A Fast Traversal Heuristic and Optimal Algorithm for Effective Environmental Coverage

Ling Xu and Tony Stentz
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
{lingx, axs}@cs.cmu.edu

***Abstract*— Tasks such as street mapping and security surveillance seek a route that traverses a given space to perform a function. These task functions may involve mapping the space for accurate modeling, sensing the space for unusual activity, or processing the space for object detection. With a prior map, an optimal path can be computed using a graph to represent the environment and generating the solution using known graph algorithms. However, the prior map may be inaccurate due to factors such as occlusion, outdatedness, dynamic objects, and resolution limitations. In this work, we address the NP-hard problem of optimal environmental coverage with incomplete prior map information.**

To utilize related algorithms in graph theory, we represent the environment as a graph. Using this representation, we present a graph coverage approach for optimal plan generation based on the Undirected Chinese Postman and Rural Postman problems. This approach produces a tractable solution through the use of low complexity algorithms in a branch-and-bound framework. Additionally, as the robot receives sensor updates during traversal of the environment, we update the graph to reflect those changes. The updated graph can be highly disconnected so computing an optimal solution can be NP-hard. To combat this, we introduce a heuristic for coverage path generation that helps maximize the connectivity of the updated graph. We evaluate our approach on a set of comparison tests in simulation.

I. Introduction

Many tasks, such as street sweeping, mail delivery, and robotic surveillance and patrol, require a robot to visit all points in an environment to accomplish a goal. These goals usually entail effectively mapping, sensing, or processing the space. In this work, we address this coverage problem where a robot is required to visit most locations in a given area. We assume a map of the environment is available. This is a reasonable assumption since information of most outdoor spaces can be obtained via satellite images. However prior maps can differ from the actual environment due to factors such as occlusions, datedness, and lower map resolutions. Even with an otherwise accurate map, dynamic conditions such as the presence of people or vehicles can diminish the effective accuracy of prior information. Therefore, an additional goal for the coverage problem we are addressing is to efficiently replan when changes occur in the environment.

In robotics, there is a number of related work in the area of continuous space coverage. In continuous space coverage, the robot must pass a detector over all points in the space in order to complete a task [1][2][3][4]. While these methods ensure completeness in terms of the area covered, they do not guarantee optimal path length. In graph theory and operations research, researchers have tackled this problem by representing the environment as a graph, and using routing algorithms such as the traveling salesman [5] [6] or postman problems [7] [8] to generate optimal solutions. Because we seek optimal paths for coverage efficiency, we choose to use the graph representation as the foundation of our solution approach.

In the graph representation, nodes in the graph denote locations in the environment and edges in the graph are the paths between the locations. For example, the map in Figure 1 is converted into a graph shown in Figure 2(a) by changing each street intersection into a node and each street into an edge. Each edge has a cost assigned to it where the cost can represent measurements such as Euclidean distance between locations, terrain traversability, travel time, or a combination of several metrics. Additionally, each edge is undirected meaning it can be traversed in both directions. Another example is a Voronoi diagram where the paths are edges in the graph and the path intersections are nodes. This is one way to generate optimal paths for some of the problems in continuous space coverage.

For our problem, we seek a coverage path that visits all the edges or a designated edge subset of the graph. This coverage problem can be modeled as an arc-routing problem. We focus on two types of arc-routing problems: the Chinese Postman Problem [9] and the Rural Postman Problem. The Chinese Postman Problem (CPP) seeks an optimal path that visits all the edges of the graph at least once, and the Rural Postman Problem (RPP) seeks an optimal path that visits a predefined subset of graph edges at least once. We define an optimal path as the lowest cost coverage path given the current graph information.

The CPP and RPP differ in two ways. The CPP has a polynomial time solution of $O(n^3)$ [7] [10]. It works well for applications where it is necessary to traverse every part of the space. For example, Sorensen uses the CPP to plan tours for farming machines in static known environments [11].

In many practical problems, it is not necessary to traverse all the edges in the graph. The RPP seeks a tour that traverses a required subset of the graph edges using the extra graph edges as travel links between the required edges. Unlike the CPP, the RPP is a NP-hard problem. Optimal solutions exist that formulate the RPP as an integer linear program and solve it using branch-and-bound [8]. One recent approach [12] introduced new dominance relations such as computing the minimum spanning tree on the connected components in a

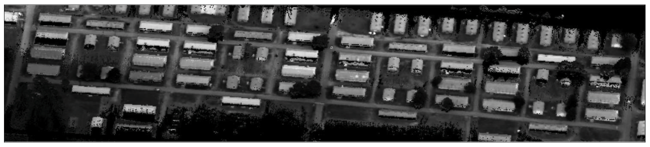


Fig. 1. The map of an urban environment is shown where the box-like shapes represent buildings and the spaces between the buildings are roads.

graph to solve large problem instances. Additionally, many TSP heuristics have been extended to the RPP [13]. For example, Christofides' approximation for the Euclidean TSP was modified for the undirected RPP and maintains its $\frac{3}{2}$ constant factor performance [14]. Improvements heuristics have been developed as postoptimality procedures for these approximation algorithms by modifying and shortening the suboptimal solution path to be as close to optimal as possible [15].

