Real-Time Planning for Covering an Initially-Unknown Spatial Environment

Vikas Shivashankar[1] and **Rajiv Jain**[1] and **Ugur Kuter**[2] and **Dana Nau**[1,2,3]
[1]Department of Computer Science, [2]Institute for Advanced Computer Studies, and
[3]Institute for Systems Research
University of Maryland, College Park, Maryland 20742, USA
{svikas, rajivj,ukuter,nau}@cs.umd.edu

Abstract

We consider the problem of planning, on the fly, a path whereby a robotic vehicle will cover every point in an initially unknown spatial environment. We describe four strategies (Iterated WaveFront, Greedy-Scan, Delayed Greedy-Scan and Closest-First Scan) for generating cost-effective coverage plans in real time for unknown environments. We give theorems showing the correctness of our planning strategies. Our experiments demonstrate that some of these strategies work significantly better than others, and that the best ones work very well; e.g., in environments having an average of 64,000 locations for the robot to cover, the best strategy returned plans with less than 6% redundant coverage, and took only an average of 0.1 milliseconds per action.

Introduction

This paper focuses on planning and execution of a path whereby a robotic vehicle will cover every point in an initially unknown spatial environment. This problem is becoming increasingly relevant in a variety of mobile robotics applications, such as robotic vacuum cleaning, mine detection, and lawn maintenance. In such applications, the robot is simply started and allowed to explore the environment by itself. As the robot moves, the on-board sensors provide the robot information about the environment, its layout, and its boundaries, and enable it to perform coverage in real time.

Most existing work on planning algorithms that guarantee coverage of a target environment have generally assumed that the environment is known rather than unknown. Existing works typically require more sensory and computational power than the simpler robots may have on board (Acar, Choset, and Atkar 2001; Acar et al. 2003; Chen and Song 2005), and they were tested in only some small-sized coverage-planning scenarios (LaValle 2004; Choset 2000; Simmons and Koenig 1995; Pirzadeh and Snyder 1990).

We formalize *RUC (Real-time Unknown-environment Coverage)* planning, i.e., the problem of generating, on the fly, a path whereby a robotic vehicle will cover every point in an initially *unknown* spatial environment. We describe four new planning strategies: Closest-First Scan (CFS), Greedy-Scan (GS), Delayed Greedy-Scan (DGS) and Iterated WaveFront (IWF), which is a generalization, to unknown environments, of the well-known WaveFront algorithm (Zelinsky et al. 1993). We provide theorems showing that our planning strategies are correct.

We present several experimental comparisons of the strategies on 200 randomly-generated RUC problems. These problems used 20 different floor layouts, of which 18 layouts were obtained from Google Images and 2 layouts were handcrafted to provide hard problems. Our results include the following:

- Our planning strategies can quite easily run in real time. On average, our floor layouts contained 64,000 locations for the robot to visit; and in terms of the average decision time per action, CFS, GS, DGS, and IWF averaged 0.099, 0.12, 0.1, and 1.592 milliseconds, respectively.
- Our strategies generated plans with only a small amount of redundancy. The average percentage of times the robot visited a location more than once was 6.41% with CFS, 10.31% with GS, 5.86% with DGS, and 5.15% with IWF.
- Finally, DGS consistently outperformed the other strategies in terms of the average total cost (total number of moves and turns) of the solution plans.

Definitions

We perform planning in a spatial environment Σ, modeled as an $n \times m$ grid world. There are two kinds of cells (i.e., *locations*) in Σ: *free* and *occupied*. A location l' is *reachable* from another location l in Σ if there is a path from l to l' such that all of the locations on that path are free. In such a case, l' is said to be *l-reachable*. If there is no such path, l' is *unreachable* from l.

Figure 1 shows an illustration of a spatial environment that represents a floor layout in a house. Each point on this two dimensional floor layout is a location. The points drawn in black denote occupied locations that the robot cannot move through. From the location l_0 of the robot as shown in the figure, l_1 is a reachable location and l_2 is not: every path from l_0 to l_2 contains at least one occupied location.

The robot has three actions: *move from one location to a neighboring location* in Σ,[1] *turn the robot left*, and *turn*

[1]We assume that the robot cannot move diagonally; however this does not restrict the generality of our formalism and the planning strategies presented in this paper.

