Online Path Planning for Multi-Robot Coverage

Kevin Bradner

Department of Electrical Engineering and Computer Science

Case Western Reserve University

January, 2020

Contents

List of Tables

List of Figures

Acknowledgements

Abstract

Robotic coverage problems task one or more robots with the goal of visiting every location in a region. Algorithms which can perform this kind of task in a time efficient manner are useful for purposes such as mapping, cleaning, or inspection. This work considers a multi agent robotic coverage problem in which the shape of the region to be explored is known in advance, but additional information and challenges are discovered during task performance. A software package is created to simulate such a scenario, generate virtual environments to be covered, and interface with policy programs that command the robots. Offline path planning algorithms are developed, and their performance on this task is evaluated. Next, online variants of these algorithms are developed to respond to events and information encountered during task execution. It is shown that the online algorithms are more robust and better performing than their offline counterparts.

Chapter 1

Introduction

1.1 Robotic Coverage Background

Robotic motion planning problems are typically concerned with finding an achievable path between two robot configurations. These configurations could be the current location and desired destination of a robotic vehicle, the current pose of a robotic arm and the pose required for it to grasp an object, or any other pair of robot states which correspond to "start" and "goal" configurations. *Robotic coverage* problems are another kind of motion planning problem. The goal of a robotic coverage problem is to find a path for the robot to follow, during which it will visit every location in a region. This region could be an area of floor to be vacuumed, the surface of a car body that needs to be painted, or the interior of a building that needs to be mapped. In all of these cases, the task is complete only once the entire region has been visited, or *covered* by a robot. Usually, another goal of these problems is to achieve coverage in a way that minimizes the time or number of movements required to complete the task. The robotic coverage problem can be viewed as a continuous generalization of the travelling salesman problem [7]. Thus, any sufficiently general instance of the problem is NP-Hard.

One particularly well studied type of robotic coverage problem is the case where the coverage region is some subset of the plane, and the robots are mobile robots that can only move in the plane. A good overview of this kind of coverage problem can be found in [1]. Algorithms to perform robot coverage in the plane have been studied since the late 1980's. Early algorithms for this purpose relied on the use of heuristics, and these algorithms usually could not guarantee that complete coverage would always be achieved. By around 2000, planar coverage algorithms were often able to guarantee complete coverage. A completeness guarantee is often proved by showing that the algorithm breaks down the coverage space into smaller regions that are relatively trivial to cover completely. Coverage algorithms that employ this kind of technique are known as *cellular decomposition* algorithms.

One type of cellular decomposition fits a square grid to the coverage space. Because a general arbitrary set in the plane can not be exactly expressed as a union of grid-aligned square regions, this technique is often called approximate cellular decomposition. Typically, each cell in this kind of decomposition can be covered entirely by a single robot configuration. For example, consider dividing a grass lawn into squares for a robotic lawn mowing task. If each square in the cellular decomposition is small enough, a lawnmower whose center is aligned with a square's center will have mowed that entire square of the lawn. In coverage problems where this assumption applies, proving that a coverage path visits each cell in the approximate decomposition is sufficient to guarantee complete coverage.

In addition to approximate cellular decomposition, there are also techniques that decompose a coverage space into a more diverse collection of cell shapes. Examples of such shapes include cells of fixed with whose top and bottom boundaries can vary in shape [3], or trapezoidal cells whose parallel sides all run in the same direction [4]. Depending on whether the union of these cells is exactly equal to the coverage space, such techniques are called either *semi-approximate* or *exact*

cell decompositions. In the case of trapezoidal cell decomposition, it is possible to combine the trapezoidal cells into a smaller number of large cells, each of which are able to be covered by a simple back-and-forth sweeping motion. This approach is known as a *boustrophedon* decomposition [4].