While most research on arc routing problems focus on the static environments, there has been work that address dynamic graphs. Moreira et al. present a heuristic-based approach for the Dynamic Rural Postman Problem (DRPP) [16]. They frame the problem as a machine cutting application where the graph changes as pieces of the cutting surface are cut and removed. They introduce and compare two heuristics for choosing the next edge to cut. While the run times they obtained were reasonable, the solution paths generated were not optimal.

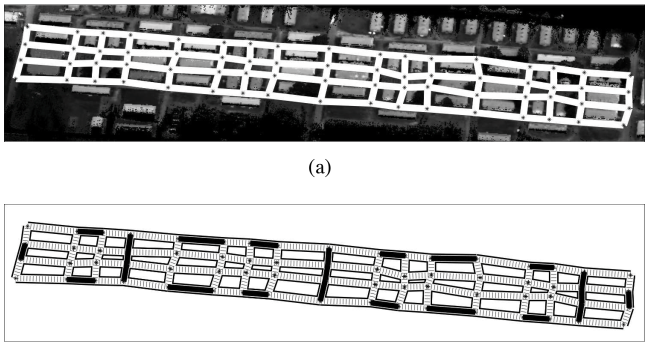
In this paper, we present two contributions. First, we introduce a general approach for the online coverage problem with an incomplete prior map. As changes in the environment are detected, we present a method for updating the graph to reflect the new changes. Our approach uses the lower complexity Chinese Postman algorithm within a branch-and-bound framework to solve the harder Rural Postman problem. Second, we present a new heuristic for path generation that traverses the graph in such a way that when changes occur, the heuristic helps keep the new coverage problem polynomial.

The rest of the paper is organized as follows. In Section II, we introduce and describe each component of the coverage algorithm. In the next section, we explain the set of tests we conducted. The results are presented and discussed in Section IV. Finally, we conclude and list future directions for this work.

II. Coverage Algorithm

A. Chinese Postman Problem

We assume the environment is initially known in the form of a prior map. This prior map is converted into a graph structure with goal locations as nodes in the graph and paths between goals as edges in the graph. The first step in solving the coverage problem is to assume the prior map is accurate and generate a tour that covers all the edges in the graph. This problem can be represented as a Chinese Postman Problem (CPP).



(a)

(b)

Fig. 2. (a) We represent the prior map as a graph where edges (white lines) denote the roads and vertices (stars) denote the road intersections. (b) Dotted and solid lines represent the CPP solution and are super-imposed on the above graph. Dotted lines denote edges that are traversed once and solid lines denote edges that are traversed more than once.

Algorithm 1: Chinese Postman Problem Algorithm

```
   Input: $s$, start vertex
          $G$, connected graph where each edge has a cost value
   Output: $P$, tour found or empty if no tour found
1  if $IsEven(G)$ then
2      $P = FindEulerCycle(G, s)$;
3  else
4      $O = FindOddVertices(G)$;
5      $O' = FindAllPairsShortestPath(O)$;
6      $Mate = FindMinMatching(O')$;
7      $G' = (G, Mate)$;
8      $P = FindEulerCycle(G', s)$;
9  end
10 return $P$;
```

The CPP optimal tour consists of traversing all the edges in the graph at least once and starting and ending at the same node. To solve this problem, we used the Edmonds and Johnson algorithm [17] [18], detailed in Algorithm 1.

The first step in the algorithm is to calculate the degree of each vertex. If all the vertices have even degree, then the algorithm finds an Euler tour using the End-Pairing technique (line 2). If any vertices have odd degree, a minimum weighted matching [19][20] among the odd degree vertices is found using an all pairs shortest path graph of the odd vertices (line 5). Because the matching algorithm requires a complete graph, the all pairs shortest pairs algorithm is a way to optimally connect all the odd vertices. The matching finds the minimum cost set of edges that connect the odd nodes (line 6). The edges in the matching are doubled in the graph making the degree of the odd nodes even (line 7). Finally, the algorithm finds a tour on the new Eulerian graph (line 8).

We ran the CPP algorithm on the example graph shown earlier. Since the graph is odd, a solution to the problem requires some of the edges to be traversed more than once. In Figure 2(b), the solution tour is depicted where the dotted and solid lines represent edges traversed once and edges traversed more than once, respectively.

B. Rural Postman Problem

In the Rural Postman Problem, there are two sets of graph edges: required and optional. We define the required subset of edge as *coverage edges*, and the optional subset as *travel edges*. Any solution would include all coverage edges and some combination of travel edges.

For combinatorial optimization problems such as the RPP, a common planning framework is branch-and-bound [21]. Branch-and-bound is a method of iterating through a set of solutions until an optimal solution is found. It consists of two steps, a branching step and a bounding step. In the branching step, the algorithm forms n branches of subproblems where each subproblem is a node in the branch-and-bound tree. The solution to any subproblem could lead to a solution to the original problem. In the bounding step, the algorithm computes a lower bound for the subproblem. These lower bounds enable branch-and-bound to guarantee a solution is optimal without having to search through all possible subproblems. Branch-and-bound is a general method for handling hard problems that are slight deviations from low complexity problems even though faster, ad hoc algorithms exist for each deviation. The process of partitioning the problem into subproblems works to our advantage since it enables the use of the applications of low complexity algorithms like the CPP as bounds on the subproblems.

