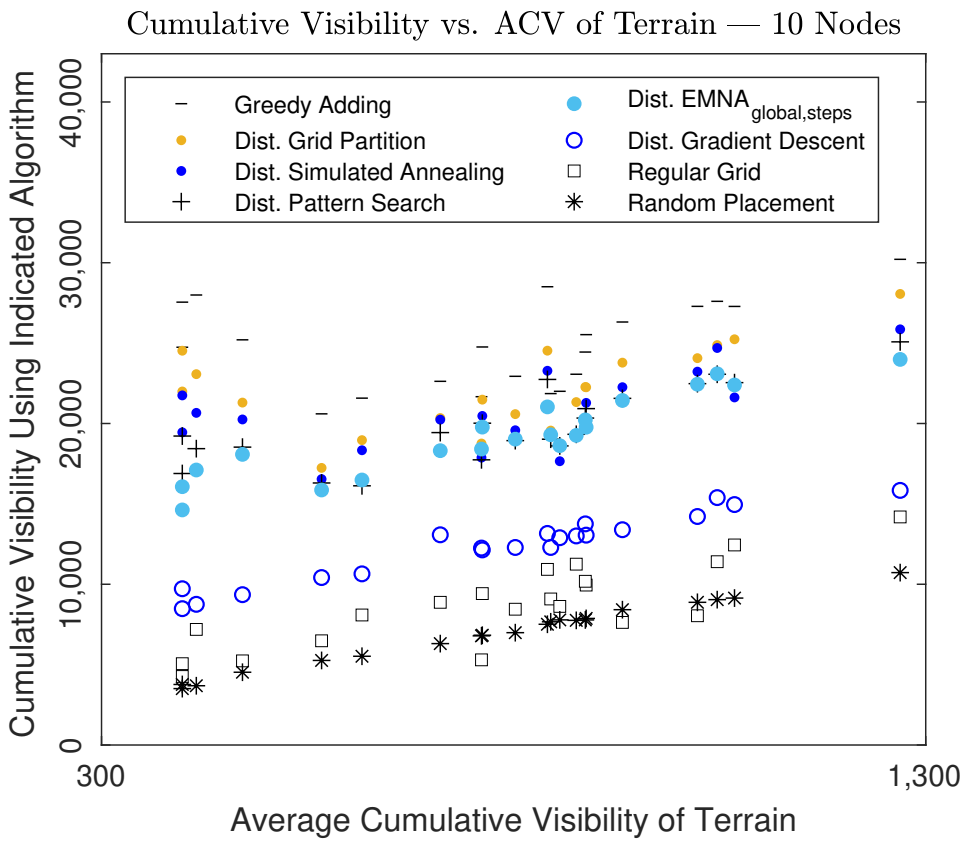


Figure 3.7: Results of algorithms on test terrains vs. ACV of terrain—10 nodes.

3.7.1 Algorithm Performance

I simulated all algorithms with 10-, 20-, and 30-node deployments. Overall algorithm results are presented in Table 3.2, with each entry representing the average value over all benchmark terrains and over all random seeds. To allow for more straightforward comparisons between algorithms, this table reports the cumulative visibility results as a percentage of Centralized Greedy Adding cumulative visibility. Detailed results are plotted in Figs. 3.6 and 3.7. Results for 20 and 30 nodes are similar and can be found in

Table 3.2: Distributed 2.5D IoT Deployment Algorithms Performance: Cumulative Visibility and Number of Fitness Computations. Results are averages over all 21 benchmarks. For comparison, visibility results appear as a percentage of Centralized Greedy Adding.

	Cumulative Visibility for Number of Nodes			**Fitness Comps per Node for Number of Nodes**		
Algorithm	**10**	**20**	**30**	**10**	**20**	**30**
Cent. Greedy Adding	100%	100%	100%	—	—	—
Dist. Grid Partition	88%	89%	90%	745	782	793
Dist. Simulated Annealing	83%	78%	76%	1025	1236	1319
Dist. Pattern Search	80%	86%	88%	184	189	188
Dist. $\text{EMNA}_{\text{global,steps}}$	78%	87%	91%	10 154	7244	7624
Dist. Gradient Descent	50%	60%	69%	251	300	330
Regular Grid	35%	43%	51%	—	—	—
Random Placement	28%	36%	44%	—	—	—

Appendix A.

As expected, centralized Greedy Adding provides an upper target for comparison against the distributed algorithms. Looking first at results for 10 nodes, Grid Partition shows the best overall results at 88% of Greedy Adding, followed by Simulated Annealing and Pattern Search. The superior performance of Grid Partition can be explained by its unique fitness function, which is a combination of WLU and local terrain height. I was surprised that Pattern Search, at 80% Greedy Adding, did so well with so few evaluations of the fitness function. The result demonstrates the effectiveness of this classic strategy.

My Population Based algorithm, $\text{EMNA}_{\text{global,steps}}$, did about as well as Pattern Search in terms of coverage, but it required nearly two orders of magnitude more fitness computations. As discussed in Section 3.2.2, the related CMA-ES algorithm is quite

computationally expensive when the number of variables is relatively small, and in my case of only two variables, I anticipated this result. However, later I reveal an unanticipated improvement when I discuss increasing the number of nodes.

Gradient Descent, with its 5×5 grid for gradient estimation, surprised me with its somewhat poor performance. Although I had chosen $w = 5$ because I feared that more than 25 fitness computations needed at each position would be computationally expensive, I discovered that the algorithm requires relatively few fitness computations. An algorithm like Gradient Descent, in which nodes look only a few cells away from their current location and can be easily trapped in a local optimum, has a disadvantage compared to other algorithms like Pattern Search and Grid Partition, which will evaluate positions much farther away before declaring victory. Exploring larger values for w is part of future work.

For easier comparison, Fig. 3.8 plots the results compared to Greedy Adding for 10, 20, and 30 nodes. I observe that the relative performance of most algorithms in terms of coverage is maintained, suggesting that their relative performance is unaffected by node count—and hence unaffected by node density as well, since the areas of the benchmark subtiles are unchanged. Interestingly, I observe that as node density increases, the performance of most algorithms improves compared to Greedy Adding. My suspicion is that the increased node density decreases the average distance that a node can travel before it causes multicoverage. Put another way, it is easier for a node that has many neighbors to find its optimal position nearby. This explanation best describes the improvement seen in Distributed Gradient Descent, which is less likely to encounter a local optimum when

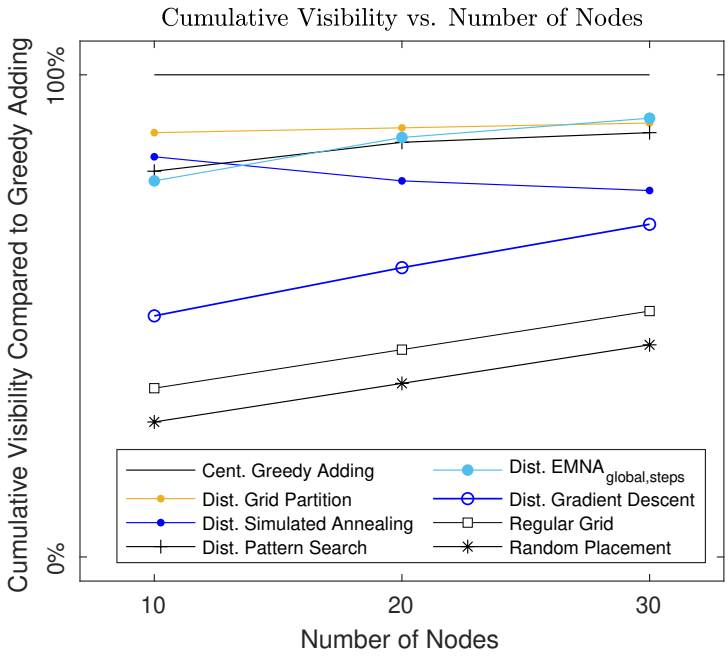


Figure 3.8: Cumulative visibility compared to Greedy Adding vs. 10, 20, and 30 nodes.

the distance that the node must travel is shorter.

Two of the algorithms' responses to changes in the node counts surprised me. First, the performance of Distributed Simulated Annealing worsened with increasing node density as compared to Greedy Adding. Since, unlike other algorithms, Distributed Simulated Annealing uses a fixed R_E and a decreasing T to focus its search (instead of using a decreasing r_E), this result suggests that for Distributed Simulated Annealing node density should be considered when choosing either R_E or the annealing schedule.

The second surprise is that $\text{EMNA}_{\text{global,steps}}$ yielded superior performance (slightly higher than Distributed Grid Partition and Distributed Pattern Search) in simulations with 30 nodes. Understanding the root cause of this behavior will be part of future work.

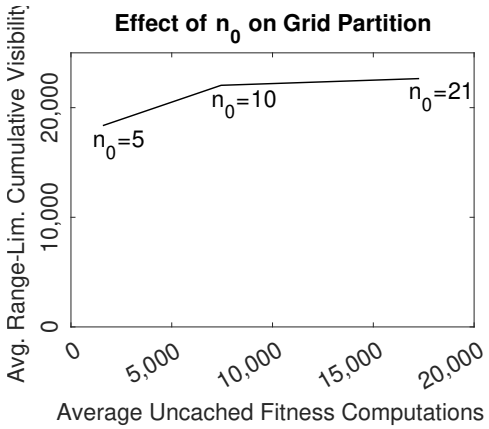


Figure 3.9: Effect of the value of n0 on Grid Partition results.

Finally, I evaluated the effect of n_0 on the performance of Grid Partition. Recall that the n_0 parameter controls the amount of effort that the algorithm expends. 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 compute 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. 3.9 shows that $n_0 = 10$ provides adequate cost-performance trade-off for the sample terrains examined.

3.7.2 Exploring the Exploration Radius

In Section 3.6.2 I explained that there may be a tradeoff between WLU accuracy and the exploration radius R_E. Recall that inequality (3.11) suggests that for my chosen simulation parameters, an ideal value for R_E is 15, but only in 2D. To exam-

ine whether such a tradeoff exists over 2.5D terrain, I ran additional simulations with $R_E \in \{10, 15, 21, 34, 51, 76\}$ using a subset of the algorithms, choosing Distributed Grid Partition, Distributed Pattern Search, and Distributed Gradient Descent. Fig. 3.10 shows the results. Each graph shows two curves, one for cumulative visibility (coverage, in blue) and the other for the number of fitness computations (computational cost, in brown).

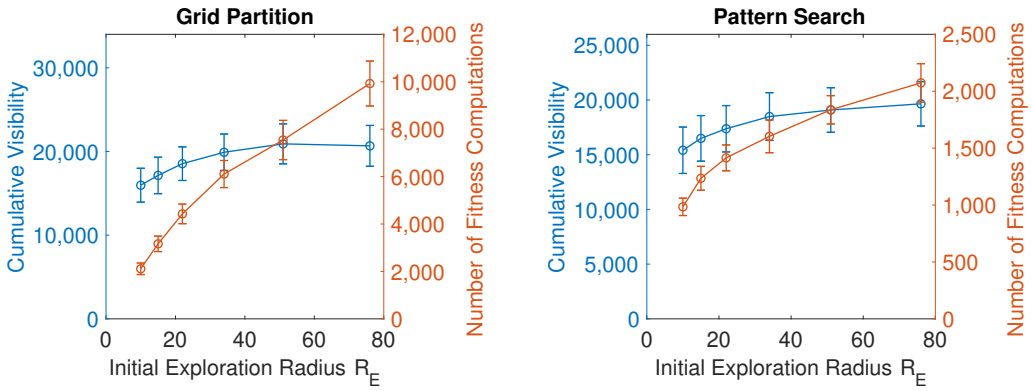
Considering Grid Partition first, one can see that $R_E = 15$ is not the optimal value, since the results continue to improve up to $R_E = 51$. With $R_E = 76$ the number of fitness computations continues to increase, but algorithm performance actually worsens. Pattern Search also shows improvement with $R_E > 15$, along with an increase in the number of fitness computations. Finally, Gradient Descent shows that changing R_E has almost no effect on performance and the number of fitness computations. In Section 3.7.1 I conjecture that Gradient Descent is easily trapped in a local optima. The observation that performance hardly changes at all between $R_E = 10$ and $R_E = 15$ suggests that most nodes find a local optimum before moving 15 units.