1.2 Spanning-Tree based Coverage Algorithms

When coverage is performed by a square shaped tool attached to a mobile robot, and when the coverage area can be exactly decomposed into a grid of squares which have double the side length of the robot tool, it is possible to compute an optimal covering path in linear time using an approach based on spanning trees [2]. The authors developed several variants of this approach with different time, memory, and prior knowledge requirements. In all cases, it was assumed that a mobile robot would complete the coverage task with a square shaped tool of size D. It was also assumed that the tool could only move in the four cardinal directions when completing the task. Finally, it was assumed that the region to be covered could be approximated by cells in a grid of squares with size $2D$. The authors argue that the final assumption is justified for realistic environments in which the size of the robots tool is significantly smaller than the dimensions of the area to be covered. Under these assumptions, the *Spanning Tree Covering* (STC) algorithm is able to generate a path through the space which visits each grid-aligned square cell of size D exactly once.

The offline variant of STC works by representing the coverage area as a graph G. Nodes on this graph represent the center of a square with size $2D$, and the set of nodes corresponds to the set of full cells when a grid with square size $2D$ is overlaid on the coverage space. Edges exist between any two nodes whose corresponding cells are adjacent in the coverage space. As the name suggests, STC's key step is

the creation of a spanning tree on G, using any node in G as the root of the tree. Many algorithms exist to compute spanning trees, but two popular examples are closely based on breadth-first-search and depth-first-search [9].

Once this spanning tree has been created, the coverage area is subdivided into squares of size D, such that each node of G coincides with the corners of four different squares in the subdivided map. Starting at any one of these squares, each of which is exactly the shape and size of the robot's coverage tool, the robot can simply move counterclockwise around the spanning tree as if it were performing wall following. In this way, each cell in the coverage space will be visited exactly once by the robot. Because of this property along with the fact that the path generated by STC end up adjacent to where it started, the path found by STC can be considered a Hamiltonian cycle on G. Although constructing a Hamiltonian path on a general planar grid is NP-complete [10], the assumed properties of G make it practical to compute such a path on this graph in linear time. It is also worth noting that this coverage path's start and end positions will be in adjacent cells. This can be beneficial in practice, as it allows deployment and retrieval of the robot to happen in the same place.

This algorithm has a few interesting variants. The first runs online, and effectively circumnavigates a depth first search spanning tree that it constructs on the fly [2]. Restriction to use of depth first search is the primary disadvantage of this approach compared to offline STC. While the coverage paths based on depth first search are equivalent to those created through any other method in terms of final path length, other spanning tree algorithms have practical benefits. For example, spanning tree algorithms such as Prim's algorithm or Kruskal's algorithm can create minimal spanning trees on weighted graphs, and this can be exploted to create coverage paths that minimze turning. In later work, the authors of STC do exactly that in an algorithm called Scan-STC, which also makes additional improvements on

the original STC as discussed below. For details on Prim's algorithm and Kruskal's algorithm, refer to [9].

The second online variant of the STC algorithm is described by its authors as *ant-like*. In this case, the robot must be able to mark visited sub-cells as visited through some means such as leaving pebbles on the ground. In addition, the robot must be able to sense the presence of these makers in the nodes of G adjacent to its current position. By marking each visited sub cell as the robot leaves that cell, the robot is able to proceed on the same path it would have created in the depth first search based online approach. Because this algorithm uses markers to identify visited nodes, it requires only O(1) memory. However, it requires the use of O(N) markers, where N is the number of sub-cells in the coverage area. Nonetheless, this algorithm presents interesting ideas for multi-agent approaches and prevention of drift in pose estimation. In both of these cases, physical and immobile records of a robot's past location can inform the decision making of any robot currently near some of those records.

Finally, there is a variant of STC that achieves *exact* coverage of an environment, even when that environment can not be represented by completely filled cells on a grid of squares with size $2D$. However, this approach still requires that the connectivity of the environment is represented adequately by such a grid. The resulting algorithm simply notices any cells that are partially occupied by a previously unseen obstacle when it passes next to them in the course of normal online STC. When such a cell is encountered, a neighborhood is swept around the entire obstacle. Then, the robot returns to traversing the fully in-bounds cells according to its usual procedure. For details concerning the size of this neighborhood and the correctness of the resulting algorithm, see [2].