In our approach, we use the branch-and-bound approach to iterate through all combinations of travel edges. We call the set of travel edges our partition set. At each branching step, we generate two subproblems by including and excluding a travel edge. Next, each subproblem in the branch-and-bound tree is solved using the CPP algorithm. The cost of the CPP solution is the lower bound on the cost of the RPP for the particular branch. If the travel set is large, this method can be computationally expensive.

To keep computation costs low, we reduce the partition set given to the branch and bound algorithm. First, we define a few terms used in the algorithm. A *coverage* or *travel vertex* is a vertex in the graph incident to only coverage or travel edges, respectively. A *border vertex* is a vertex in the graph incident to at least one coverage edge and at least one travel edge. A *travel path* is a a sequence of travel segments and travel vertices connecting a pair of border vertices. Finally, an *optimal travel path (OTP)* is a travel path connecting a pair of border vertices such that it is the lowest cost path (of any type) between the vertices, and the vertices are not in the same cluster (i.e., there is no path consisting of just coverage segments between them). In other words, OTPs are shortest paths between clusters of coverage segments that do not cut through any part of a coverage cluster. Additionally, we modify the CPP algorithm to permit the use of travel edges as part of the shortest path between the set of odd nodes. This allows the CPP to reason directly about the paths that cut through coverage clusters.

All the OTPs are computed by finding the lowest cost path p_{ij} (searching over the entire graph) between each pair of border vertices v_i and v_j in different clusters. If p_{ij} is a travel path, we save it as an OTP. If p_{ij} is not a travel path, then v_i and v_j do not have an OTP between them (i.e., p_{ij} = NULL). The OTPs become our partition set. The OTPs are unlabeled at the beginning of the algorithm. We iterate through the OTP set within the branch-and-bound framework. At each branch step, the algorithm generates a new subproblem by either including or excluding an OTP. At the beginning of the RPP algorithm shown in Algorithm 2, we assign cost 0 to the unlabeled OTPs and solve the problem with all the OTPs set as required edges using the CPP algorithm (lines 2,3). The problem and CPP cost are pushed onto a priority queue (line 4). The CPP cost is the lower bound on the problem since all the OTPs have zero cost. While the queue is not empty, the lowest cost subproblem is selected from the queue (lines 5,6).

Algorithm 2: Rural Postman Problem Algorithm

```
Input: $s$, start vertex
       $G = (C, T)$, graph where each edge has a label and a
       cost value. $C$ is the subset of coverage edges, and $T$ is
       the subset of travel edges
       $OTP$, subset of OTPs
Output: $P$, tour found or empty if no tour found
1  pq = [];
2  $G' = [G, OTP]$ where $\forall OTP$, cost($p_{ij}$) = 0
3  $P = CPP(s, G')$;
4  $AddToPQ(pq, [G', P])$;
5  while $!isEmpty(pq)$ do
6      $[G', P] = PopLowestCost(pq)$;
7      $p_{ij} = FindMaxOTP(G')$;
8      if $p_{ij} == []$ then
9          return $P$;
10     end
11     $G'' = IncludeEdge(G', p_{ij})$;
12     $P1 = CPP(s, G'')$;
13     $AddToPQ(pq, [G'', P1])$;
14     $G'' = RemoveEdge(G', p_{ij})$;
15     $P2 = CPP(s, G'')$;
16     $AddToPQ(pq, [G'', P2])$;
17 end
18 return []
```

For the subproblem, the algorithm selects an unlabeled OTP p_{ij} with the highest path cost (line 7). By employing this strategy of choosing the OTP with the highest path cost, our aim is to increase the lower bound by the largest amount, which may help focus the search to the correct branch and prevent extraneous explorations. Once an OTP p_{ij} is selected, two branches are generated; the first branch includes p_{ij} in the solution (line 11), this time with the real path cost assigned, and the second branch omits p_{ij} from the solution (line 14). A solution to each branch is found using the CPP algorithm (lines 12,15). Because each solutions is generated with a cost of 0 assigned to the unlabeled OTPs in the subproblem, the costs of the inclusion and exclusion CPP solutions represent lower bounds on the cost of the RPP with and without using p_{ij} for travel, respectively. These new subproblems are added to the priority queue (lines 13, 16), and the algorithm iterates until the lowest cost problem in the queue contains no OTPs (line 8). The solution to this problem is the optimal solution to the RPP since it has either included or excluded every single OTP in the solution, and has a path cost that is equal to or lower than the lower bounds of the other branches. The branch-and-bound algorithm for the RPP is an exponential algorithm with a complexity of O($|V|^3 2^t$) where t is the number of OTPs and $|V|$ is the number of vertices in the graph.