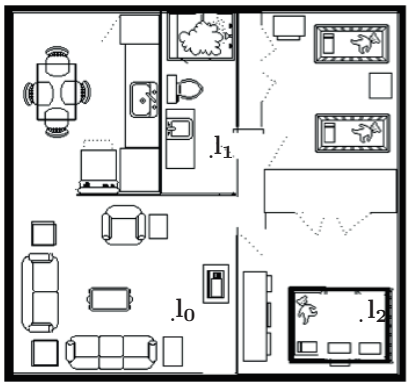

Figure 1: An illustration of a real-time coverage-planning environment. The points drawn as black pixels denote occupied locations that the robot cannot move through. From l_0, the location l_1 is reachable and l_2 is unreachable.

the robot right. A *plan* is a sequence of actions that, when executed, induces a path in Σ starting from an initial location l_0 and ending in a location l. We define L_π as the set of all locations visited by the execution of π in l_0. A plan π *covers* Σ if every l_0*-reachable* location in Σ is in L_π.

The robot is given *no prior knowledge of the environment*. For example it does not know initially the location it is in, the boundaries of the environment, which locations are free and which are occupied, and the enclosed spaces formed by the occupied locations. Thus, the robot must decide which direction it can or should move along **only** using its *sensors* and its *mental map* of the environment, as defined below.

The robot has four sensors for the directions North, South, East, and West. The sensors are *directional*, i.e., they return distance information about a particular direction d at a location l. The sensors are not capable of returning information beyond an occupied location. For example in Figure 1, the robot's sensors provide information about the obstacles such as the walls, furniture, and other objects on the floor layout. Given a location symbol s and a direction d, the *sensor function* $\sigma_d(s)$ specifies the length of the path that starts in the robot's current location and ends in an occupied location along d.

The *mental map* M of the robot is a tuple $(\mathcal{G}, V, F, O)$, where $\mathcal{G}$ is a snapshot of the area mapped out by the robot so far, initialized to the empty grid in the beginning. V holds the set of cells in $\mathcal{G}$ that are already visited by the robot, and F and O are the sets of free and occupied locations that the robot discovers in $\mathcal{G}$. A mental cell is said to be *uncovered* if it is free but not visited. A mental cell is said to be *unobserved* if is neither free nor occupied. Note that we assume **no** knowledge of the size or shape of Σ in M.

A *real-time unknown-environment coverage (RUC) planning problem* is a tuple $P = (\Sigma, l_0, M_0, \sigma)$, where Σ is a spatial grid world environment, l_0 is the initial location of the robot in Σ, M_0 is the robot's initial mental map of Σ, and σ is the robot's sensor function. A *solution* for P is a plan π that covers Σ.

Algorithm 1: A high-level description of a generic RUC planning strategy $\mathcal{S}$. Initially, $V = F = O = \emptyset$.

```
1  Procedure S(G, V, F, O, σ);
2  begin
3      get an arbitrary mental cell s in G for the current location ;
4      V ← V ∪ {s}; π ← ∅;
5      ⟨G, V, F, O⟩ ← UpdateMap(G, V, F, O, σ, s);
6      repeat
7          π′ ← GenerateSubPlan (G, V, F, O, s);
8          π ← π.π′ ;
9          repeat
10             remove action a from head of π′ and execute it;
11             get the next cell s;
12             V ← V ∪ {s} ;
13             ⟨G, V, F, O⟩ ← UpdateMap(G, V, F, O, σ);
14         until π′ = φ;
15     until F = V;
16     return π;
```

Strategies for Solving RUC Problems

Algorithm 1 shows a high-level description of a generic RUC planning strategy $\mathcal{S}$. The input to the planning procedure is the empty mental map $(\mathcal{G}, V, F, O)$ where $\mathcal{G}$ is initialized to a $k \times k$ grid for some number k, and the sensor model σ. Note that the robot does not initially know the other locations and the transitions in the environment Σ. In Line 3, $\mathcal{S}$ arbitrarily selects a cell s in $\mathcal{G}$ as the robot's current location.