In the previously mentioned Scan-STC algorithm, STC is improved to explicitly handle boundary cells, or graph cells that are partially occupied by an obstacle.

Using the same notation as before, this algorithm generates a path with length no greater than $(n+m)D$, where n is the number of nodes in G, and m is the number of boundary cells [8]. The authors assume that no node will contain an arrangement of obscales such that the graph of that node's subnodes is disconnected. If an obstacle exists such that the normal grid of nodes would violate this assumption, the corresponding location in the coverage area is assigned two different nodes. With that assumption in place, avoidance of obstacles is done by simply displacing the normal path that goes around the obstacle. This displacement tactic is what leads to the m term appearing in the maximum path length bound. The authors claim that in practical environments, the path length is closer to $n + \frac{m}{2}$.

The other significant insight of Scan-STC relates to its name. By allowing the mobile robot to sense the state of the eight cells the surround its current position, it is possible to discover that horizontal neighbors of the current cell has a free vertical neighbor, and that the current cell has a free vertical neighbor in the same direction. In this case, it is guaranteed that a path into the nodes horizontal neighbor exists such that the vertical neighbors of these cells appear between the current cell and its horizontal neighbor. With this insight, it is possible to skip the creation of a horizontal edge between the current node and its horizontal neighbor. The benefit of this approach is that it leads to coverage paths that move primarily in the vertical directions. This can be useful for limiting the amount of turns performed, thereby reducing the number of opporuntities to introduce dead reckoning error from imprecise turns.

1.3 Muti-Agent Robotic Coverage

Another imporant feature of a robotic coverage algorithm is whether that algorithm controls a single robot or a group. Algorithms that control a group of robots for this

purpose are known as multi-agent coverage algorithms. Multi agent approaches to coverage can often complete a coverage task several times faster than a single agent approach because of the division of work that use of multiple robots enables. Use of multiple robots can enhance robotic coverage by allowing for more precise localization through information sharing. Finally, multi-agent methods are often able to adapt to the failure of one or more robots, as there may be remaining robots capable of completing the coverage task [1]. However, multi-agent techniques often require more complex motion planning strategies to coordinate the motion of multiple robots in a way that takes full advantage of these potential efficiency improvements and other benefits.

It is also possible to adapt a spanning tree coverage algorithm to the multi-agent case [6][11]. In their work on this subject, Noam Hazon and Gal Kaminka generalize the STC algorithm to the multi agent case, resulting in an algorithm called *Multirobot Spanning-Tree Coverage*, or MSTC. The MSTC problem assumes that the coverage area can be adequately approximated by a grid of square cells with side length $2D$, as in the spanning tree coverage algorithm. As with the original version of STC, it is assumed that all cells in the coverage space are free of obstacles. It is also assumed that all robots have the same speed and tool size.

With these assumptions in place, the first version of MSTC simply constructs a spanning tree of the space offline, then has all of the drones start to follow the spanning tree from their initial positions, much the same as single agent STC. If the robots are able to share information about their status, i.e. working or broken, and whether they have completed their assigned section, then it is possible to have all working robots continue to follow the spanning tree path until the coverage task is complete. Under these assumptions, complete coverage is guaranteed as long as one robot remains in working order until the end of the task.

If there are k robots exploring a coverage space with n sub-cells as described

above, then the best case time for this algorithm is shown to be $\frac{n}{k} - 1$. This performance is achieved when the starting points of the robots are distributed such that each robot is evenly spaced along the length of the spanning tree coverage path. The worst case is stated to be $n-k-1$, for the case where all k robots start on k adjacent subcells that appear together in the spanning tree graph. Implicit in both bounds is the assumption that no robots fail. In a pathological case with the same initial conditions used to give the previously stated worst case bound, the $k-1$ robots that start directly in front of the last robot could all fail at time $n-k-2$, just before the front robot reached the back robot's position. In this case, it would take another $n-1$ steps for the remaining robot to complete the coverage task according to Algorithm 2 as described by the authors, leading to a total coverage time of $2n-k-2$. The authors note that many practical multi-robot coverage scenarios require starting all robots from the same location, meaning that performance close to the worst case bound $n-k-1$ is relatively common. This motivates the development of a more sophisticated coverage algorithm. By assuming that backtracking is allowed, the authors develop an algorithm with a worst case of $\frac{n}{2} - 1$ for $k > 2$. However, because all algorithms in [11] only consider the possibility of moving foreward and backward along the initially calculated spanning tree path, many opporunities for further improvement are missed.