C. Online Changes

Dynamic changes occur when the environment differs from the original map. There are two categories of planners that handle these differences. Contingency planners model the uncertainty in the environment and plan for all possible scenarios. Assumptive planners [22] presume the perceived world state is correct and plan based on this assumption. If disparities arise, the perceived state is corrected and replanning occurs. We choose to use the lower complexity assumptive planning in order to generate solutions quickly.

In our planner, an initial plan is found based on the graph of the environment. As the vehicle uncovers differences between the map and environment during traversal, the algorithm propagates them into the graph structure. This may require a simple graph modification such as adding, removing, or changing the cost of an edge. But it can also result in more significant graph restructuring. As mentioned earlier, these changes may convert the initial planning problem into an entirely different problem.

For the coverage problems we are addressing in this work, most changes to the environment are discovered when the robot is actively traversing the space. These online changes are typically detected when the robot is not at the starting location, but at a middle location along the coverage path. At this point, some of the edges have already been visited. Because it is not necessary to revisit the edges that are already traversed, the visited edges in the previous plan are converted to travel edges. As shown in Algorithm 3, if the unvisited edges in the new problem are connected, we run the CPP algorithm; otherwise, we run the RPP.

When environmental changes are found online, replanning is done on the updated graph with different starting and ending vertices. To remedy this disparity, we add an artificial edge from the current vehicle location c to the ending vertex s in the graph (line 3). This edge (c, s) is assigned a large cost value to prevent it from being doubled in the solution. Using this modified graph, a tour from s to s is found. The edge (c, s) is then deleted from the graph and from the tour (line 11). The algorithm adjusts the coverage path to start at the current location and travel to the end location (line 12). The table below shows the details for adjusting the path. The terms on the left hand side indicates the four possible initial path configurations where s→c indicates an edge. The terms in the middle are the procedures for adjusting the specific path, and the terms on the right are the final paths returned by the coverage algorithm.

s...s→(c...s)	⇒	(c...s)→s...s	⇒	c...s...s
(s...c)→s...s	⇒	Reverse(s...c)→s...s	⇒	c...s...s
s→(c...s)	⇒	(c...s)→s	⇒	c...s
(s...c)→s	⇒	Reverse(s...c)→s	⇒	c...s

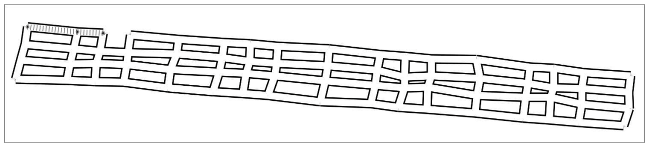


Fig. 3. During traversal of the CPP solution, the robot discovers that the third edge in the path is missing and ends the traversal. Dotted lines denote edges that are traversed once and solid lines denote edges that are traversed more than once.

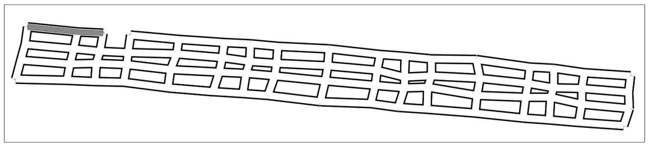


Fig. 4. To prevent visiting edges that have been already traversed, the graph is modified by setting the traversed edges to be travel edges which are represented by gray lines.

We now step through an example that illustrates the online coverage algorithm. In Figure 2(b), a CPP solution of the example graph is shown. In Figure 3, the vehicle traverses the initial CPP path until it discovers the third edge in its path is missing. The vehicle does not know about this change until it reaches the previous edge. The algorithm then sets the traversed edges in the path to be travel (Figure 4) and replans a new coverage tour shown in Figure 5. However, it encounters another missing edge as it travels along the new path. The traversed edges in the previous path are also converted to travel edges (Figure 6) and a final plan is found (Figure 7).

Algorithm 3: Online Coverage Algorithm

```
Input: $s$, start vertex,
       $c$, current vertex,
       $G = (C, T)$, graph where each edge has a label and a
       cost value. $C$ is the subset of coverage edges, and $T$ is
       the subset of travel edges
       $OTP$, subset of OTPs
Output: $P$, tour found or empty if no tour found
1  $G' = G$;
2  if $c \neq s$ then
3  |   $G' = [G, (c, s, INF)]$;
4  end
5  if $IsConnected(C)$ then
6  |   $P = CPP(s, G')$;
7  else
8  |   $P = RPP(s, G', OTP)$;
9  end
10 if $c \neq s$ and $P \neq []$ then
11 |   $RemoveEdge(P, (c, s, INF))$;
12 |   $AdjustPath(P, c, s)$;
13 end
14 return $P$;
```

D. Farthest Distance Heuristic

In our algorithm, when a robot encounters a change at a particular node, these changes are obstacles which prevent the robot from completing the current coverage solution. As a result, the problem needs to be re-solved with the previously

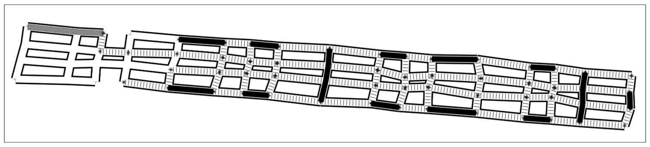


Fig. 5. The planner replans a new tour for the modified graph starting at the current location. During its new traversal, the robot discovers another edge in the path is missing and stops traversal.