In Line 5, $\mathcal{S}$ updates $\mathcal{G}$ using the UpdateMap subroutine. Algorithm 2 describes the UpdateMap procedure. The input to the procedure is the current mental map, the sensor model σ and the current location s. For each direction d, UpdateMap gets the sensory information about the distance $\sigma_d(s)$ from s to the closest occupied locations in Σ in direction d (Line 4). UpdateMap then updates the mental map of the robot with that information (Lines 7 – 9). Since the robot's sensors cannot see beyond an occupied location in a given direction d, UpdateMap knows that all of the locations between the robot's current location and the occupied location along direction d are free. Assuming l_i denotes the cell i units from s in direction d, it marks cells

Algorithm 2: Description of the UpdateMap subroutine

```
1  Procedure UpdateMap (G, V, F, O, σ, s);
2  begin
3      foreach direction d do
4          c ← σ_d(s) ;
5          if c > dist_max(s, d) then
6              Expand G by c − dist_max(s, d) units along
               direction d;
7          for i < c do
8              F ← F ∪ {l_i};
9          O ← O ∪ {l_c};
10     return (G, V, F, O);
```

Figure 2: An example environment in which the robot is initially at location A. The GS strategy in this scenario will keep changing the robot's direction to move along the path $\langle A, B, C, D, E, F, G, H, I\rangle$, along which the maximum number of uncovered cells lie. As a result, the robot visits the locations A, B, C, D, E, F, and G several times.

$l_1 \ldots l_{c-1}$ in the robot's mental map as free (Line 8) and l_c as occupied (Line 9).

If the number of cells from the current cell s to the boundary of the mental grid $\mathcal{G}$ in direction d, say $dist_{max}(s, d)$, is less than $\sigma_d(s)$, then UPDATEMAP *expands* $\mathcal{G}$ with $\sigma_d(s) - dist_{max}(s, d)$ rows or columns of cells depending on d (Lines 5 – 6). All of the new cells in the expanded $\mathcal{G}$ are marked as unobserved, except the ones just sensed by the robot. Finally, UPDATEMAP returns the updated mental map.

In Lines 6–15 of Algorithm 1, $\mathcal{S}$ successively generates and executes plans in Σ. In each iteration, it invokes the subroutine GENERATESUBPLAN to get the partial plan π' to execute in that iteration. RUC planning strategies differ in the implementation of this subroutine, as described in the following sections. $\mathcal{S}$ then updates π with the partial plan π' (Line 8) and executes π'. At each step of the execution, it marks the resulting cell s as 'visited' and updates the mental map at s using UPDATEMAP (Lines 12 and 13).

When the set of visited cells V is equal to the set of free cells F, $\mathcal{S}$ terminates and returns the plan π (Line 16).

Closest-First Scan (CFS). The CFS strategy implements GENERATESUBPLAN as follows. It first checks for uncovered cells adjacent to s. If it finds any, it randomly chooses one of them and returns π' consisting of the move action to that cell. If not, it performs a breadth-first search in $\mathcal{G}$ to find an uncovered cell s' closest to s and the corresponding plan π' to get to it. It then returns π'.

Greedy Scan (GS). When there exist uncovered cells adjacent to s, instead of randomly choosing one of them as CFS does, GS calculates the number of uncovered cells from s in each direction d and greedily chooses the direction d_{max} that maximizes this quantity. It then returns the partial plan π' consisting of a single move action along d_{max}.

Note that this strategy has the potential for trapping the robot in a location with no adjacent uncovered cells. In such situations, we perform breadth-first search in $\mathcal{G}$ to move the robot to the closest uncovered cell.

Delayed Greedy-Scan (DGS). The strategies above may often cause the robot to visit a cell multiple times, particularly in environments that contain many obstacles. For example, consider the environment shown in Figure 2. The robot starts in the location A in the middle. During execution, GS will change the robot's direction at each timestep to move along the path along which the maximum number of uncovered cells lie. GS can therefore, because of its greedy choices at every timepoint, fragment the map badly leading to many revisits.

DGS alleviates this issue by delaying the decision-making to the end of execution of a *sub-plan* rather than an action. It implements the GENERATESUBPLAN routine as follows. In a cell s, DGS calculates the number n_d of *consecutive* uncovered cells starting from s in each direction d and chooses the direction d_{max} that maximizes this quantity. It then returns a partial plan π' consisting of $n_{d_{max}}$ move actions in the direction d_{max}. In case there are no adjacent uncovered cells, we perform breadth-first search in $\mathcal{G}$ to move the robot to the closest uncovered cell.