These results show that adhering to the strict constraint in (3.11) with 2.5D terrain does more harm than good. I observe that choosing a value for R_E can help an algorithm achieve an adequate balance between performance and computational cost, but the argument that leads to (3.11) must be reconsidered. It is possible that algorithms that can properly compensate (backtrack) away from multicoverage do not need the additional constraint that (3.11) imposes.

3.8 Discussion

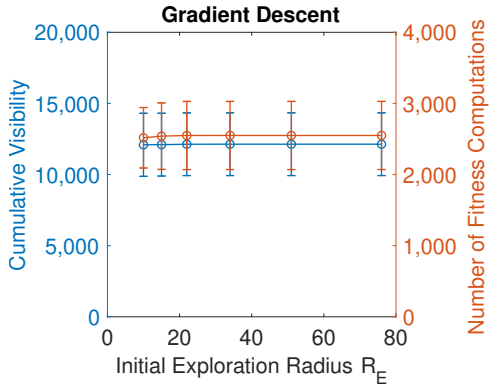
In this paper, I evaluated my distributed deployment algorithms for outdoor IoT camera networks over 2.5D terrain. I compared my own algorithms (Distributed Grid Partition and Distributed $\text{EMNA}_{\text{global,steps}}$) to traditional optimization algorithms (Distributed Simulated Annealing, Distributed Pattern Search, and Distributed Gradient Descent), and to centralized baseline algorithms (Regular Grid, Random Placement, and Greedy Adding). The results show that my algorithms compare favorably. I used TAFFI in conjunction with two terrain classifiers that I proposed, Greedy Adding and ACV (Average Cumulative Visibility), in order to subject the 2.5D deployment algorithms considered in my study to a variety of terrains ranging from flatter to rougher topography, suggesting that the proposed terrain classifiers can serve as a valuable part of a 2.5D-benchmark selection methodology.

My future work includes improving the framework's communication model to consider other radio propagation regimes, adding network connectivity constraints, characterizing the effect of the number of generations on $\text{EMNA}_{\text{global,steps}}$, looking further at the algorithms' scalability-coverage tradeoff (e.g., deployments beyond 30 nodes), and exploring cost-performance tradeoffs of the Distributed Gradient Descent algorithm.



(a) Distributed Grid Partition

(b) Distributed Pattern Search



(c) Distributed Gradient Descent

Figure 3.10: Comparison of Range-Limited Cumulative Visibility and the number of uncached Fitness Computations in the distributed algorithms that were tested.

Chapter 4

Outdoor IoT Recovery Algorithms

In this chapter I describe the suite of TARCI recovery algorithms. The chapter is organized as follows: Section 4.1 describes related work. Section 4.2 presents the TARCI algorithm. The algorithm is evaluated in Section 4.3, and results are presented in Section 4.4.

4.1 Related Work

Recovery from node failures has been well researched for deployments over two-dimensional (2D) planes, with researchers proposing numerous mechanisms. The surveys in [73][74] classify and describe existing 2D recovery approaches. In the remainder of this section, I reference their taxonomy of fault tolerance to discuss related work and its applicability to 2.5D terrain.

4.1.1 Proactive Methods

One class of recovery approach is based on *proactive* mechanisms. For example, avoiding faults through over-provisioning. One can do this by deploying k-connected wireless networks where $k > 1$, meaning that every node is connected to the sink by k node-disjoint paths. Such a strategy ensures that a network can withstand up to $k - 1$ node failures without becoming partitioned.

An example of this approach using $k = 2$ is Basu & Redi[4]. In their work, a bi-connected network is created from a connected network by repeatedly identifying a connected subset of nodes and then moving the subset as a block toward the rest of the network. Each block of nodes moves until the appearance of a redundant path eliminates a cut vertex.

I note some shortcomings to this approach. First, while provisioned approaches in general prevent consequential loss of connectivity following a node failure, such solutions usually bear the cost of requiring redundant nodes. Second, if a provisioned algorithm does not add redundant nodes, then the area of coverage will be reduced by the necessity of moving nodes closer together to create redundant paths. Third, proactive approaches incur overhead even when no faults happen.

I also note a drawback from using block moves over 2.5D terrain. Any strategy for 2D can assume that block moves that maintain intra-block node-to-node distances also will maintain intra-block node-to-node communication. However, with 2.5D terrain, two

nodes of a moving block may lose contact if one node drops into a valley or if both nodes move to opposite sides of a hill. That is, generally, any algorithm that assumes that a block move will maintain intra-block communication is inappropriate for 2.5D terrain.

4.1.2 Reactive Methods

Reactive methods are an alternative to proactive approaches where recovery mechanisms are activated only when a node failure that causes the network to partition is detected.

One example of a reactive approach is RIM (Recovery by Inward Motion)[75]. The RIM algorithm works as follows: after detecting a node failure, the node's 1-hop neighbors move inward until they form a connected subnetwork. Then, if the inward motion of the failed node's 1-hop neighbors causes disconnection with any of *their* neighbors, there will be a cascading motion of nodes, with each node moving toward its inner neighbor until it reconnects.

The problem with RIM when used with 2.5D terrain is that it assumes that nodes that are closer than R_C can communicate. However, as I mentioned earlier, on 2.5D terrain, any repositioning that ignores hills and other obstructions can cause loss of communication, regardless of inter-node distance. Consequently, RIM is not appropriate for 2.5D terrain deployments.

Another example is DARA[1], an algorithm that is similar to RIM but that avoids the shortcomings of deployment over 2.5D terrain. DARA will direct a node to move only to

a position that was just vacated by the mobility of loss a neighboring node. Consequently, a DARA-recovered network never attempts to establish new links between node positions. That is, DARA's cascading node motion will change which nodes occupy which positions, but the *set of positions* will remain unchanged, and hence connectivity between positions will not be altered. (That is, aside from any connections to the position that eventually is abandoned as part of DARA's network recovery effort.) This "reuse" of node positions by DARA means that it will work with 2.5D terrain. However, I must note that the very position restrictions that let DARA ensure node connectivity also prevent the algorithm from considering new connected positions that would mitigate coverage loss or even improve coverage.

4.1.3 Hybrid Methods

As just discussed, it's possible that a recovery mechanism can be purely reactive, with nodes formulating a strategy only after the need for network recovery is detected, but alternative *hybrid* approaches also are possible. Hybrid approaches have two phases: a *pre-failure planning* phase and a *failure response* phase.

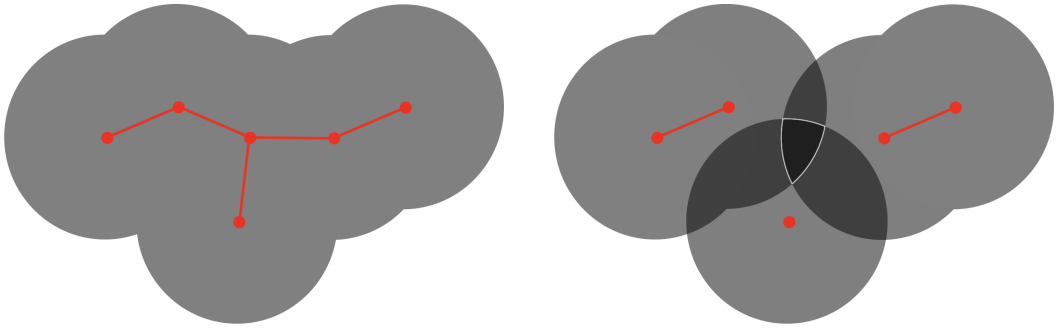
Some examples of hybrid approaches are PADRA[3], PCR[24], and NORAS[57]. During the pre-failure planning phase of these approaches, the critical nodes of the network self identify using a distributed algorithm. (Cut vertices are critical nodes, but a heuristic algorithm may mistake a non-cut-vertex as a critical node.) Then each critical node uses a heuristic mechanism to designate a recovery node for itself (also called a *backup*) that will

activate should the critical node fail.

After the pre-failure planning phase is complete, the network is primed to respond to the failure of a critical node. For these algorithms the failure response is similar: following the failure of a critical node, the corresponding recovery node moves to take the position of the failed node, possibly initiating cascading node movement in order to maintain network connectivity. Critical nodes of PADRA and PCR potentially select a recovery node that is not an immediate neighbor, and so to balance energy usage, cascading node movement is initiated along the path between the failed node and its backup.

These approaches have a few shortcomings. First, as already noted, the distributed algorithms that identify cut vertices as critical nodes sometimes mistakenly designate a node that is *not* a cut vertex as critical. Although such a mistake will not harm connectivity—since failure of a mistakenly identified node merely will activate the recovery mechanism on a still-connected network—unnecessary activation causes unnecessary energy usage.

The second shortcoming is apparent when one considers 2.5D terrain. While the examples that I have cited are similar to DARA in that they direct a node to move only to a position that was just vacated by a neighboring node (meaning that node-to-node connectivity assumptions will be maintained with 2.5D terrain), these position restrictions prevent the algorithm from considering new connected positions that would mitigate coverage loss or would improve coverage.



(a) Commshed of a network. The gray region indicates positions that are within distance R_C of at least one node of the network. A new node that is placed anywhere in the commshed can communicate with the network.

(b) Commsheds of a partitioned network. The darkest gray region indicates the intersection of all of the partitions' commsheds. A new node that is placed anywhere in the intersection can communicate with all three partitions and will repair the network.

Figure 4.1: Conceptual example of using commshed intersections to identify candidate positions for a recovery node. For clarity this example uses the 2D disk model, but one could extend the example to 2.5D by replacing the disk model with a terrain-based range-limited CLOS model, similar in appearance to the viewsheds shown in Fig 1.1.

4.2 Terrain Aware Recovery with Commshed Intersections

To address the requirements of Outdoor IoT deployments over 2.5D terrain, I propose a suite of hybrid repair algorithms. I call my algorithms TARCI for Terrain Aware Recovery with Commshed Intersections. TARCI leverages the notions of commsheds and viewsheds as they quantitatively capture my goals of connectivity restoration and coverage-loss mitigation. Based on available 2.5D terrain models, TARCI uses this data to compute the commsheds and viewsheds that guide its pre-failure planning decisions.

In other words, the key idea behind TARCI is to use commsheds and viewsheds to compute the set of recovery positions for a partitioned network by finding the intersection of the partitions' commsheds (Fig 4.1). Then, given a set of potential recovery positions

provided by the commshed intersection, one can compute the viewsheds of each of the potential recovery positions to prioritize the positions based on the amount of additional coverage that each potential recovery position provides.

Ideally one would evaluate all potential recovery positions within the commshed intersection, but during algorithm development I observed that in some circumstances the commshed intersection is quite large, meaning that one should try to limit the number of potential recovery positions (and corresponding viewsheds) that are evaluated during the pre-failure planning phase. So to understand the necessity and benefits of such limits, I developed and evaluated four variants of TARCI that use different approaches to prioritize potential positions for each candidate recovery position:

- TARCI-E exhaustively considers all possible new positions for a recovery node. Although it expends the most energy during pre-failure planning of all of the algorithm variants, it will find the optimal recovery solution. Hence, I use TARCI-E as a baseline in my evaluations, i.e., the worse case for energy consumption and the best case for recovery optimality.