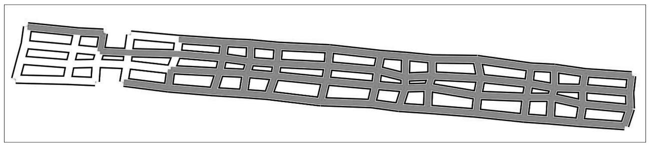


Fig. 6. For replanning, the traversed edges are again set to be travel edges, denoted by gray lines, in the graph.

traversed edges converted to travel edges. The number of optimal travel paths can be very large if the travel edges break the coverage subgraph into a large number of connected components. Therefore, to maintain efficiency, we want to keep the number of coverage clusters close to one. Ideally, we want the coverage subgraph and travel subgraph to be mutually independent. This would ensure that the required subgraph is connected, which is important for maintaining the coverage problem as a CPP, and maximize the likelihood that when the next change is detected, the travel edges do not disconnect the coverage components.

In the CPP algorithm, we use the End-Pairing technique to generate the Eulerian cycle from the graph. The algorithm consists of two steps. First, it builds cycles that intersect at at least one vertex. Next, the cycles are merged together two at a time by adding one cycle onto another at the intersecting vertex. The cycle building step is shown in Algorithm 4. During each step of the algorithm, edges are added to a path sequence and removed from the graph until the starting node of the path is encountered. In the original End-Pairing algorithm, the heuristic for choosing the next edge to add to the sequence consisted of picking a random edge incident to the current node.

To maintain a small coverage component set, we intuitively want to choose edges in such a way that the path travels away from the start and then travels back always visiting the farthest unvisited edges until it reaches the start. Essentially the coverage path should be always walking along the boundary of the coverage subgraph. This will allow the edges around the start to be as connected as possible while separating the coverage and travel subgraphs. This idea is translated into the Farthest Distance heuristic shown in equation 1 where s is the start node, i is the current node, $\{i, j\}$ is the set of edges incident to i, and D calculates the number of edges in the shortest path between s and j. To reduce computation time, we used D* Lite [23] to compute $D(s, j)$ since s is the same for each cycle generation step. In our CPP algorithm, we modified the End-Pairing algorithm to use the Farthest Distance heuristic to choose the next edge to add to the path.

$$e = argmax_{\forall\{i,j\}} D(s, j) \tag{1}$$

Algorithm 4: Cycle Building Algorithm

```
Input: $s$, start vertex,
       $G$, graph
Output: $C$, cycle found
1  $C = [s]$;
2  $i = s$;
3  $e = NextEdgeHeuristic(G, s, i)$;
4  $i = OtherEndPoint(e, i)$;
5  while $i \neq s$ do
6      $e = NextEdgeHeuristic(G, s, i)$;
7      $i = OtherEndPoint(e, i)$;
8      $C = [C; i]$;
9      $RemoveEdge(G, e)$
10 end
11 return $C$;
```

III. Comparison Tests

The goal of our testing is to compare the Farthest Distance heuristic against the original heuristic of randomly selecting a neighboring edge as the next edge. For the tests, we vary four different parameters: traversal heuristic, graph, starting node s, and set of changes. At the beginning of the test, the graph is connected and has no travel edges. Using the CPP algorithm, a tour is computed. The test simulates a robot executing the tour starting at node s. If along the execution, the next edge to be traversed is in the change set, that edge is considered blocked. It is removed from the graph, and the graph is updated to reflect the visited edges and current vertex location. The updated graph is either a CPP or RPP, and the coverage algorithm calls the corresponding method to find a new solution. The execution is simulated again until another edge along the new path is found to be blocked or the traversal is completed.

We conducted two sets of tests. The first test set evaluated the two heuristics on rectilinear graphs, and the second set evaluated the heuristics on a real-world road network. We will first explain the procedure for each test set, and then present the metrics for comparing the two heuristics. The code ran on a machine with a 3.8GHz Intel Xeon processor with 5GB of RAM.

A. Rectilinear Graphs

Rectilinear graphs are graphs where the vertices were generated uniformly in the plane. Edges connect one vertex to its four closest neighbors, and are vertical and horizontal lines that

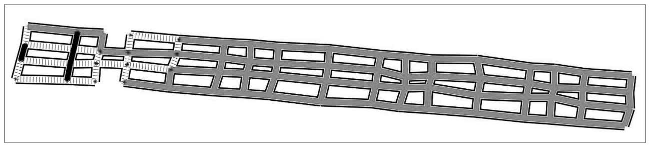


Fig. 7. The final path is replanned for the updated graph. This path starts at the node where the previous traversal ended.