There are two important points to note here. Firstly, we consider only the number of *consecutive* uncovered cells to generate π' in contrast to GS where we consider *all* uncovered cells in a particular direction. This ensures that we cover a small number of uncovered cells immediately reachable from s rather than a large number of uncovered cells far away from s, minimizing the number of revisits.

Secondly, we do not make a decision after every move, but execute all $n_{d_{max}}$ actions in π' before making the next decision. This minimizes fragmentation of $\mathcal{G}$. For example, in Figure 2, the sequence of cells visited by the DGS strategy would be $\langle A, B, E, F, I, F, E, B, A, C, D, G, H\rangle$, which has much lesser revisits than GS.

Iterated WaveFront (IWF). The IWF planning strategy is based on the WaveFront algorithm reported by Zelinsky et al (Zelinsky et al. 1993). The WaveFront algorithm performs a breadth-first search from a specified location in an a priori known environment and marks each cell with the minimum depth in the tree until every cell in the map is reached. The outcome of this search is the WaveFront map. Once the map is generated, the WaveFront algorithm follows the path from cells with the highest values to ones with the lowest values, never returning to a covered cell. This results in a coverage of the map, ending at the goal state.

IWF generalizes the WaveFront algorithm to unknown environments. Algorithm 3 shows the implementation of the GENERATESUBPLAN routine for IWF. Since the map of the environment is not available to the algorithm, IWF generates wavefront maps over its *mental map* to reach an *unobserved* cell. In particular, IWF calculates the set C of all unobserved cells closest to s in $\mathcal{G}$ and chooses a cell g randomly from it (Lines 3 – 4). If there are no such cells, then IWF selects an uncovered cell g in $\mathcal{G}$ closest to s (Line 6). With g, IWF creates a WaveFront map and calculates the path (and corresponding plan π) in this map from s to g with decreasing wavefront values (Lines 7 and 8). If g has no adjacent uncovered locations, then we perform a breadth-first search to estimate π'', the plan to reach the closest uncovered cell (Line 10). It then returns the concatenation of π' and π'' (Line 11).

Algorithm 3: A high-level description of the GENERATESUBPLAN routine used by IWF.

```
1  Procedure IWF-GENERATESUBPLAN(G, V, F, O, s);
2  begin
3      let C be the set of all unobserved cells closest to s;
4      if C ≠ ∅ then arbitrarily select a cell g from C ;
5      else
6          let g be an uncovered cell closest to s ;
7      create a WaveFront map with g;
8      let π' be the plan to traverse path of highest numbers ;
9      if there are no adjacent uncovered cells of g then
10         π'' ← perform breadth-first search to estimate plan
           to reach nearest uncovered cell ;
11     return π'.π'';
```

Formal Properties

All of the strategies described above are provably correct, i.e. a plan π returned by any of the strategies for the problem P, is guaranteed to cover all l_0-reachable cells.

Let $\mathcal{S}$ represent the generic planning strategy described in the previous section and $M = (\mathcal{G}, V, F, O)$, the mental map obtained after executing $\mathcal{S}$.

Theorem 1. *$V = F$ iff all l_0-reachable cells are covered.*

Proof. Let us assume the contrary, i.e. $V = F$ but there exists an l_0-reachable cell (say l_n) that is uncovered. Since l_n is l_0-reachable, there exists a path $L{=}< l_0, l_1, ..., l_n >$ of free cells. Assuming without loss of generality that l_n is the first uncovered cell on this path, this implies that by the geometry of the map, l_n should have been added to F when the robot reached l_{n-1}. In all the strategies, a cell is added to V only when it is covered. Therefore, this implies that since l_n was added to F but not subsequently covered (and hence not in V), $V \neq F$, leading to a contradiction.

Suppose all l_0-reachable cells are covered. Since a cell is added to F only if it is l_0-reachable, F can contain only l_0-reachable cells. Since a cell is added to V once it is covered, when all l_0-reachable cells are covered, $V = F$. □

Theorem 2. *$\mathcal{S}(\mathcal{G}, V, F, O, \sigma)$ always terminates.*

Proof. Since an RUC problem is bounded, the number of l_0-reachable cells is finite. Also, $|V|$, the number of l_0-reachable cells visited by the robot, monotonically increases since all of the strategies execute at least one move action into a free cell in every iteration. Since $|F|$ (and therefore, $|V|$) is bounded above by the number of l_0-reachable cells, all strategies are guaranteed to terminate. □