- TARCI-H uses the height of each possible new position as a proxy of the position's potential communication effectiveness, and so it evaluates the highest H positions of the commshed intersection. It expends much less energy during pre-failure planning than TARCI-E, as determined by control parameter H. When choosing a value for control parameter H, there are trade-offs between the quality of the recovery solution

and the pre-failure energy usage. I discuss these tradeoffs in Section 4.4.

- TARCI-GP is similar to TARCI-H, but it uses my Grid Partition[60] mechanism to distribute the considered positions across the entire commshed intersection, rather than considering only the highest positions, which will tend to cluster together. Grid Partition places a grid of approximately H squares over the commshed intersection and then evaluates the highest position of each grid square. For a given value of H, TARCI-GP expends about the same amount of energy during pre-failure planning as TARCI-H. And as with TARCI-H, the chosen value for control parameter H leads to trade-offs between the quality of the recovery solution and the pre-failure energy usage (see Section 4.4).

- TARCI-D minimizes energy usage during both pre-failure planning and recovery. During pre-failure planning, it immediately chooses the recovery position within the commshed intersection that is closest to the candidate recovery node's current position. Then during recovery, the node can reach the commshed intersection by traveling the shortest distance.

The pre-failure planning phase of a hybrid recovery algorithm typically corresponds to a relatively small amount of time compared to the network's total deployment time. I also note that the sinks usually are larger nodes that are equipped with more computation capabilities and have larger battery packs/solar panels or are connected to continuous energy sources. Hence, I made the design choice of using a centralized pre-failure planning

Table 4.1: Message Types

Message Type	**Purpose**	**Payload**
NEIGHBORS-R	Sink requests neighbor lists.	
NEIGHBORS	Node sends its neighbor list to the sink.	Neighbor List
ASSIGNMENT	Sink sends a recovery assignment to a recovery node.	Assigned Node
ACTIVATE	Sink instructs nodes to start recovery tasks.	
KEEP-ALIVE	Nodes detect the failure of an immediate neighbor.	
FAILED-NODE	A node that detects a failed neighbor broadcasts a FAILED-NODE message that identifies the neighbor.	Failed Node

algorithm instead of a distributed algorithm to: (1) identify critical nodes and (2) choose appropriate failure responses. As my experimental results demonstrate (see Section 4.4), this results in a relatively small energy usage increase due to additional communication.

4.2.1 Pre-Failure Planning

Pre-failure planning computations comprise the bulk of the recovery algorithm's work and are performed while the network is whole. The overall goal of pre-failure planning is for TARCI to select a recovery response for every cut vertex of the network and then to send appropriate instructions to corresponding recovery node(s).

To summarize this phase, pre-failure planning begins when the sink broadcasts a NEIGHBORS-R request message. (All messages exchanged are summarized in Table 4.1.) Then in response, each node sends a list of its neighbors to the sink in a NEIGHBORS message. Using this information, the sink determines the network topology. This topology

Algorithm 4.1 Pre-Failure Planning: Network Initialization by the Sink. Input: network graph N.

1: Compute each node's commshed.
2: $V \leftarrow$ cut vertices of N
3: **for** every cut vertex $v \in V$ **do**
4: **for** every node $r \in N$ where $r \neq v$ **do**
5: Evaluate the effectiveness of moving node r to recover from the failure of cut vertex v. {See Algorithm 4.2.}
6: **end for**
7: Choose the node r with the best recovery response and assign it as the recovery node for cut vertex v.
8: **end for**

is used by TARCI's pre-failure planning algortihm.

Using any selected means, TARCI identifies the network's cut vertices and then, for every cut vertex, TARCI evaluates every other node of the network as a potential recovery node which will respond if the identified cut vertex fails. After TARCI selects the best recovery node for the identified cut-vertex, the sink sends an ASSIGNMENT message to the recovery node informing it that it is the backup node for the cut vertex and where it will reposition to should the cut vertex fail. Details of TARCI's pre-failure planning algorithm are below.

Algorithm 4.1 – Network Initialization by the Sink: Input is the network graph N, and the result is the recovery response for every cut vertex of the network. The algorithm relies on knowledge of each node's commshed, and so the commsheds are computed in **Step 1**. **Step 2** identifies the cut vertices of the network. I note that since the relative execution time of this step is relatively small, one can use the simple approach of removing each vertex one-by-one and then checking for a disconnected network with

Algorithm 4.2 Pre-Failure Planning: Evaluating the effectiveness the moving node r to recover from the failure of cut vertex v. Inputs: network graph N, node r, and cut vertex v.

1: $N' \leftarrow N - r - v$
2: Split N' into its connected components $P_1 \ldots P_n$.
3: $C \leftarrow \bigcap_{i=1}^{n}$ `commshed(`P_i`)`
{Intersection C contains all of the potential new positions for node r that will reconnect the network.}
4: **if** $C = \varnothing$ **then**
5: Return failure.
6: **else**
7: Select all or a subset of the positions $c_i \in C$. {See text.}
8: For every selected c_i compute `fitness(`c_i`)`.
9: Return the best `fitness(`c_i`)` and the corresponding c_i.
10: **end if**

depth-first search. However, if one desires better asymptotic performance, Problem 22-2 of [9] leads one to a more efficient algorithm that identifies cut vertices through a traversal of a single depth-first search. **Steps 3–8** form a loop that formulates a recovery response for the failure of each cut vertex v. Within the loop, **Steps 4–6** evaluate the capability of every other node r to respond for the failure of cut vertex v. (Details of the evaluation are in Algorithm 4.2.) Then **Step 7** compares node capabilities and chooses the best node r to act as the recovery node for cut vertex v.

Algorithm 4.2 – Recovery Node Effectiveness: This subroutine accepts as input a network graph N, the position of a failed cut vertex v, and the position of a recovery node r. It returns the fitness of assigning r as the recovery node for v, or it returns failure if no repositioning of r will repair the network. **Step 1** removes cut vertex v and node r from the network N. This network modification is needed to evaluate the failure of cut vertex v along with node r moving in response. **Step 2** computes the combined effect of these

changes: a set of network partitions $P_1 \ldots P_n$. Next, **Step 3** determines which potential new positions for node r could repair the network. This set of potential new positions, C, is the intersection of all of the partitions' commsheds. **Steps 4–10** determine the result of the algorithm. If C is the empty set, then the algorithm indicates that there is no new position for node r that will let the network recover from the loss of cut vertex v. (One might think that moving node r to the position of failed cut vertex v always would repair the network, but r might be a cut vertex itself, and so finding a repair position for r is not certain.) If set C is not empty, then it contains one or more potential new positions for node r, and in **Step 7** all or some of these positions are evaluated for fitness. The set of positions that are evaluated depends on the variant of TARCI that is running (TARCI-E, TARCI-H, TARCI-GP, and TARCI-D), as mentioned earlier. **Step 8** determines the fitness of all of the selected positions, and then **Step 9** compares the fitness results and returns the best position.

Once the sink has made all recovery assignments, it broadcasts an ACTIVATE message, and the nodes start their Failure-Response Tasks.

4.2.2 Failure Response

After pre-failure planning, all nodes receive an ACTIVATE message and run a pair of failure-response tasks. One task (Algorithm 4.3) relies on KEEP-ALIVE messages to detect the failure of an immediate neighbor. Upon detecting such a failure, a node broadcasts to its partition a FAILED-NODE message that identifies the failed neighbor.

Algorithm 4.3 Node Failure-Response Task #1.

```
1: if this node detects that a neighbor has failed then
2:    Broadcast a FAILED-NODE message that identifies the failed neighbor.
3:    if this node is the recovery node for the failed neighbor then
4:       Take the assigned recovery action.
5:    end if
6: end if
```

Algorithm 4.4 Node Failure-Response Task #2.

```
1: if this node is the assigned recovery node for a received FAILED-NODE message then
2:    Take the assigned recovery action.
3: end if
```

Then, if it is the designated recovery node for the failed neighbor, it performs its assigned recovery action and repositions itself.

The other task (Algorithm 4.4) is run by nodes that have been designated as a recovery node. This task checks all FAILED-NODE messages to determine whether the recovery node needs to activate its recovery response for a failed node and reposition itself.

4.3 Evaluation Methodology

I used the TAFFI framework to evaluate the TARCI recovery algorithms using a suite of benchmarks, as described below.

4.3.1 Experimental Evaluation Infrastructure

To evaluate and compare the different variants of TARCI, I modeled and simulated its algorithms using the C++ version of TAFFI (Terrain-Aware Framework for IoT, see Chapter 2). Compared to TAFFI/Java, TAFFI/C++ better matches TARCI's centralized

algorithms since the alternative of using TAFFI/Java's Cooja-Contiki simulator would have added the complication of running a centralized algorithm in a distributed Cooja-Contiki simulation environment.

My analysis tasks were facilitated through the following TAFFI/C++ features:

- A C++ library of line-of-sight algorithms that aid the creation of command-line simulation models for 2.5D terrain. Using the library improves code quality and programming efficiency.
- A benchmark suite of actual terrains that help with reproducibility and algorithm comparisons.
- Command-line job management using GNU Make[53], preventing my inadvertently duplicating simulation runs and also helping me run simultaneous simulations on multiple servers.
- Data collection from thousands of simulation logs using GNU Awk[49] scripts.
- MATLAB[54] scripts that analyze data and generate plots as Encapsulated PostScript files, which are used with LaTeX[29] in the creation of technical papers.

Simulations were run on a compute server with two Intel® Xeon® Processor E5-2670 CPUs running at 2.60 GHz, providing the server with a total of 32 threads. For one of the performance metrics, runs measured processor time using the C++ `clock()` function.

4.3.2 Benchmark Networks

Each of the TARCI algorithm variants was evaluated for its post-recovery coverage and estimated energy usage as it repaired 475 different cut vertices in 200 benchmark networks.

Each of the 200 benchmark networks was created using a three-step process: step **(A)** choose a benchmark terrain subtile from a terrain dataset, step **(B)** place a sink in one of ten predefined positions on the subtile and deploy ten nodes into a connected network, and step **(C)** if necessary, decrease the communications range, breaking links until the network has one or more cut vertices.

Regarding step **(A)**, for my source of 2.5D benchmark terrain subtiles, I leveraged my prior work on two terrain classifiers that help me verify that a set of Shuttle Radar Topography Mission (SRTM)[16] terrain subtiles represent a wide range of landform variation[60][62]. See Section 2.1.3 for details.

Regarding step **(B)**, the ten sink positions are chosen from locations on a regular triangular grid. Using a grid ensures that the possible sink positions will be distributed uniformly over the subtile and will help create a variety of experimental setups. After placing the sink, I run *Connected Greedy Adding*, or CGA (see below for more details on CGA) to compute an initial network deployment of ten nodes. I chose to use ten-node networks because earlier work with this source of terrain data showed that ten nodes is ideal with the 240-by-180 unit terrain subtiles and my chosen value of R_C. Indeed, choosing

Algorithm 4.5 Connected Greedy Adding. Input: network graph N and position of the sink s.

1: $C \leftarrow \text{commshed}(s)$
2: **for** every node $x_i \in N$ **do**
3: Place node x_i at the position in C that best improves coverage.
4: $C \leftarrow C \cup \text{commshed}(x_i)$
5: **end for**

a larger number of nodes with the same subtile would require me to decrease the value of R_C, which would reduce the resolution of the commsheds.

The initial-deployment algorithm Connected Greedy Adding is a modification of the set-cover heuristic Greedy Adding[8] that, in addition to trying to optimize coverage, also includes steps to ensure that a network is connected. Referring to Algorithm 4.5, the salient additions are **CGA Step 1** and **CGA Step 4**, which construct and maintain the commshed C. This commshed is the set of all positions of the terrain that can communicate with the sink or with any already placed node of the network. The algorithm ensures network connectivity by construction because **CGA Step 3** places nodes only in positions that are part of the current commshed.