Work by Zheng et al. surpasses the average-case performance of MSTC with their *Multi-Robot Forest Coverage* (MFC) algorithm [6]. As the name suggests, MFC computes a separate spanning tree for each of the robots working on the coverage task. For convenience, the authors of MFC also assume that multiple robots can occupy the same subcell simultaneously, and that the coverage area can be represented by grid-aligned cells with side length twice that of the robot's coverage tool. Experimental results showed that MFC achieved performance close to a theoretical lower bound in many cases. In addition, the authors of [6] were able to

prove that their algorithm's cover times are no worse than eight times the optimal value, thanks to properties of the tree cover algorithm for general graphs which they adapted to this purpose.

1.4 Problem Statement

As with much of the previously discussed work, this work presents multi-agent coverage algorithms based on spanning trees. The details of the coverage problem stated here vary somewhat from other work - these differences are motivated by a use case in which a team of aerial drones (UAVs, robots) must perform sensor coverage of an area affected by disaster. Two particularly novel features are introduced through this consideration: robots that are able to change the size of their field of view, and environments with subsections which require different levels of scrutiny. These two changes are closely related to one another.

Concrete motivation for an environment that requires varying levels of scrutiny is based on the idea that not all areas are worth viewing in great detail when gathering information for a search and rescue effort. For example, some areas of the coverage environment may be covered in water or similarly uniform. Such areas may not be worth covering in detail because their uniform nature makes it easy to conclude that nothing of interest is happening in these regions. Oppositely, trying to observe a region with dense tree cover may not be practical in a coverage scenario. In contrast, areas with partial occlusions, areas with visually complex damaged structures, or places where people are particularly likely to be in need of help may merit closer looks. Another assumption of this problem is that the distribution of locations requiring one level of scrutiny or another is not known in advance of coverage. It may not be possible to know the exact location of collapsed structures, flooded regions, or other environment features that may come up with relatively

little warning in an emergency.

The other significant novel feature of this problem formulation, robots with a variable field of view, arises naturally in light of the above consideration. Real drones can usually alter the size of a ground area being photographed. Increasing the size of the observation area may be achieved by ascending with a constant viewing angle or by using a camera zoom feature to increase the viewing angle. Likewise, decreasing the field of view may be achieved by descending, zooming in, or by some combination of these. Increasing the field of view has the obvious benefit of allowing a robot to cover more of its environment at once. Decreasing the field of view increases the detail with which a drone observes the area directly below it, and this allows high detail viewing of those areas that require it. While the distribution of environment sections that require low vs high scrutiny is unknown a priori, it is assumed that the boundaries of the coverage area and of the obstacles that border it are known in advance.

Unlike most other work on spanning-tree based coverage, this work assumes that the robots can move along diagonals, rather than only in the four directions orthogonal to the sides of its tool. For convenience of notation, the four directions commonly used by spanning tree robots will be referred to as *cardinal directions*, and the four directions that are offset by $\frac{1}{8}$ turn from the cardinal directions will be called intercardinal directions. In addition, concrete examples shown in this work may refer to a particular cardinal direction such as East, or a particular intercardinal direction such as South-West (SW).

One reasonable concern about the introduction of diagonal motion in a world represented by an approximate cellular decomposition is how it affects the connectivity of the coverage area. This work assumes that all motions in the environment move directly between the centers of the cells created by approximate cellular decomposition. In addition, it is assumed that the robots are able to move directly

between cells which share only a corner, even if one or both of the adjacent cells that would connect them are out of bounds. Since other work typically assumes that the robot is roughly the same size as the area it covers, this assumption requires justification. Because the problem explored here is motivated by a team of autonomous drones observing an area from above, it is safe to assume that the robots are significantly smaller than the area they are covering at a particular moment. As a result, it seems justifiable to assume that the robots are able to move between diagonally connected in bounds cells, as there is likely to be enough room for a drone to complete such a move in a realistic scenario.