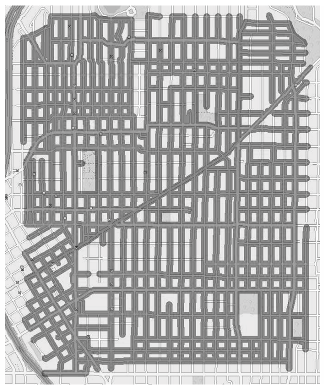


Fig. 8. This is the road network we used as our real-world example. This network covers an area of 1.5 miles by 2 miles. The edges and nodes included in our graph are highlighted in gray.

represent the Euclidean distance between the vertices. In other words, these graphs consist of a regular pattern of nodes and edges. We chose rectilinear graphs because they are similar to the networks used in many coverage applications. For example, in street coverage, the graph layout is similar to a grid where all streets (edges) meet at intersection points (nodes), and most intersections are four ways.

For our testing, three graph sizes were used; the sizes are $|V| = \{10 \times 10, 14 \times 14, 17 \times 17\}$. Five change sets were randomly generated for each of the three graphs. The change sets consisted of thirty different edges. We chose thirty because it is a significant number of changes (10% to 30% of the total edges in the graphs), but is a small enough number to keep computation low. For each of the graph-changeset combinations, we varied the starting node over all the nodes in the graph to avoid bias. Therefore, the coverage algorithm was called $3 \times 5 \times 30 \times |V|$ times for each of the two heuristics where $|V|$ is the number of nodes in the graph.

B. Real-world Example

The real-world example we used for testing is the road network of an urban neighborhood obtained from a dataset gathered by Newson and Krumm [24]. The network is shown in Figure 8. We converted the network into a graph with 764 vertices and 1130 edges. We generated five sets of changes for the data. Each change set consisted of five edges. Because the graph was so large, instead of varying the starting node over all nodes in the graph, we sampled a set of fifty distinct nodes for each change set. The samples were consistent for the two heuristics. In total, this test set yielded $5 \times 5 \times 50$ replans for the two heuristics.

C. Metrics

During each call to the coverage algorithm, the following items were measured: the number of connected coverage components in the graph, the number of optimal travel paths in the partition set, the number of branches in the search tree, the percentage of replanning calls that are CPP calls rather than RPP calls, and the computation time in seconds. Since the number of replanning calls was large, we placed a time limit on the branch-and-bound algorithm. The time limit was 70 seconds for rectilinear graphs and 100 seconds for the road network. If a replan was unable to produce a coverage path within the time limit, the particular test set failed.

IV. Results

Before presenting the results, we first highlight some aspects of the graphs to supplement the results. This information is shown in Table I. The first two columns display the number of nodes and edges in the graphs. The last two columns show the mean and standard deviation of the degrees of the vertices.

TABLE I

$\|V\|$	$\|E\|$	Mean Degree	Std Dev
10×10	180	3.6	0.57
14×14	364	3.7	0.50
17×17	544	3.8	0.47
764	1130	2.96	1.00

Results from the testing are presented in Tables II through V. The first table shows the average percentage of calls to the CPP algorithm over all trials, average computation time for successful trials, and the success percentage. The success percentage is the average number of successful replans over the total number of trials for each graph-heuristic combination. The second table shows the number of coverage components in the problem for each replan, number of OTPs when the number of components is greater than 1, and number of branches in the branch-and-bound tree for each RPP call. These values are computed only on the successful trials since the failed trials never return a solution path. Each column contains two numbers. The first number in the column represents the average for the metric and the second number represents the maximum.

A. Rectilinear Graphs

For the rectilinear graphs, the Farthest Distance heuristic performed a factor of two better than the random heuristic in percentage of CPP calls as shown in Table II. Using the heuristic, on average 96% of the replans resulted in graphs where the coverage subgraph was connected (number of connected components is one) and the lower complexity CPP was used. For the random heuristic, roughly half of the graphs were CPP graphs meaning half were the harder RPP problems. Furthermore, with our traversal heuristic, when the RPP algorithm did get called, the run time for all trials was never more than 70 seconds which results in 100% success. While the random heuristic did well on the 10×10 rectilinear trials with 84% of the replans successful, as the graph size got larger, the computation time for replans took longer and the success percentage went down to roughly 43%.

Looking at Table III, we can see that on average, the Farthest Distance heuristic keeps the number of connected coverage components smaller. Furthermore, when branch-and-bound is needed, the heuristic generates problems with a smaller set of OTPs. This smaller set translates to a shallower search tree. In terms of computation times, the Farthest Distance heuristic performs roughly 150 times better than the random heuristic.

B. Real-world Example

For the road network, the Farthest Distance heuristic performed more than a factor of two better than the random heuristic in maintaining connectivity among the coverage edges as shown in Table IV. When using the random heuristic, due to the high number of calls to the RPP algorithm, the run time on the majority of the trials exceeded the time limit. In contrast, the Farthest Distance heuristic had 98.4% success rate. Part of the reason for the drastic difference in successes was due to the large size of the graph (this graph is more than twice the size of the 17×17 graph). This resulted in the RPP algorithm needing more computation time. As shown in Table V, the number of connected components was similar for both heuristics. For the branch-and-bound data, we can see that the number of OTPs for the Farthest Distance heuristic was half the number of OTPs for the random heuristic which led to a smaller search tree.