The result of CGA is initial positions for all nodes x_i, thereby defining a network that has good coverage and that also is connected. I repeat this first step using 200 different pairs of subtiles and sink positions to generate 200 benchmark networks.

This brings us to the optional step **(C)** of benchmark creation. Although all of the benchmark networks that are generated have good coverage and also are connected, some of the networks inadvertently are bi-connected, meaning that they lack cut vertices,

making them poor benchmarks for hybrid recovery methods. Consequently, I perform step **(C)** on all bi-connected networks, ensuring that each network has at least one cut vertex. Specifically, step **(C)** reduces the communications range R_C for all nodes of a bi-connected network until at least one cut vertex appears. After this process is complete, all of the 200 benchmark networks have a specific value for R_C and one or more cut vertices.

4.3.3 Experiments

I evaluated all of the algorithms using three performance metrics: (1) CPU time used during pre-failure planning (to compare relative pre-failure-planning energy usage), (2) change in coverage following network recovery after the loss of a cut vertex, and (3) distance traveled during network recovery after the loss of a cut vertex (to compare relative recovery energy usage).

Also, to allow comparing the performance of TARCI variants against 2D algorithms, I devised a proxy algorithm that will perform no worse than the 2D algorithms. Since 2D recovery algorithms often ignore coverage, I expect that they will perform no better than a proxy algorithm that merely moves the network's remaining nine nodes into the original positions of nodes x_1 through x_9. The idea is that nodes x_1 through x_9 are in the positions chosen by Connected Greedy Adding for a nine-node network, and so they should provide better coverage than the positions chosen by any 2D recovery algorithm that ignores coverage. I call this algorithm *9-Node CGA* and use it as an estimated upper bound for 2D recovery algorithms like DARA[1]. Since 9-Node CGA simply restores the

Table 4.2: Control Parameters

Parameter Purpose	**Name**	**Value**
Number of Nodes	m	10
Initial Communications Range	R_C	130
Event-Detection Range	R_S	50
Number of Candidate Positions	H	$\{10, 20, 30, 40, 50, 100\}$

remaining nodes to the original positions of nodes x_1 through x_9, it expends essentially zero energy during pre-failure planning.

I ran algorithms TARCI-E, TARCI-H, TARCI-GP, and TARCI-D over all 475 cut verticies of the benchmark networks. Table 4.2 shows the values of control parameters used during the experiments. I chose values of m, R_C, and R_S that worked well in other research experiments with this size of terrain subtile.

Since TARCI-H and TARCI-GP evaluate H positions, I determined the sensitivity of these algorithms to the control parameter H by repeating the tests using the six values of H listed as *Number of Candidate Positions* in Table 4.2.

4.4 Results

CPU time for pre-failure planning: Table 4.3 compares the average pre-failure-planning CPU time with the average post-recovery change in coverage. As expected, the exhaustive TARCI-E algorithm uses the most CPU time, but TARCI-GP with $H = 30$ shows nearly as good post-recovery coverage as TARCI-E while using about two orders of magnitude less CPU time for pre-failure planning! TARCI-H is similar, but its performance

Table 4.3: Pre-Failure Planning CPU Time

Algorithm	H	**Pre-Failure Planning Minutes CPU Time**	**Post-Recovery Change in Coverage**
TARCI-E	—	65	-4.7%
TARCI-GP	100	2.4	-5.0%
TARCI-GP	30	0.74	-5.2%
TARCI-GP	10	0.30	-6.0%
TARCI-H	100	2.7	-5.9%
TARCI-H	30	0.77	-6.6%
TARCI-H	10	0.37	-7.6%
9-Node CGA	—	0.0	-6.5%
TARCI-D	—	0.064	-11.2%

is slightly worse than TARCI-GP. Finally, I note that looking at the post-recovery change in coverage, TARCI-H and TARCI-GP best 9-Node CGA, the estimate for the upper bound of the 2D recovery algorithms like DARA[1].

Change in coverage: Another view of post-recovery change in coverage is presented in Fig. 4.2 where I can see that nearly all of the TARCI algorithms (except for TARCI-D and TARCI-H with $H = 10$) have superior recovery responses. I am not surprised by this result because—similar to 2D algorithms DARA, PADRA, PCR, and NORAS—9-Node CGA never will direct a node into a previously unoccupied position, while the TARCI algorithms, without this restriction, can find positions with superior coverage. As noted in the previous section, the 2D algorithms that ignore coverage are expected to perform no better than 9-Node CGA.

Distance traveled during recovery: Fig 4.3 estimates energy usage during

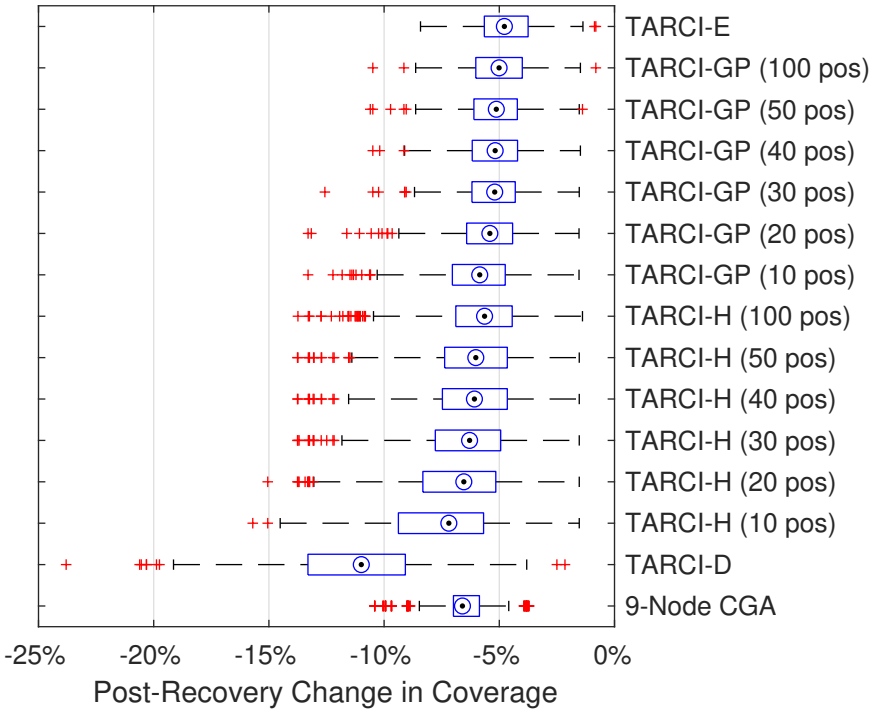


Figure 4.2: Change in coverage boxplot.

failure response from the distance that a backup node travels. Comparing TARCI-GP and TARCI-H, I see that TARCI-GP reduces coverage less than TARCI-H, but it also expends more recovery energy. The positive correlation between the goodness of the recovery response and the distance traveled can be explained by noting that TARCI-GP searches more widely, and so it is able to find a superior, more distant position. The datapoint for TARCI-D is consistent with this explanation, since TARCI-D always chooses the shortest possible recovery distance, and consequently it ignores any positions that have better coverage. Comparing datapoints of all of the TARCI variants, the positive correlation mentioned above between the average distance traveled and the average change

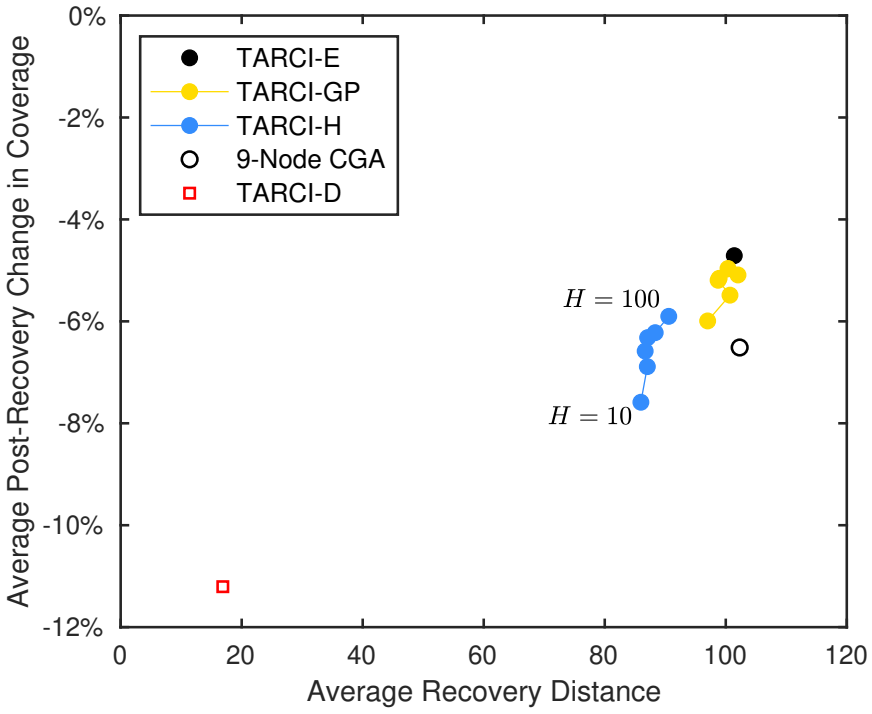


Figure 4.3: Change in coverage vs distance.

in post-recovery coverage is clear.

Finally, considering the sensitivity of TARCI-GP and TARCI-H to the value of H, Fig. 4.4 shows that TARCI-GP's average effectiveness is nearly as good as that of TARCI-E when TARCI-GP considers 30 positions. The graph also shows that TARCI-H must consider at least 40 positions in order to better the results of 9-Node CGA.

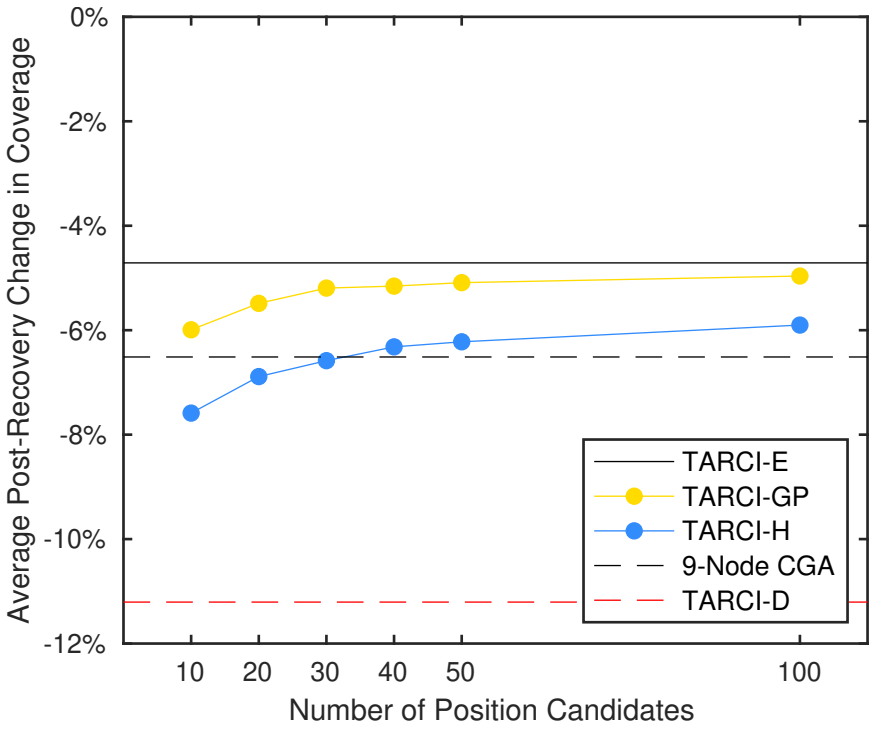


Figure 4.4: Change in coverage vs num positions.