In this work, the author assumes that all robots start out in the same location. This assumption makes the work less general than other work that allows for arbitrary starting locations, such as [11]. However, starting all robots in one location is often a practical assumption, as it allows for easy deployment of robots from a single location. It is also worth noting that starting each robot in the same place often makes for a more challenging coverage task than starting with robots distributed throughout the space. This is because spatially distributed robots may already be close to evenly dispersed along a single Hamiltonian cycle through the coverage space, and so moving to this extremely convenient state may be a low cost operation. Note that in order to take advantage of this situation with the highest probability, robots must be able to move freely between their starting position and an idealized starting postion for spanning tree following. Note that this initializtion step is not addressed by the MSTC algorithm in [11].

Unlike many of the efforts to adapt spanning tree coverage to a multi agent case, this work allows for environments that can't be cleanly represented as a collection of obstacle-free cells twice the size of a robot's coverage footprint. This assumption complicates the issue of creating and traversing spanning trees significantly. However, it is likely to be an important practical assumption for many realistic tasks. For

example, in a search and rescue setting, people may be trapped around collapsed structures and other difficult to navigate corners of the coverage area. The capaility to plan spanning tree paths through such complex environments is also useful for the case where a robot has viewed an area with low scrutiny and must now revisit a small portion of that area that requires high scrutiny. Although it is not necessary to forbid revisiting locations that have already been adequately covered, a good path planning strategy should not visit these locations unless doing so is the most efficient way of covering the locations that do require a visit.

Like much of the work that addresses multi-agent robot coverage, this work assumes that the robots share information while performing coverage. This assumption abstracts away a significant and complex barrier to implementing the described algorithms on real robots. However, because this work is motivated by a use case in which the robots are substantially smaller than their coverage area, and because an approximate cellular decomposition is used to describe the set of locations in the coverage area, it is possible to communicate full information between the robots in this scenario using only small and relatively infrequent mesages. In addition, realistic flying robots are almost always equipped with long range communication devices to allow manual control by a human operator and to transmit observations in real time. For a search and rescue scenario in particular, it seems likely that the drones would be equipped with the ability to transmit high resolution photos at the same rate that coverage information is passed, meaning that the passing key planning information represents only a small fraction of the drone's available bandwidthe. With the considerations in mind, it seems safe to assume that the required level of communication between drones is at least possible in a realistic scenario.