Theorem 3. *The plan π returned by $\mathcal{S}(\mathcal{G}, V, F, O, \sigma)$ covers all l_0-reachable cells.*

Proof. By Theorems 1 and 2, since all strategies terminate (and are guaranteed to by Theorem 2) when $V = F$, the plan π returned by the strategies is guaranteed to cover all l_0-reachable cells. □

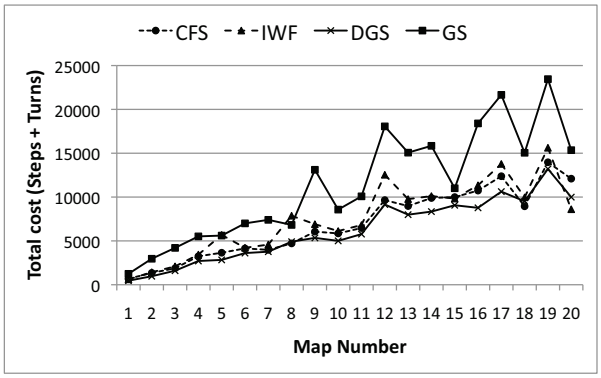

Figure 3: Comparison of the total cost (i.e., the sum of total number of extra steps and turns) of CFS, DGS, GS, and IWF. Each datapoint is the average of the total cost over 10 randomly-generated RUC planning problems per floor layout.

Experiments

We have conducted several experiments to compare the performance of our four planning strategies. Our primary hypotheses was that DGS and IWF would significantly outperform CFS and GS in terms of the average total cost, with IWF having the least number of forward moves and the DGS strategy having the fewest number of turns.

We have implemented a simulation environment using Java 1.6 for RUC problems described above. The simulation environment stores the map of the environment to validate the plan generated by the RUC planning strategies. Note, however, that this map is not available to the strategies themselves. The execution engine uses a bitmap representation for an input floor layout. A floor layout in this representation is a black-and-white image, where white spaces denote free locations whereas the black spaces denote occupied ones.

We have used 20 different floor layouts, of which 18 layouts were obtained from Google Images and 2 layouts were handcrafted to provide hard problems. We have acquired the layouts from Google Images by searching for the keywords "room," "floor," and "blueprint," and by choosing the first images that contained floor layouts or enclosed spaces similar to Figure 1. The average size of our experimental floor layouts were 64,000 locations.

We converted the layout images to black and white bitmap images so they could be run in the simulation. This limits the bias in creating manual floor layouts, while still picking realistic environments. Additionally this approach provided a diverse set of layouts. In the simulations, we have generated 10 random initial locations for the robot on each floor layout. If an initial location was an occupied location on the layout or in locations such as inside the sofa in Figure 1, we have not used the resulting RUC planning problem and generated a new random problem instead.

For mobile robots, sometimes turns occupy more time and induces more errors than moving in straight lines (Choset 2001), depending on the specific robot and the environment. For that reason we graphed total cost and the number of turns separately. Figure 3 shows the results of the comparisons

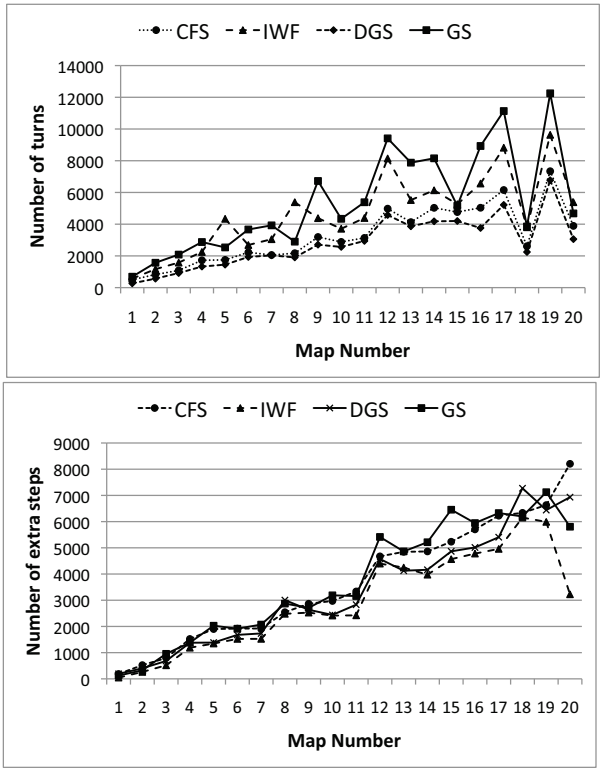

Figure 4: Average number of turns (top) and extra steps (bottom) performed by CFS, DGS, GS, and IWF in 20 floor layouts. Each datapoint is the average of the total cost over 10 randomly-generated RUC planning problems in a floor layout.