Chapter 5

Prior Accomplishments

While at UCSC, I published addition research not directly related to this dissertation.

RTT Estimation via Machine Learning. In 2011, I contributed to using machine learning to control RTT estimation in TCP[41][42]. Specifically, I corroborated simulation results by implementing a proposed algorithm in the Linux kernel. This effort required converting a floating-point machine-learning algorithm to use fixed-point arithmetic.

Testbeds & Vehicle Tracking. During 2011 and 2012, I designed and installed the prototype for the BTS2 campus-bus tracking system. This effort involved creating designs for five different printed-circuit boards, assembling the prototypes, and helping manage the prototype's installation in a campus bus. It also involved switching the base station hardware away from Mini-ITX PCs to Raspberry Pi boards, rewriting the original BTS base station software, and helping manage the deployment of the new base-station

hardware on the roofs of five campus buildings. Finally, I wrote prototype web server and database software to display the bus tracking results. In 2013, I helped run experiments using the SCORPION testbed and then, based on that experience, I documented the testbed's management suite and published said documentation in IEEE conference proceedings[64]. In 2014, I managed undergraduate and grad students who were paid to assemble BTS2 production devices for all of the campus buses, and then I assisted with managing the BTS2 installation in all of the buses. In 2017, I (finally) wrote and submitted my Masters Thesis for BTS2[58].

Reproducibility. In 2019, I helped re-run some of my earlier Dissertation experiments as part of validating the Popper reproducibility tool[10].

Chapter 6

Conclusions

This chapter concludes the dissertation by first revisiting my contributions and then discussing directions for future work.

6.1 Contributions Revisited

In this dissertation I developed two novel terrain-aware algorithms for deploying outdoor IoT networks over 2.5D terrain. Grid Partition accounts for visibility over 2.5D terrain and finds solutions that compete with a centralized deployment algorithm. It also serves as a meta-heuristic. $\text{EMNA}_{\text{global,steps}}$finds solutions that compete with a centralized deployment algorithm, although it requires more calls to the utility function. I compared these algorithms to distributed versions of simulated annealing, gradient descent, and pattern search.

I developed and evaluated a class of terrain-aware hybrid network recovery algo-

rithms which guide networks in self-repair following a node failure. Terrain-Aware Recovery with Commshed Intersections (TARCI) consists of four algorithm variants. Experiments show that the variant that incorporates Grid Partition as a meta-heuristic (TARCI-GP) achieves nearly the same quality of results as a variant that includes an exhaustive step, although TARCI-GP uses two orders of magnitude less CPU time. I show that my terrain-aware recovery algorithms perform better than recovery algorithms that are intended for 2D deployments.

To help with the development of my terrain-aware algorithms, I created a terrain-aware experimental framework. The Terrain-Aware Framework For IoT (TAFFI) simulates Java- and C++-based algorithms, both distributed and centralized. The framework provides a library of line-of-sight algorithms and a suite of benchmark terrains. To ensure that the benchmarks include a wide range of landform variation, another contribution of this dissertation is the design of two terrain classifiers that ensure that the benchmark terrains used in the experiments represent a wide variety of terrain landforms.

6.2 Future Work

The work in this dissertation stimulates several ideas for future work.

Radio Propagation. First, I want to make clear the restrictions of this work's radio propagation modeling. The pragmatic choice of using CLOS for radio propagation could be addressed by incorporating Fresnel zones[18]. This change would require reconsidering

the Wang/Robinson/White dynamic programming CLOS visibility model. Probabilistic radio propagation models could address the effects of fading. We note that adding probabilistic propagation models would require changing the all algorithms to respond to actual connectivity.

We note that unlike the recovery algorithms proposed in this dissertation, the deployment algorithms use a simple disk model for connectivity, meaning that radio connections diffract over obstructions with the only limit being range. An obvious step is to modify the deployment algorithms to use the same radio propagation models that the recovery algorithms use.

Real deployments. Another area of inquiry is to compare the ideas of this work with a real-world deployment. Not only would such an exercise validate the suggested changes to radio-propagation modeling, but it also would lead to using real-world modeling of vegetation (trees) as visual obstructions, along with the necessary algorithm changes to handle observed connectivity.

Connectivity. There are two areas where non-propagation effects of connectivity can be explored. First, we can incorporate methods to account for network partitioning that might occur during the initial random placement. Second, Grid Partition—and the other deployment algorithms—could use multi-constraint optimization methods to maximize coverage while keeping the network connected.

Other Deployment Work. One can consider heterogeneous deployments where an algorithm saves resources by adding communications-only nodes (relay nodes). But

unlike 2D algorithms[28][31], one must design an algorithm that understands 2.5D terrain. Since most of the algorithms spend the much of their time computing visibility, one could evaluate the possibility of storing precomputed viewsheds and commsheds as a way of trading off computation for storage. Since we create recovery benchmark networks by decreasing radio power (converting bi-connected networks into networks with cut vertices), the inverse must be true: one should consider combining node mobility with increasing radio power to recover from a partitioned network.

Another area of inquiry is "temporary" deployments where "hopping" nodes sense and then return to the base.

Appendix A

Additional Plots

This appendix contains additional plots from 20-node and 30-node simulations that show nearly the same results as plots from 10-node simulations referenced in Section 2.1.3.

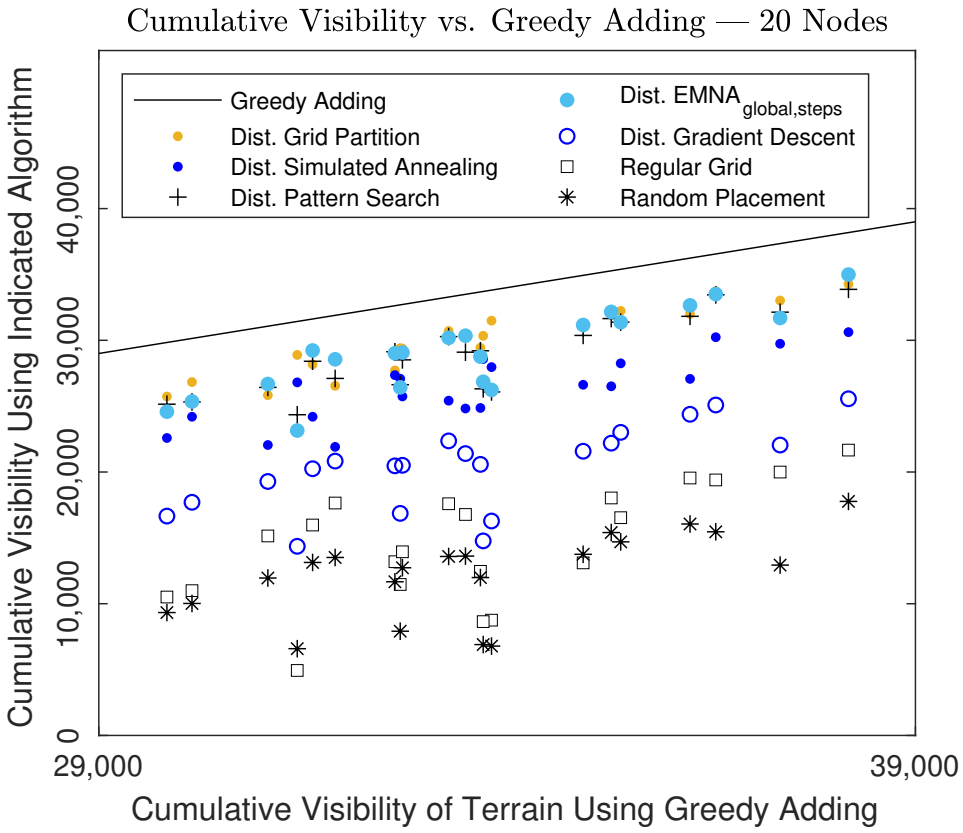


Figure A.1: Results of algorithms on test terrains vs. centralized Greedy Adding—20 nodes.

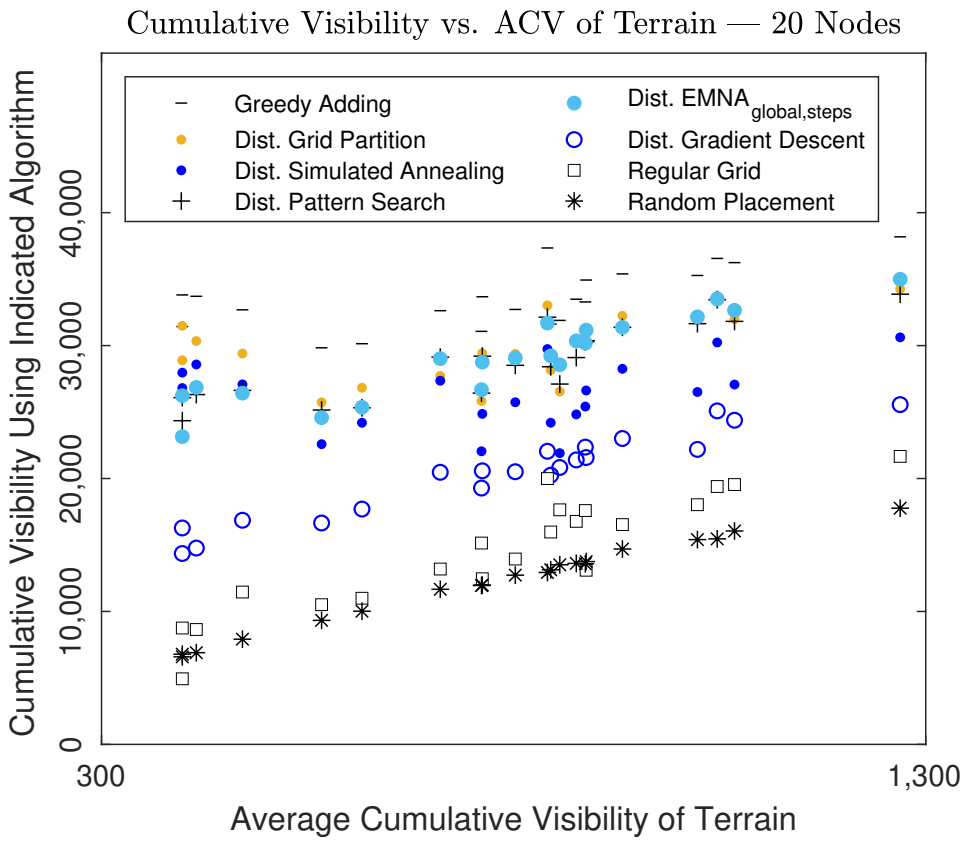


Figure A.2: Results of algorithms on test terrains vs. ACV of terrain—20 nodes.