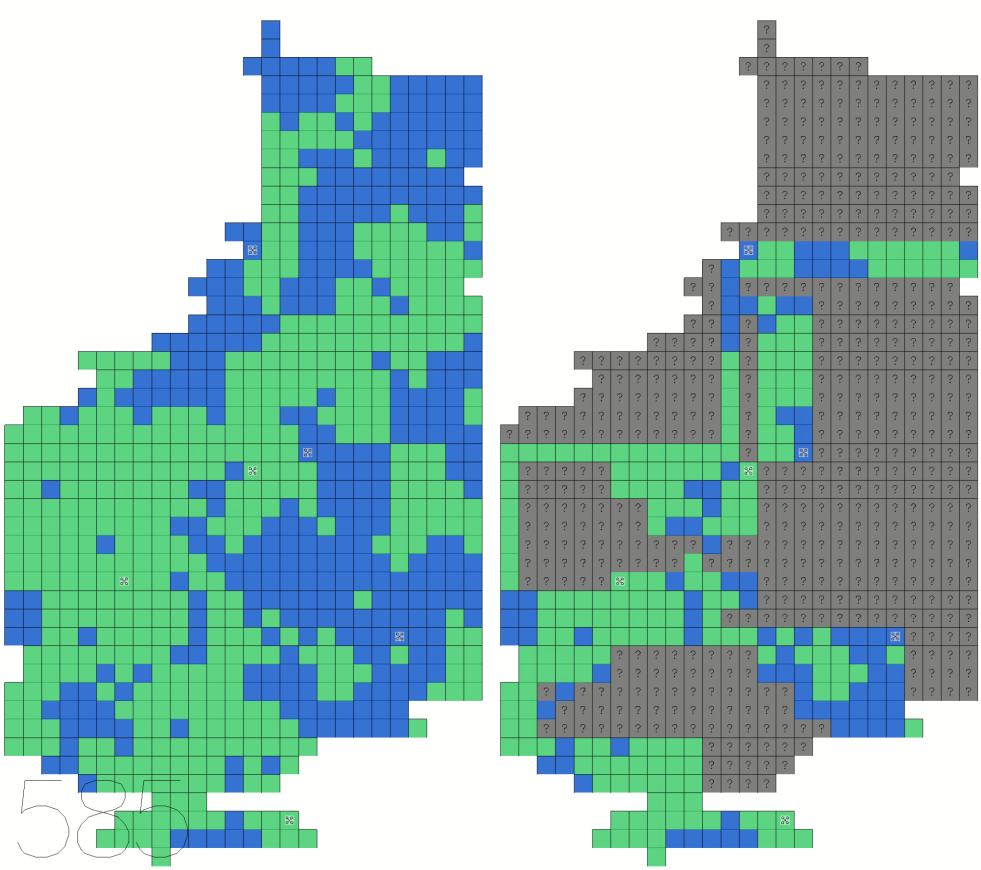
As previously stated, each drone in this formulation has access to moves that increase and decrease its field of view. For simplicity, it is assumed that a drone's field of view can be in only two states which correspond to two possible altitudes.

A drone flying low can view a single cell in the environment's approximate cellular decomposition. A drone flying high can view a 3x3 grid of these environment cells. Depending on which altitude the drone currently occupies, it has access one of two vertical motions: ascend and descend. Because this problem statement allows for more types of robot state changes than the previously covered work, it is worth addressing the optimization criteria and cost function for this task. Much of the existing work on multi agent coverage aims to find a set of robot paths that complete the coverage task while minimizing the length of the longest path taken by any of the robots. In this work, it makes more sense to assign time costs to all of the moves available to the drones. Then, the optimization goal for this task is to achieve complete coverage in the least possible time. It is assumed that ascending and descending each take as long as a cardinal motion between adjacent environment cells. Intercardinal motion between cells that share a vertex is assumed to take a factor of $\sqrt{2}$ more time than cardinal motions in order to correspond with the greater euclidean distance between the centers of these cells.

Chapter 2

Simulation Development

This chapter describes a software package, *oprc_env,* that simulates the scenario described above. This software exposes a standard interface, the *Policy* typeclass, that allows for interaction with a wide variety of control algorithms. The package also includes functionality to generate large environment datasets based on custom mixture distributions of environment generation algorithms, as well as functionality to run large batches of simulations with policies that are capable of modifying themselves between scenarios. Finally, the package includes a replay animation utility to aid in communicating results visually.



2.1 Simulated Environment Dynamics

The oprc_env package represents a multi agent coverage problem as a simulation in which all times and locations are represended with integer values. All ground positions in the environment are indexed by a pair of integers (x, y). For ease of communication, the x and y coordinates are assumed to correspond to longitude and lattitude, respectively. For example, the position $(2, 1)$ is directly east of the position $(1, 1)$, and the postion $(0, 0)$ is to the southwest of $(1, 1)$. The minimum x and y coordinates in an environment are both 0 by default, although this is not guaranteed to be true in general. All simulations start at time T = 0.

Note: names of datatypes, type constructors, and type constants in oprc_env start with capital letters, e.g. Position. For brevity, the names of these datatypes

will be used in reference to both the computational abstractions and the concepts they model.