of the average total cost of the plans generated by the four strategies. The total cost of a plan executed by the robot is the sum of the average number of extra steps, i.e., the robot's moves into a cell that's already been visited before, and the average number of turns the robot performed.

In these experiments, **DGS performed better than the other 3 strategies consistently across almost all planning problems.** DGS' solutions had an average cost 45% lesser than those of GS and 18.29% lesser than those of IWF. IWF performed in general worse than CFS because the actions generated by IWF moved the robot in a step-like manner; thus alternating between moving forward and turning quite often. For example, in the presence of diagonal walls, IWF moved the robot along the border since it optimizes only on the number of steps, disregarding the number of turns.

Figure 4(top) shows the comparison of average number of turns generated by the CFS, DGS, GS, and IWF strategies in our 20 floor layouts, respectively. IWF returned plans with a high number of turns as compared to both DGS and CFS: more precisely, the number of turns it produced was 46.07% higher than those produced by CFS. Figure 4(bottom) shows the comparison of average number of extra steps of the CFS, DGS, GS, and IWF strategies. Here, IWF did 20% better than CFS and 20.95% better than GS. This demonstrates that **the IWF strategy optimizes heavily on the number of steps**, which might be preferable in case of errors in the odometer.

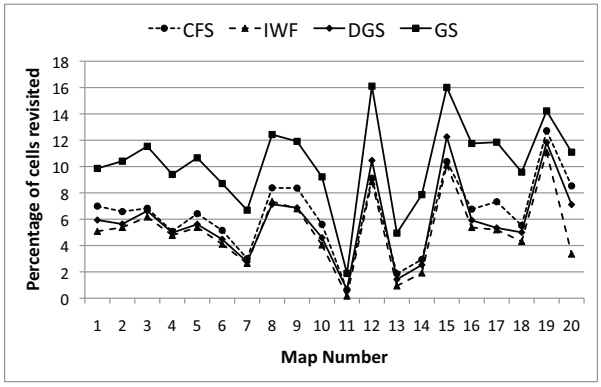

Figure 5: Percentages of free space covered multiple times by CFS, DGS, GS, and IWF in 20 floor layouts. Each datapoint is the average percentage over 10 random RUC planning problems in a floor layout.

Figure 5 compares the percentage of the floor layout revisited by the robot across all test layouts, as a measure of the redundancy in the solutions produced by our strategies. IWF performed the best with an average revisit percentage of 5.15%. DGS has an average revisit percentage of 5.86% while CFS revisits 6.41% of the free locations. GS, however, performs badly in comparison to the others, having a revisit percentage of 10.31%. **This indicates that the solutions generated by IWF, DGS and in some cases, CFS have minimal redundancy.**

In our experiments above, CFS and DGS were the two best performers in terms of the average running times. These strategies averaged 5.287 seconds and 5.315 seconds per floor layout, respectively. GS too performed similarly with an average running time of 6.279 seconds per floor layout. IWF is an order of magnitude worse, averaging 95.085 seconds per layout. The difference in running times of IWF and the others is due to the additional complexity in generating a new WaveFront map each time the robot reaches a goal or a location whose neighbors are already covered.

The results showed that our planning strategies are able to decide and execute an action in the world very quickly: in terms of the decision times per move, CFS, GS, DGS, and IWF averaged 0.099, 0.12, 0.1 and 1.592 milliseconds, respectively.[2] Hence, we believe that they are suitable for real-time robot RUC planning applications.

Related Work

There appears to be little use of coverage planning by companies such as iRobot[3], which makes the most popular robot vacuum cleaner. The most state-of-the-art vacuum cleaners in the market today make use of random patterns to achieve

[2] All of the experiments were run on a 2.0 GHz Dual Core computer with 4 GB of memory.