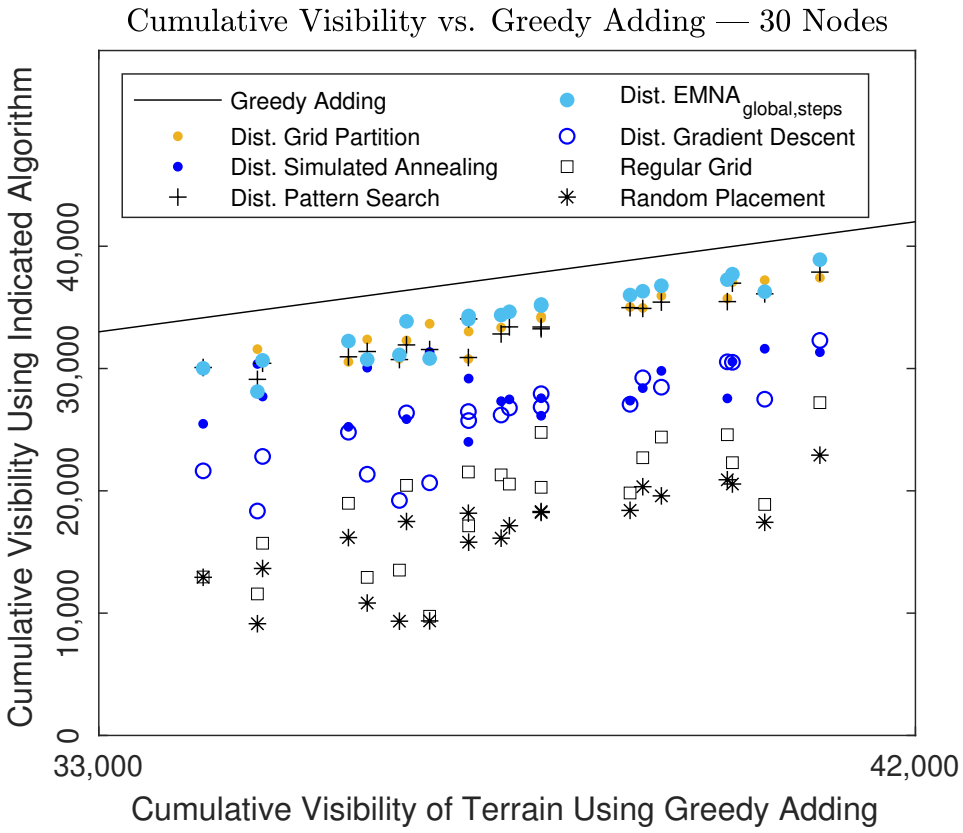


Figure A.3: Results of algorithms on test terrains vs. centralized Greedy Adding—30 nodes.

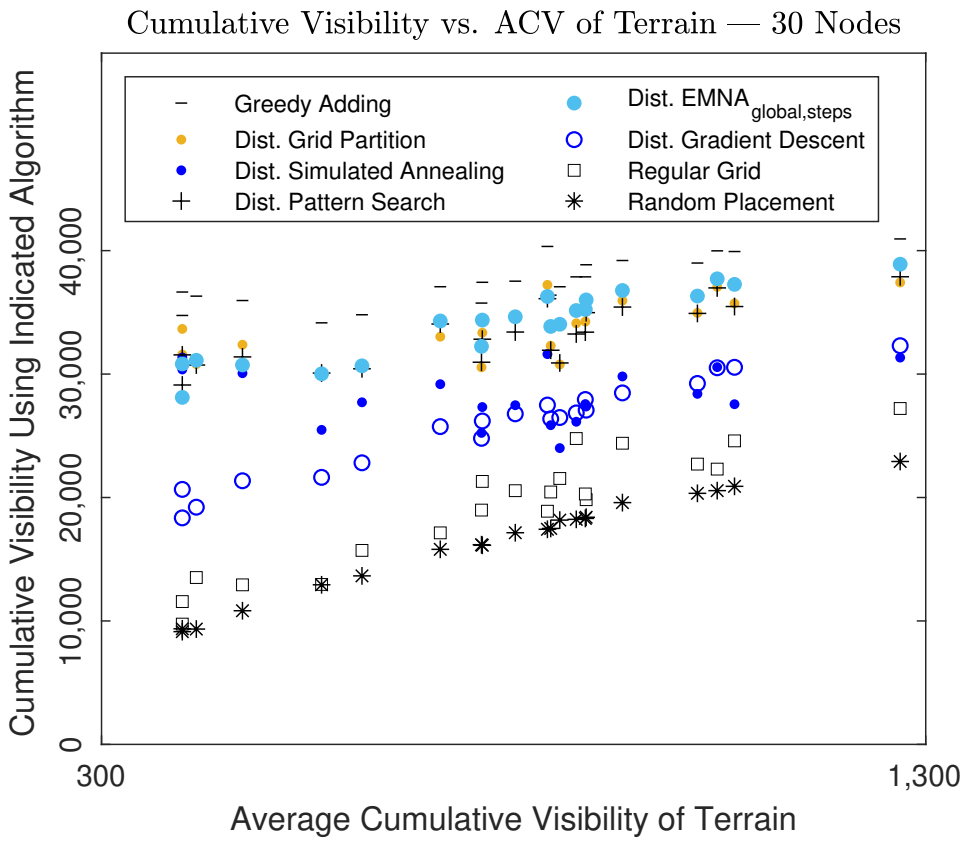


Figure A.4: Results of algorithms on test terrains vs. ACV of terrain—30 nodes.

Appendix B

Custom C++ Programs

During the course of this research, I implemented several Java classes and command-line C++ programs. The Java classes are documented in Section 2.2.1. This appendix describes the C++ programs.

Many of the programs share the same features, and many accept the same command-line options:

- They read or write terrain data files using the SRTM height-file (HGT) format (`-hgt`).
- They indicate the origin of the SRTM height file by the latitude and longitude of its southwest corner (`-hgt-latlon`).
- They create graphical representations of input and output data in image files that use the BMP format (`-bmp`). (You can convert such files into other image formats using the ImageMagick program `convert`.)

- They limit an experiment to a subtile of terrain by specifying a coordinate window or a two-letter subtile name (`-w` or `-subtile`).

- They specify a sensor radius and a communication radius (`-r` and `-cr`).

B.1 Generating Synthetic Terrains

Early in the research I explored using synthetic terrains, but I quickly moved to using actual STRM terrain data. Nonetheless, [59] includes some synthetic terrains, and so I document the terrain generators here.

Two programs generate wavy terrain. Over the 3600×3600 synthetic HGT tile, `mk_wavy_srtm` makes ten "bumps" vertically and 15 bumps horizontally, and `mk_wavy2_srtm` makes 20 bumps vertically and 30 bumps horizontally. (The artificial appearance of these arbitrary "landform" shapes led me to consider using real terrain data.) The program `mk_flat_srtm` generates a completely flat terrain at elevation 257 meters, the program `mk_dish_srtm_small` generates a "dish" centered on Cooja's window with elevation ranging from 100 meters in the middle to 1100 meters at the dish's edge, and `mk_slope` generates sloped terrain with elevation ranging from 100 meters in the west to 1100 meters in the east.

```
Usage: mk_wavy_strm
Usage: mk_wavy2_strm
Usage: mk_flat_strm
Usage: mk_dish_srtm_small
Usage: mk_slope
```

B.2 Centralized Deployment Algorithms

All of the centralized deployment algorithms are tested using the `visibility` program. (Some of the centralized simulations can be run using the Cooja-Contiki simulator, too.)

```
Usage: visibility [options]
Options:
    -hgt infile.hgt                         HGT input file
    -hgt-latlon latitude longitude          position of HGT file in decimal deg.
    -bmp outfile.bmp                        BMP output file
    -bmp-sequence basename                  sequence of BMPs
    -bmp-palette type                       0 = grayscale
                                            1 = heat map
                                            2 = blue/yellow
                                            3 = dark-blue/yellow,
                                            4 = dark-blue/light-yellow #1,
                                            5 = dark-blue/light-yellow #2,
                                            6 = black/light-yellow
                                            7 = black/yellow #1
                                            8 = black/yellow #2
                                            9 = black/white
    -bmp-grid-x x                           vertical gridlines every x pixels
    -bmp-grid-y y                           horizontal gridlines every y pixels
    -txt outfile.txt                        TXT output file
    -w lat_min lon_min lat_max lon_max      window in decimal degrees
    -wm x_min y_min x_max y_max             window in meters (unimplemented)
    -v latitude longitude                   viewpoint in decimal degrees
    -vm xMeters yMeters                     viewpoint in meters using Cooja
                                            coordinates, where the upper-left
                                            corner of the window is (0,0)
    -vrand num_rand_viewpoints              random viewpoints
    -vh viewpoint_height_above_terrain      same units as HGT file
    -r radius_max                           radius for cumulative visibility
    -s unsigned_seed                        init random number generator
    -V                                      compute visibility
    -C                                      compute cumulative visibility
    -C-hgt-out outfile.hgt                  with -C, optional HGT output file
    -C-min minimum                          with -C, optionally force c_min
    -C-max maximum                          with -C, optionally force c_max
    -W                                      compute WLU with -v, -vm, or -vrand
```

```
    -W-min minimum                      with -W, optionally force wlu_min
    -W-max maximum                      with -W, optionally force wlu_max
    -Wvp                                compute WLU for last viewpoint
    -H                                  write heights BMP
    -Hv                                 write heights BMP with visibility
Gradient Ascent options:
    -G method                           compute results of gradient ascent
    -v longitude latitude               viewpoint(s) in decimal degrees
    -t num_iterations                   number of iterations
    -delta pattern_search_delta         integer
Simulated Annealing options:
    -S                                  compute results of S.A.
    -v longitude latitude               viewpoint(s) in decimal degrees
    -T                                  initial temperature
    -M                                  initial iterations at temp
    -a                                  alpha
    -b                                  beta
    -l                                  lambda
    -d                                  dist_max
    -t                                  max_time
    -Sd                                 use distributed algorithm
    -er                                 exploration radius
    -cr                                 communication radius
Potential Games options:
    -P                                  compute results of P.G.
    -b                                  beta
    -d                                  dist_max
    -t                                  max_time
```

B.3 Generating Recovery Test Cases: CGA

Connected Greedy Adding (CGA) is the first of a pair of programs that together generate test cases for recovery algorithms. Given a fixed sink position, `cga` chooses initial positions for the nodes over terrain. The generated networks are guaranteed by construction to be connected, but in addition, `cga` tries to maximize coverage. The program creates text strings that you can use to call the `topology` program, which finalizes the generation of

the test cases by converting any bi-connected networks into networks that have at least one cut vertex.

```
Usage: cga [options]
Options:
    -hgt infile.hgt                         HGT input file
    -hgt-latlon latitude longitude          position of HGT file in decimal deg.
    -bmp-viewshed outfile.bmp               BMP viewshed output file
    -bmp-commshed outfile.bmp               BMP commshed output file
    -bmp-heights outfile.bmp                BMP heights output file
    -bmp-palette type                       0 = grayscale
                                            1 = heat map
                                            2 = blue/yellow
                                            3 = dark-blue/yellow,
                                            4 = dark-blue/light-yellow #1,
                                            5 = dark-blue/light-yellow #2,
                                            6 = black/light-yellow
                                            7 = black/yellow #1
                                            8 = black/yellow #2
                                            9 = black/white
    -bmp-grid-x x                           vertical gridlines every x pixels
    -bmp-grid-y y                           horizontal gridlines every y pixels
    -w lat_min lon_min lat_max lon_max      window in decimal degrees
    -subtile code                           two-letter subtile of window
    -sink index                             sink index (1 to 10)
    -tweak-sink                             move sink to nearby high point
    -m num_nodes                            number of nodes
    -vh viewpoint_height_above_terrain      same units as HGT file
    -r radius                               sensor radius
    -cr radius                              communications radius
```

B.4 Generating Recovery Test Cases: Topology

`topology` is the second of a pair of programs that together generate test cases for recovery algorithms. `topology` extracts a network topology from an indicated terrain subtile, a sink position, and a set of node positions. (The program `cga` generates command line strings that correctly invoke `topology`.) The program's output is an adjacency matrix

and a MATLAB .m file that will plot the generated test network. `topology` also is used to plot the Post-Recovery network using a command line string that is generated by `recover`.

```
Usage: topology [options]
Options:
    -hgt infile.hgt                         HGT input file
    -hgt-latlon latitude longitude          position of HGT file in decimal deg.
    -w lat_min lon_min lat_max lon_max      window in decimal degrees
    -subtile code                           two-letter subtile of window
    -v latitude longitude                   viewpoint in decimal degrees
    -vp xPixels yPixels                     viewpoint in pixels using SRTM
                                            coordinates, where the lower-left
                                            corner is (0,0) up to (3600,3600)
    -vm xMeters yMeters                     viewpoint in meters using Cooja
                                            coordinates, where the upper-left
                                            corner of the window is (0,0)
    -failed-node nodeid
    -recovery-node nodeid                   1 ... num nodes
    -recovery-vp xPixels yPixels            like -vp
    -objective                              objective code in base filename
    -vh viewpoint_height_above_terrain      same units as HGT file
    -cr range                               communications range in pixels
    -io                                     force intervisibility (or)
    -ia                                     force intervisibility (and)
```