The physical environment of each coverage scenario represents the locations which are in bounds, the level of detailed scrutiny required to cover each of those locations, and the positions of obstacles inside the environment. This is achieved by a map from Position to Patch, where a Patch can represent an environment cell which requires observation at a particular level of detail. This level of detail can be either Close or Far. Patches which require Close detail must be observed by a drone flying at a Low altitude in order to be considered covered, while patches that require only Far detail will be covered once observed by a drone flying at either a High or Low altitude.

Each Patch in an environment can have up to eight neighbors. Four of these neighbors are adjacent to the patch in the four cardinal directions North, East, South, West, while the remaining four share a vertex with the patch and lie in one of the four intercardinal directions NE, SE, NW, SW. For a given location, any arbitrary subset of these eight neighboring locations may be in bounds.

2.2 Environment Generation

In order to evaluate the coverage performance of any agent with a Policy instance, it is necessary to have a simulated environment for that algorithm to cover. It is also useful to have a large dataset of simulated environments for the agent to cover when collecting performance statistics. Finally, since some Policy instances may be capable of learning and specializing their behavior to the distribution of environments previously explored, it is useful to have fine control over the nature of the environments in a particular dataset. To meet all of these needs, *oprc_env* has a utility executable called *generate_environments* to create custom environments and

environment datasets based on a specification of the distribution from which those environments should be drawn.

Before describing the capabilities of *generate_environments* in full, it is useful to consider the algorithms used to generate one specific environment. There are a few different algorithms for this purpose, and each one generates environments from a distribution that is qualitatively different from the others.

The simplest environment generation algorithm to describe is called the *BernoulliGen*. As the name suggests, this generator samples a required level of scrutiny from the same Bernoulli distribution for each location in the generated environment's Footprint. As a result, most of the content of the *BernoulliGen* algorithm comes from the procedure used to generate Footprints. The *randomFootprint* function supplies this functionality.

The Fisher-Yates shuffle is an algorithm to reorder, or shuffle, the elements of a finite sequence. It is possible to run this algorithm in-place on a sequence of length n in $O(n)$ time. In addition, given a way to randomly select an element from a range of integers with uniform probability, the Fisher-Yates shuffle has the useful property of producing any possible reordering of the sequence with uniform probability. The original work by Richard Durstenfeld to develop this algorithm for use on a computer can be found in [12]. The version presented here is a generalization of that algorithm to operate directly on sequences of arbitrary data.

The *Shuffle* function assumes access to a function g that can accept an integer argument i and return an integer sampled uniformly from 0 to i, inclusive. It operates on l, a finite sequence of values of any datatype. For simplicity, it also assumes access to functions that get the length of a finite sequence and swap the elements of that sequence at two indices. Finally, note that sequence indexing starts at 0.

function SHUFFLE(g, l)

$n \leftarrow length(l)$

while $n > 0$ **do**

$s \leftarrow g(0, n)$

$swap(l, s, n)$

$n \leftarrow n - 1$

end while

return l

end function

The actual source code for this algorithm can be found in the FisherYatesShuffle module in *oprc_env*.

2.3 Environment Interface

At any given time during an in-progress coverage scenario, only a limited subset of environment information is made available to the Policy algorithm that controls drone motions. The set of in bounds locations, or Footprint, of the environment is always known. The partial information available to an agent is represented by the EnvironmentInfo datatype, and this type contains a map from Position to PatchInfo. The set of Position values in this map's domain is normally the same Footprint that belongs to the real environment. A PatchInfo represents a state of knowledge about the patch at a particular location. Patches can be Unseen, Classified, or FullyObserved. As the names suggest, Unseen and FullyObserved correspond to patches which have never been seen at all and patches that have been completely covered, respectively. A Classified patch is one where the level of observation detail required (DetailReq) is known, but the patch has not been fully covered yet. In practice, this only comes up for patches which require Close scrutiny and have only been seen by one or more drones flying at a High altitude.