C. Evaluation of the Coverage Algorithm

So far, we have shown comparison results regarding the Farthest Distance heuristic and the random heuristic. Next, we want to show some results of the coverage algorithm. From the road network data, we calculated the ratio of the travel edges over the total number of edges in the graphs for all the trials. The mean of the distribution was 0.44 with a standard deviation of 0.28. This denotes the travel set ranged from almost zero to almost the entire graph. Next, we calculated the size of the OTP sets that corresponded to each travel set (Figure 9). The ratio of the OTPs to travel edges has a mean of 0.08 with a standard deviation of 0.14. This indicates that our coverage algorithm dramatically reduced the partition set given to the branch-and-bound algorithm through the use of OTPs.

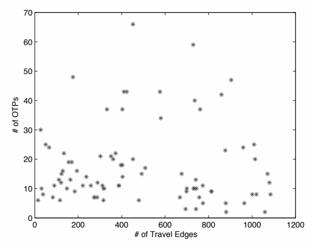


Fig. 9. This graph shows the correspondence between the number of travel edges, along the x-axis, and the number of OTPs, along the y-axis, for the road network data.

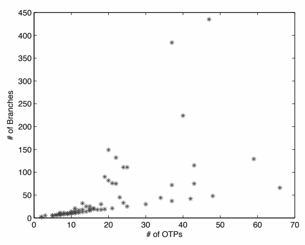


Fig. 10. This figure shows the relationship between the OTP set size, along the x-axis, and the number of branches in the search tree, along the y-axis, for the road network data.

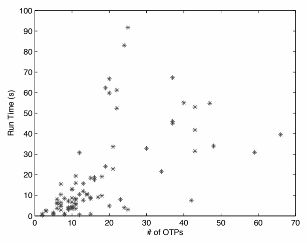


Fig. 11. This figure shows the relationship between the OTP set size, along the x-axis, and the computation time (shown in seconds), along the y-axis for the road network data.

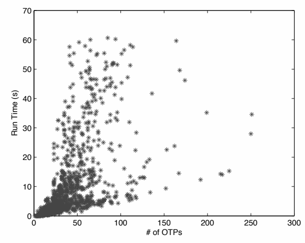


Fig. 12. This figure shows the relationship between the OTP set size, along the x-axis, and the computation time (shown in seconds), along the y-axis for the rectilinear graph data.

Next, we show how the smaller partition set affected the search tree and run time. From Figure 10, the maximum number of branches in the search tree was 435. Given that the largest number of travel edges was 1085, if we had used the travel edges as the partition set, the branch-and-bound problem corresponding to the maximum set would need to solve at least 1085 subproblems before finding the optimal solution. The smaller search tree translated into faster computation as evidenced in Figure 11. Similarly, the relationship between the number of OTPs and run time for the rectilinear graphs is shown in Figure 12.

D. Discussion

The results show that the Farthest Distance heuristic improves the percentage of CPP calls by a factor of two. The difference in the percentage between the rectilinear graphs and the road network is due to the lower degree of the vertices in the road network. As we can see from Figure 8, the road network is not a completely regular grid. There are intersections where more than four roads meet. Additionally, there are streets that dead end in one direction. One aspect of the graph that the heuristic relies on is that there is a path from the start node to another node j in the graph, and a path from j

TABLE II

Computational results I for Rectilinear Graphs

$\|V\|$	Heuristic	CPP Calls	Time(s)	Success Percentage
10×10	Random	46.67%	1.68	84.21%
	FarDist	92.19%	0.01	100.0%
14×14	Random	55.18%	3.02	51.08%
	FarDist	97.63%	0.02	100.0%
17×17	Random	59.59%	3.25	42.57%
	FarDist	98.66%	0.02	100.0%

TABLE III

Computational results II for Rectilinear Graph

$\|V\|$	Heuristic	Components	OTPs	Branches
10×10	Random	2.11, 9.0	35.62, 153.2	352.45, 7823.8
	FarDist	1.08, 3.8	9.42, 38.8	15.86, 268.6
14×14	Random	1.67, 7.2	38.96, 204.4	276.03, 3773.6
	FarDist	1.03, 3.6	11.30, 36.6	24.42, 324.0
17×17	Random	1.52, 6.6	38.79, 244.2	182.52, 2135.2
	FarDist	1.01, 2.8	11.97, 30.4	18.66, 106.2

TABLE IV

Computational results I for Road Network graph

$\|V\|$	Heuristic	CPP Calls	Time(s)	Success Percentage
764	Random	34.39%	12.63	30.40%
	FarDist	87.25%	1.14	98.40%

TABLE V

Computational results II for Road Network data

$\|V\|$	Heuristic	Components	OTPs	Branches
764	Random	1.62, 4.0	23.59, 56.6	53.47, 326.4
	FarDist	1.14, 2.8	12.44, 40.0	18.65, 63.6

back to the start. This assumption depends on path redundancy in the graph which may not hold for all graphs. For example, in a graph that contains bridges, the coverage problem will almost definitely become the more complex RPP when graph changes occur. This is due to the fact that converting a bridge edge to a travel edge will usually separate the start node and the remaining unvisited edges into different components. Hence, covering a graph with multiple bridges will limit the performance of not only the Farthest Distance heuristic, but also the random heuristic.