B.5 Recovery Algorithm TARCI

`recover` is a program that simulates the Pre-Failure and Post-Recovery phases of the hybrid recovery algorithm TARCI and then reports the results. The program uses as input a MATLAB .m file that was generated by `topology`. From this file, `recover` extracts the communications radius and the (x, y) coordinates of the network's nodes, and then it searches for the best recovery node for each cut vertex of the network. The results of the simulations are printed to stdout, along with command line strings that will direct

`topology` to make a MATLAB plot file for the each result.

```
Usage: recover [options]
Options:
    -hgt infile.hgt                       HGT input file
    -hgt-latlon latitude longitude        position of HGT file in decimal deg.
    -m infile.m                           MATLAB input file
    -subtile code                         two-letter subtile of window
    -sink node                            sink id
    -r radius                             sensor radius
    -H num                                test the highest num positions
    -P percent                            test the highest % positions
    -grid-partition                       use grid partition
```

Appendix C

Perimeter and Area of Intersecting Circles

The Grid Partition algorithm chooses its grid size based on the perimeter p and the area A of a search region that is defined by the intersection of two circles. The shape of such an intersection depends on the circles' radii and their center-to-center separation. Let the radii of the circles be r and r_E, with $r \leq r_E$, and let the center-to-center separation of the circles be d. The equations for p and A depend on the relationship between the values of r, r_E, and d:

Case #1. If the larger circle completely encloses the smaller circle, then $d \leq r_E - r$. The perimeter and area of the intersection are the same as those of the smaller circle: $p = 2\pi r$ and $A = \pi r^2$.

Case #2. If the circles do not intersect, then $d \geq r_E + r$. Both the perimeter and area are zero: $p = 0$ and $A = 0$.

Case #3. If the circles overlap partially, then $r_E - r < d < r_E + r$. The perimeter

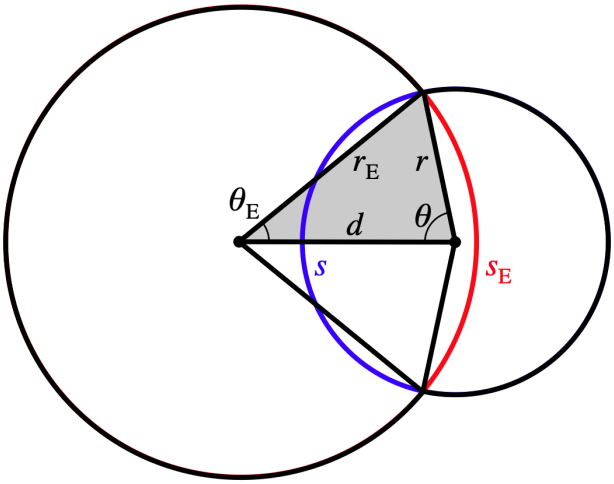


Figure C.1: The perimeter of a circle-circle intersection: $p = s + s_E$.

p, which is the sum of arc lengths s and s_E in Fig. C.1, can be computed straightforwardly. First, apply the Law of Cosines [68] to determine the angles θ and θ_E.

$$\theta = \cos^{-1}\left(\frac{d^2 + r^2 - r_E^2}{2dr}\right) \tag{C.1}$$

$$\theta_E = \cos^{-1}\left(\frac{d^2 + r_E^2 - r^2}{2dr_E}\right) \tag{C.2}$$

Then double each angle and multiply by its corresponding radius to obtain arc lengths.

$$s = 2r\theta$$

$$s_E = 2r_E\theta_E$$

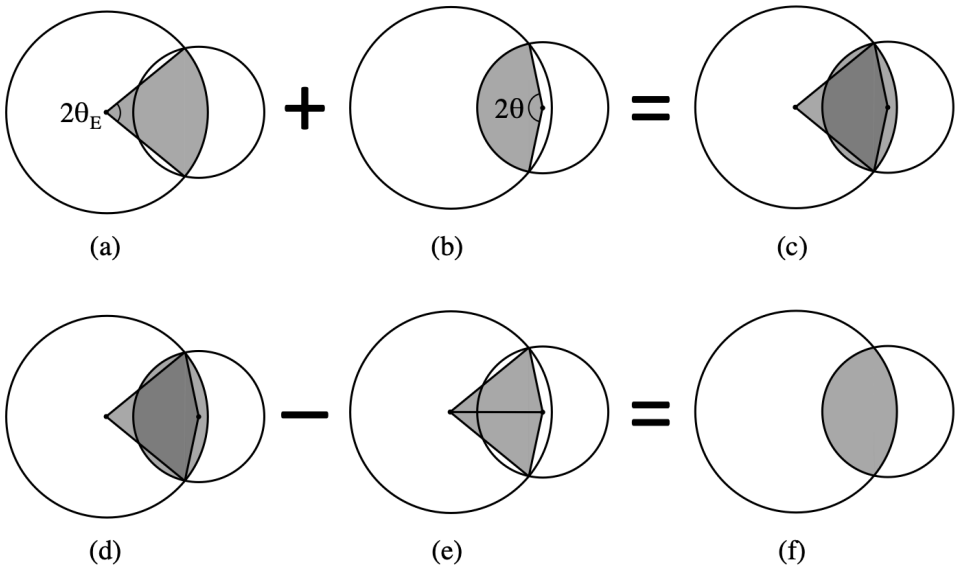


Figure C.2: Area arithmetic used for computing the area of a lens-shaped circle-circle intersection. The areas of the circular sectors in (a) and (b) are easy to compute and are summed into (c), but with the darker region double counted. We obtain the area of the lens-shaped region in (f) by subtracting the two triangular regions in (e).

The perimeter p is the sum of the two arc lengths.

$$
\begin{aligned}
p &= s + s_E \\
p &= 2r\theta + 2r_E\theta_E \\
p &= 2r\cos^{-1}\left(\frac{d^2 + r^2 - r_E^2}{2dr}\right) + 2r_E\cos^{-1}\left(\frac{d^2 + r_E^2 - r^2}{2dr_E}\right)
\end{aligned} \tag{C.3}
$$

We find the area A of the lens-shaped circle-circle intersection of Fig. C.2(f) through area arithmetic. First we add the areas of the two circular sectors of Fig. C.2(a) and Fig. C.2(b). The resulting sum in Fig. C.2(c) double counts the area of the overlapping region. Next, to eliminate the double counting, we subtract the area of the two triangles of Fig. C.2(e). This leaves the area of the circle-circle intersection in Fig. C.2(f), which can be

expressed algebraically as

$$A = A_{2\theta_E} + A_{2\theta} - 2A_{\text{triangle}} \tag{C.4}$$

To fully evaluate this equation, we observe that the area of a circular sector is the area of a full circle multiplied by the portion of the circle that the subtended angle represents:

$$A_{2\theta_E} = \pi r_E^2 \left(\frac{2\theta_E}{2\pi}\right) = r_E^2 \theta_E \tag{C.5}$$

$$A_{2\theta} = \pi r^2 \left(\frac{2\theta}{2\pi}\right) = r^2 \theta \tag{C.6}$$

To obtain the area of the triangle in Fig. C.1 with sides d, r, and r_E, we use Heron's Formula[67]:

$$A_{\text{triangle}} = \frac{1}{4}\sqrt{(d + r + r_E)(-d + r + r_E)(d - r + r_E)(d + r - r_E)} \tag{C.7}$$

Substituting (C.5), (C.6), and (C.7) into (C.4), the area of the circle-circle intersection is

$$A = r_E^2 \theta_E + r^2 \theta - \frac{1}{2}\sqrt{(d + r + r_E)(-d + r + r_E)(d - r + r_E)(d + r - r_E)} \tag{C.8}$$

Appendix D

Size of the Grid Partition

Given an exploration region with perimeter p from (C.3) and area A from (C.8), we want to know the grid size g that will partition the region into approximately n_0 squares. (We are content with an approximation because the Grid Partition algorithm is a heuristic, and an exact result is not necessary.) We can derive an equation for g by generalizing an estimate for the number of squares that are completely enclosed by a circular region.

The relationship between A, g, and n_0 for circular regions is identical to that investigated by the semiconductor manufacturing industry to estimate gross dice per wafer (GDW). We start with (3) in reference [12].

$$\mathrm{GDW} = \frac{\pi R_{\mathrm{eff}}^2}{A_{\mathrm{die}}} - \frac{c_1 R_{\mathrm{eff}}}{\sqrt{A_{\mathrm{die}}}} \tag{D.1}$$

We substitute our own variables in (D.1) using $\mathrm{GDW} \approx n_0$, $R_{\mathrm{eff}} = r_E$ and $A_{\mathrm{die}} = g^2$. For

reasons that will become clear, we also switch the proportionality constant using $c_1 = 2\pi k$:

$$n_0 \approx \frac{\pi r^2}{g^2} - \frac{2\pi k r}{g} \tag{D.2}$$

Examining (D.2), one understands that it estimates the number of fully encompassed squares n_0 by starting with the ratio between the area of a circular region πr^2 and the area of a square g^2. Since this area ratio overestimates the number of fully encompassed squares by including the contributions of partially encompassed squares, the equation applies a correction term. The correction term is the ratio between the perimeter $2\pi r$ of the circular region and the edge length g of a square. One can see that this ratio will be roughly proportional to the number of squares that the perimeter crosses. The correction term of (D.2) is this ratio adjusted by a constant value k.

We generalize (D.2) for non-circular regions by replacing πr^2, the area of a circle, with A, and by replacing $2\pi r$, the perimeter of a circle, with p. The result is

$$n_0 \approx \frac{A}{g^2} - \frac{kp}{g} \tag{D.3}$$

Multiplying both sides by g^2 and solving the resulting quadratic equation for g, we obtain an estimate for the grid size from the area A and perimeter p of a region:

$$g \approx \frac{\sqrt{4An_0 + k^2p^2} - kp}{2n_0} \tag{D.4}$$

Empirically we've determined a good value for the proportionality constant k.

$$k = 0.6 \tag{D.5}$$

As a check, we can compute a value for k from the constant $c_1 = 1.16\pi$ of [12] and our substitution $c_1 = 2\pi k$.

$$c_1 = 1.16\pi$$

$$2\pi k = 1.16\pi$$

$$k = \frac{1.16\pi}{2\pi}$$

$$k = 0.58$$

This result verifies our value $k = 0.6$, as it is nearly identical.

Bibliography

[1] Ameer Ahmed Abbasi, Kemal Akkaya, and Mohamed Younis. A distributed connectivity restoration algorithm in wireless sensor and actor networks. In *32nd IEEE Conference on Local Computer Networks (LCN 2007)*, pages 496–503, New York, 2007. IEEE.

[2] Vahab Akbarzadeh, Christian Gagné, Marc Parizeau, Meysam Argany, and Mir Abolfazl Mostafiva. Probabilistic sensing model for sensor placement optimization based on line-of-sight coverage. *Transactions on Instrumentation and Measurement*, 62(2):293–303, February 2013.

[3] Kemal Akkaya, Aravind Thimmapuram, Fatih Senel, and Suleyman Uludag. Distributed recovery of actor failures in wireless sensor and actor networks. *2008 IEEE Wireless Communications and Networking Conference*, pages 2480–2485, 2008.