[3] `http://www.irobot.com/`

coverage, the hypothesis being that these random moves, executed over a sufficiently long period of time, will achieve near-complete coverage.

Most existing works address coverage planning in known environments. Choset (Choset 2000) proposed the sound and complete Boustrophedon Decomposition algorithm to solve coverage-planning problems, assuming a fully known environment. Simmons and Koenig (Simmons and Koenig 1995) proposed algorithms to navigate robots through corridors, given approximate structural information of the corridors such as shape, size, etc. Choset (Choset 2001) provides an exhaustive survey of coverage planning methods.

Pirzadeh and Snyder (Pirzadeh and Snyder 1990) focus on coverage planning in unstructured terrains where the position of the obstacles is unknown, but the structure of the environment is known a priori. Chen and Song (Chen and Song 2005) discuss coverage motion control for cleaning robots using infrared sensors. However, their approach necessitates presence of landmark beams to guide the robot, specifically around the obstacles.

Acar et al (Acar, Choset, and Atkar 2001) focus on robotic de-mining for an unknown environment based on the concept of critical points. The robots used, however, require more complex sensors that can detect these critical points. They extend this work to outdoor environments with emphasis on mine-detecting applications (Acar et al. 2003).

Yang and Luo (Yang and Luo 2004) discuss a neural network based approach to coverage planning in dynamic but completely known environments, which they then extend to unknown environments (Luo and Yang 2008). However, they still assume that the size of the map is known a priori.

Conclusions

We have formalized the problem of planning, in real time, a path that covers an unknown spatial environment. We have presented four strategies for the problem, have presented theorems showing the correctness of these strategies, and have compared the strategies experimentally in 200 randomly generated planning problems.

Our experiments demonstrated that the three strategies, IWF, DGS, and sometimes CFS, generated solutions with low redundancy percentages. The results also showed that these strategies were able to decide and execute an action in the world very quickly. Among them, the best strategy overall was Delayed Greedy-Scan (DGS). It produced plans having the lowest number of turns and the lowest total cost of the robot's movement, and it produced these plans very quickly. On floor layouts having an average of 64,000 locations to cover, it had a redundancy percentage of only 5.86% and took an average of 0.1 milliseconds per action.

In the near future, we will investigate optimality in RUC planning problems. We intend to start with our redundancy metrics described in the paper and attempt to minimize redundancy in the solution plans as a way of achieving optimality. One possibility is to start the planning algorithms with an approximate initial mental map as background knowledge. We are also interested in investigating heuristics such as the ones used in real-time heuristic search (Korf 1990).

One limitation of our work is that we only considered *static* spatial environments, in which there are no changes in the world except for the robot's movements. We intend to generalize our work to *dynamic* real-time coverage planning scenarios, in which there is sensor uncertainty and there are moving objects other than the robot.

Acknowledgments. This work was supported in part by DARPA and U.S. Army Research Laboratory contract W911NF-11-C-0037, AFOSR grant FA95500610405, and NSF grant IIS- 0948123. The information in this paper does not necessarily reflect the position or policy of the U.S. Government, and no official endorsement should be inferred.

References

Acar, E.; Choset, H.; Zhang, Y.; and Schervish, M. 2003. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. *The International Journal of Robotics Research* 22(1):441 – 466.

Acar, E. U.; Choset, H.; and Atkar, P. N. 2001. Complete sensor-based coverage with extended-range detectors: a hierarchical decomposition in terms of critical points and voronoi diagrams. In *International Conference on Intelligent Robots and Systems*.

Chen, C.-H., and Song, K.-T. 2005. Complete coverage motion control of a cleaning robot using infrared sensors. In *IEEE International Conference on Mechatronics*.

Choset, H. 2000. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots* 9(3):247–253.

Choset, H. 2001. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31(1-4):113–126.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence*.

LaValle, S. M. 2004. Planning algorithms.

Luo, C., and Yang, S. 2008. A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. *IEEE Transactions on Neural Networks* 19(7):1279–1298.

Pirzadeh, A., and Snyder, W. 1990. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *IEEE International Conference on Robotics and Automation*, 2113–2119 vol.3.

Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *IJCAI*, 1080–1087.

Yang, S., and Luo, C. 2004. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 34(1):718–724.

Zelinsky, A.; Jarvis, R.; Byrne, J. C.; and Yuta, S. 1993. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *International Conference on Advanced Robotics*.