As shown in the results, the coverage algorithm greatly reduces the size of the partition set used to search for a solution. By limiting the branch-and-bound algorithm to only reason about the travel edges that make up the optimal path between boundary nodes, the faster CPP can be used to evaluate the remaining travel edges. For the RPP algorithm, the results show lower computation time on rectilinear graphs than for the road network graph. This is due to the smaller problem sizes of the rectilinear graphs. The larger problem size not only increases the run time of the CPP algorithm, but it also increases the time it takes to generate the OTP set.

V. Conclusion and Future Work

We present a general approach to the online coverage problem with an incomplete prior map. By representing the environment as a graph, our algorithm leverages the lower complexity Chinese Postman algorithm to produce optimal coverage solutions. This algorithm enables a faster run time by reducing the partition set to sequences of edges rather than single edges. Our algorithm is highly efficient when the number of OTP edges is small compared to the number of travel edges since it can exploit the faster CPP algorithm. This means a better connected graph can greatly decrease the computation time of our general coverage algorithm.

To improve the connectivity of the coverage graph when changes occur, we introduce a heuristic for generating the coverage path. Results from testing show that the Farthest Distance heuristic performs much better than the random heuristic in keeping the problem polynomial. One of the benefits of the Farthest Distance heuristic is that it is independent of our coverage algorithm. For example, it can be added to any coverage approach, either optimal or approximate, as a way to lower the complexity of the coverage problem when graph changes occur online.

For future work, we plan to test this method on a physical robot with real-time sensor updates to gather practical results of our approach. We also plan to extend this approach to directed and mixed graphs. Finally, we intend to investigate the potential of using the Farthest Distance heuristic on additional arc-routing problems.

Acknowledgment

This work was sponsored by the U.S. Army Research Laboratory, under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

References

[1] Howie Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113 – 126, 2001.

[2] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

[3] Howie Choset, Sean Walker, Kunnayut Eiamsa-Ard, and Joel Burdick. Sensor-Based Exploration: Incremental Construction of the Hierarchical Generalized Voronoi Graph. *The International Journal of Robotics Research*, 19(2):126–148, 2000.

[4] E.U. Acar, H. Choset, and Ji Yeong Lee. Sensor-based coverage with extended range detectors. *IEEE Transactions on Robotics*, 22(1):189–198, Feb. 2006.

[5] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[6] David S. Johnson and Lyle A. Mcgeoch. The traveling salesman problem: A case study. In *Local Search in Combinatorial Optimization*. Wiley and Sons, 1997.

[7] H. A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

[8] H. A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

[9] M Guan. Graphic programming using odd and even points. *Chinese Mathematics*, 1:273–277, 1962.

[10] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, July 1998.

[11] C.G. Sorensen, T. Bak, and R.N. Jorgensen. Mission planner for agricultural robotics. *Agricultural Engineering International*, 2004.

[12] Gianpaolo Ghiana and Gilbert Laporte. A branch-and-bound algorithm for the Undirected Rural Postman Problem. *Mathematical Programming*, 87(3):467–481, 2000.

[13] Gilbert Laporte. Modeling and solving several classes of arc routing problems as traveling salesman problems. *Comput. Oper. Res.*, 24(11):1057–1061, 1997.

[14] G. N. Frederickson. Approximation algorithms for some routing problems. *Mathematical Programming*, 40:225–246, 1979.

[15] Alain Hertz, Gilbert Laporte, and Pierrette Nanchen Hugo. Improvement procedures for the undirected rural postman problem. *INFORMS J. on Computing*, 11(1):53–62, 1999.

[16] Luís M. Moreira, José F. Oliveira, A. Miguel Gomes, and J. Soeiro Ferreira. Heuristics for a dynamic rural postman problem. *Comput. Oper. Res.*, 34(11):3281–3294, 2007.

[17] Jack Edmonds and Ellis Johnson. Matching, Euler tours, and the Chinese Postman. *Mathematical Programming*, 5(1):88–124, 1973.

[18] Edward. Minieka. *Optimization algorithms for networks and graphs*. M. Dekker, New York :, 1978.

[19] Harold Neil Gabow. *Implementation of algorithms for maximum matching on nonbipartite graphs*. PhD thesis, Stanford, CA, USA, 1974.

[20] E. Rothberg. Implementation of H. Gabow's weighted matching algorithm (1992). ftp://dimacs.rutgers.edu/pub/netflow/matching/weighted/.

[21] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[22] Illah Nourbakhsh and Michael Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal*, 3(1):49–67, 1996.

[23] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 968–975, 2002.

[24] Seattle road network data. http://research.microsoft.com/en-us/um/people/jckrumm/MapMatchingData/data.htm.