[4] P. Basu and J. Redi. Movement control algorithms for realization of fault-tolerant ad hoc robot networks. *IEEE Network*, 18(4):36–44, 2004.

[5] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization:

Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, 01 2001.

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

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

[8] Richard Church and Charles ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32:101–118, 12 1974.

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

[10] Andrea David, Mariette Souppe, Ivo Jimenez, Katia Obraczka, Sam Mansfield, Kerry Veenstra, and Carlos Maltzahn. Reproducible computer network experiments: A case study using popper. In *Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems*, 2019.

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

[12] Dirk K. de Vries. Investigation of gross die per wafer formulas. *IEEE Transactions on Semiconductor Manufacturing*, 18(1):136–139, February 2005.

[13] H.M. Dodd. The validity of using a geographic information system's viewshed function

as a predictor for the reception of line-of-sight radio waves. Master's thesis, Virginia Tech, Blacksburg, VA, USA, August 2001.

[14] Marco Dorigo and Thomas Stützle. Ant colony optimization: Overview and recent advances. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*. Springer International Publishing, Cham, Switzerland, 3rd edition, 2019.

[15] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, and Daniel Weiskopf. *Real-Time Volume Graphics*. A. K. Peters, Ltd., Wellessley, Massachusetts, 2006.

[16] Tom G. Farr, Paul A. Rosen, Edward Caro, Robert Crippen, Riley Duren, Scott Hensley, Michael Kobrick, Mimi Paller, Ernesto Rodriguez, Ladislav Roth, David Seal, Scott Shaffer, Joanne Shimada, Jeffrey Umland, Marian Werner, Michael Oskin, Douglas Burbank, and Douglas Alsdorf. The shuttle radar topography mission. *Reviews of Geophysics*, 45(2):1–33, 2007.

[17] Leila De Floriani and Paola Magillo. Algorithms for visibility computation on digital terrain models. In *SAC*, pages 380–387, 1993.

[18] Roger L. Freeman. *Line-of-Sight Microwave Radiolinks*, chapter 2, pages 37–131. John Wiley & Sons, Ltd, 2007.

[19] Jean-loup Gailly. GNU Gzip. `https://www.gnu.org/software/gzip/manual`, 2021.

[20] Rob Gordon. *Essential JNI: Java Native Interface*. Prentice Hall, Englewood Cliffs, New Jersey, 1998.

[21] Nikolaus Hansen. The cma evolution strategy: A comparing review. In Jose A. Lozano, Pedro Larrañaga, Iñaki Inza, and Endika Bengoetxea, editors, *Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms*, pages 75–102. Springer-Verlag, Berlin, 2006.

[22] Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '10, page 1689–1696, New York, NY, USA, 2010. Association for Computing Machinery.

[23] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *JACM*, 8(2):212–229, 1961.

[24] Muhammad Imran, Mohamed F. Younis, Abas B. Md Said, and Halabi Hasbullah. Partitioning detection and connectivity restoration algorithm for wireless sensor and actor networks. *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 200–207, 2010.

[25] Samuel L. S. Jacoby, Janusz S. Kowalik, and J. T. Pizzo. *Iterative Methods for*

Nonlinear Optimization Problems. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972.

[26] Chih-Wei Kang and Jian-Hung Chen. Multi-objective evolutionary optimization of 3d differentiated sensor network deployment. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO'09)*, pages 2059–2064, New York, 01 2009. Association for Computing Machinery.

[27] Donald Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Boston, Massachusetts, 2nd edition, 1981.

[28] Gaurav Kumar and Virender Ranga. Healing partitioned wireless sensor networks. In Sergio F. Ochoa, Pritpal Singh, and José Bravo, editors, *Ubiquitous Computing and Ambient Intelligence*, pages 545–557, Cham, 2017. Springer International Publishing.

[29] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley Pub. Co, Boston, Massachusetts, 1986.

[30] Pedro Larrañaga. A review on estimation of distribution algorithms. In Pedro Larrañaga and Jose Antonio Lozano, editors, *Estimation of distribution algorithms: a new tool for evolutionary computation*, pages 57–100. Kluwer Academic Publishers, Boston, Massachusetts, 2002.

[31] Sookyoung Lee, Mohamed Younis, and Meejeong Lee. Connectivity restoration in a

partitioned wireless sensor network with assured fault tolerance. *Ad Hoc Networks*, 24:1–19, 2015.

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

[33] Xiaodong Li and Maurice Clerc. Swarm intelligence. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*. Springer International Publishing, Cham, Switzerland, 3rd edition, 2019.

[34] Frank Yeong-Sung Lin and Pei-Ling Chiu. A simulated annealing algorithm for energy-efficient sensor network design. In Eitan Altman and Holger Karl, editors, *3rd International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt 2005), 4-6 April 2005, Trentino, Italy*, pages 183–189, Piscataway, New Jersey, 2005. IEEE Computer Society.

[35] Pin Lv, Jin-fang Zhang, and Min Lu. An optimal method for multiple observers sitting on terrain based on improved simulated annealing techniques. In Moonis Ali and Richard Dapoigny, editors, *Advances in Applied Artificial Intelligence*, pages 373–382, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[36] R.A. MacMillan and P.A. Shary. Landform and landform elements in geomorphometry. In T. Hengl and H.I. Reuter, editors, *Geomorphometry: Concepts, Software, Applica-*

tions, volume 33 of *Developments in Soil Science*, chapter 9, pages 227–254. Elsevier, Amsterdam, 2009.

[37] Sam Mansfield, Kerry Veenstra, and Katia Obraczka. Terrainlos: An outdoor propagation model for realistic sensor network simulation. In *International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 463–468, New York, 09 2016. IEEE.

[38] Guillermo Molina and Enrique Alba. Wireless sensor network deployment using a memetic simulated annealing. In *International Symposium on Applications and the Internet*, pages 237–240, Piscataway, New Jersey, August 2008. IEEE.

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

[40] Jorge Nocedal and Stephen J. Wright. Derivative-free optimization. In Thomas V. Mikosch, Sidney I. Resnick, and Stephen M. Robinson, editors, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, 2nd edition, 2006.

[41] Bruno Astuto A. Nunes, Kerry Veenstra, William Ballenthin, Stephanie M. Lukin, and Katia Obraczka. A machine learning approach to end-to-end rtt estimation and its application to tcp. In *20th IEEE International Conference on Computer Communications and Networks (ICCCN'11)*, pages 1–6, July 2011.

[42] Bruno Astuto A. Nunes, Kerry Veenstra, William Ballenthin, Stephanie M. Lukin, and Katia Obraczka. A machine learning framework for tcp round-trip time estimation. *EURASIP Journal on Wireless Communications and Networking*, 2014:1–22, 2014.

[43] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 3 of *International Series of Monographs on Computer Science*. Oxford University Press, New York, 1987.

[44] Carlos Domingo Ortiz, Joaquín Mares Puig, Carlos E. Palau, and Manuel Esteve. 3d wireless sensor network modeling and simulation. In *SensorComm 2007: International Conference on Sensor Technologies and Applications*, pages 307–312, Los Alamitos, CA, October 2007. IARIA, IEEE Computer Society.

[45] Fredrik Österlind. A sensor network simulator for the contiki os. Technical Report T2006:05, Swedish Institute of Computer Science, 2006.

[46] Thomas K. Peucker and David 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.

[47] Thomas K. Peucker, Robert J. Fowler, James J. Little, and David M. Mark. The triangulated irregular network. In *Proceedings, American Society of Photogrammetry: Digital Terrain Models (DTM) Symposium*, pages 516–540, Bethesda, Maryland, May 1978. American Society for Photogrammetry & Remote Sensing, American Society of Photogrammetry.

[48] Chet Ramey and Brian Fox. The GNU Bash Reference Manual. https://www.gnu.org/software/bash/manual, 2020.

[49] Arnold D. Robbins. Gawk: Effective AWK Programming. https://www.gnu.org/software/gawk/manual, 2020.

[50] 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, Stevenage, England, 2013.

[51] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms with Applications in Engineering.* IEEE Computer Society, Los Alamitos, CA, 1999.

[52] Mustapha Senouci and Abdelhamid Mellouk. *Deploying Wireless Sensor Networks: Theory and Practice.* ISTE Press Ltd, London, 2016.

[53] Richard M. Stallman, Roland McGrath, and Paul D. Smith. The GNU Make Manual. https://www.gnu.org/software/make/manual, 2020.

[54] The MathWorks, Inc. MATLAB. https://www.mathworks.com/products/matlab.html, 2021. [Online; accessed 24-Aug-2021].

[55] Constantine Toregas and Charles Revelle. Binary logic solutions to a class of location problem. *Geographical Analysis*, 5:145 – 155, 1973.

[56] Costis Toregas, R. Swain, C. ReVelle, and L. Bergman. The location of emergency services facilities. *Operations Research*, 19:1363–1373, 01 1971.

[57] Ketaki Vaidya and Mohamed F. Younis. Efficient failure recovery in wireless sensor networks through active, spare designation. *Proc. of the 1st Int'l Workshop on Interconnections of Wireless Sensor Networks (IWSN'10)*, 2010.

[58] Kerry Veenstra. Bts 2: A robust, low-cost, real-time bus tracking system. Master's thesis, University of California, Santa Cruz, CA, USA, March 2017.

[59] Kerry Veenstra and Katia Obraczka. Guiding Sensor-Node Deployment Over 2.5D Terrain. In *Proceedings of IEEE International Conference on Communications*, pages 6719–6725, New York, June 2015. IEEE.

[60] Kerry Veenstra and Katia Obraczka. Grid partition: an efficient greedy approach for outdoor camera IoT deployments in 2.5D terrain. In *29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, New York, 2020. IEEE.

[61] Kerry Veenstra and Katia Obraczka. Apparatus and method for efficient deployment of nodes in a network. Provisional patent application, 2021.

[62] Kerry Veenstra and Katia Obraczka. TAFFI: A distributed deployment framework for IoT networks over 2.5D terrain. Unpublished, 2022.

[63] Kerry Veenstra and Katia Obraczka. TARCI: Recovery for outdoor IoT networks over 2.5D terrain. Unpublished, 2022.

[64] Kerry Veenstra, Katia Obraczka, Wade Gobel, Daniel Olivares, and Vladislav Petkov. A management suite for a disruption-tolerant wireless network testbed. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 844–847, 2013.

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

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

[67] Heron's formula. `https://en.wikipedia.org/wiki/Heron%27s_formula`.

[68] Law of cosines. `https://en.wikipedia.org/wiki/Law_of_cosines`.

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

[70] Thomas H. Wonnacott and Ronald J. Wonnacott. *Introductory Statistics for Business and Economics*. John Wiley & Sons, Inc., Hoboken, New Jersey, 2nd edition, 1977.

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

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

[73] M. Younis, S. Lee, I.F. Senturk, and K. Akkaya. Topology management techniques for tolerating node failure. In Habib M. Ammari, editor, *The Art of Wireless Sensor Networks, Volume 1: Fundamentals*, pages 273–311. Springer, Berlin, 2014.

[74] M. Younis, I. Sentruk, K. Akkaya, S. Lee, and F. Senel. Topology management techniques for tolerating node failures in wireless sensor networks: a survey. *Computer Networks*, 58:254–283, 2014.

[75] Mohamed Younis, Sookyoung Lee, and Ameer Abbasi. A localized algorithm for restoring internode connectivity in networks of moveable sensors. *IEEE Trans. Computers*, 59:1669–1682, 12 2010.

[76] Mostefa Zafer, Mustapha Reda Senouci, and Mohamed Aissani. Terrain partitioning based approach for realistic deployment of wireless sensor networks. In Abdelmalek Amine, Malek Mouhoub, Otmane Ait Mohamed, and Bachir Djebbar, editors, *Computational Intelligence and Its Applications*, pages 423–435, New York, 2018. Springer.