The EnvironmentInfo datatype, together with the previously described Ensem-

bleStatus, forms the content of the WorldView datatype. A WorldView represents all of the information about the current state of the simulation available to a control algorithm. Control algorithms plug in to the simulation via an instance of the Policy typeclass, which abstracts the behavior of any type capable of commanding directions to an Ensemble and potentially maintaining a notion of internal state. Specifically, the typeclass is defined as:

```
class Policy p where
  nextMove :: p -> WorldView -> (NextActions, p)
```

As the name suggests, NextActions is a datatype that expresses newly issued commands for some drones to follow:

```
type NextActions = [(Drone, Action)]
```

Some control algorithms that are capable of learning from experience and modifying behavior in a given simulation based on results from a previous one. The PersistentPolicy typeclass exists in order to provide the interface required for this functionality. .....

2.4 Running Simulations

The Scenario type and the functions which operate on it are responsible for running simulations. The type itself consists of a WorldState and a Policy for tracking the instantaneous state of the simulation, along with time and history parameters to allow simulation replays and performance analysis:

```
data Scenario p =
  Scenario {
    getPolicy :: p
```

```
  , getWorldState :: WorldState
  , getTime :: Integer
  , getHist :: MoveHistory
  }
```

The function that steps simulations forward in time takes any `(Policy p => Scenario p)` and applies nextMove to that Scenario's Policy and a WorldView generated from the current WorldState. It then

Chapter 3

Offline Planning Policies

Chapter 4

Online Planning Policies

Chapter 5

Conclusion

5.1 Future Work

Although this work is fairly permissive of complex environment shapes, it does not consider coverage of environment that can't be exactly represented as a square grid. As noted previously, the search-and-rescue scenarios that motivate this work could benefit from close inspection of tight corners and areas near the perimeter of environment boundaries. Since these boundaries may be due to collapsed or naturally occusring structures, their borders are unlikely to be aligned with cardinal directions. An algorithm that could adapt its behavior near these regions could be of great practical importance. For an example of other work that has achieved this kind of extension in the past, consider the exact coverage variant of STC as described in [2]. It may also be useful to represent the borders of regions requiring low or high scrutiny as arbitrary polygons, although the opportunity for improved algorithmic performance or other practical benefits are less clear in this case.

Another potential improvement of this work would allow for persistent coverage. Targets may be mobile in certain search and rescue situations, and so it is necessary to re-visit locations periodically in order to make sure nobody has moved

into them.

Chapter 6

Bibliography

[1] H. Choset, "Coverage for robotics - A survey of recent results," *Annals of Mathematics and Artificial Intelligence,* vol. 31, October 2001.

[2] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence,* vol. 31, October 2001.

[3] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain-covering algorithm for an AUV," *Autonomous Robots,* vol. 3, June 1996.

[4] H. Choset, E. Acar, A. Rizzi, and J. Luntz, "Exact Cellular Decomposition in Terms of Critical Points of Morse Functions," *Proceedings of the 2000 IEEE International Conference on Robotics and Automation,* April 2000.

[5] D. Kurabayashi, J. Ota, T. Arai, S. Ichikawa, S. Koga, H. Asama, and I. Endo, "Cooperative Sweeping by Multiple Mobile Robots with Relocating Portable Obstacles," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* November 1996

[6] X. Zheng, S. Jain, S. Koenig, and D. Kempe, "Multi-Robot Forest Coverage," *IEEE/RSJ International Conference on Intelligent Robots and Systems,* August 2005

[7] E. Arkin, S. Fekete, and J. Mitchell, "The lawnmower problem," *Proceedings of the 5th Canadian Conference on Computational Geometry,* August 1993

[8] Y. Gabriely and E. Rimon, "On-Line Coverage of Grid Environments by a Mobile Robot," *Computational Geometry,* April 2003

[9] T. Cormen, C. Lieserson, R. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition,* MIT Press and McGraw-Hill, 2001

[10] A. Itai, C. Papadimitrou, and J. Szwarcfiter, "Hamilton Paths in Grid Graphs," *SIAM Journal on Computing,* November 1982

[11] N. Hazon and G. Kaminka, "Redundancy, Efficiency and Robustness in Multi-Robot Coverage," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation,* April 2005

[12] R. Durstenfeld, "Algorithm 235: Random Permutation," *Communications of the ACM,* July 1964