Technical Report: Swarm Robots EECE 2560: Fundamentals of Engineering Algorithms

William Fox, Ishaan Desai, Jack Gladowsky
Department of Electrical and Computer Engineering
Northeastern University
`fox.wi@northeastern.edu`
`desai.is@northeastern.edu`
`gladowsky.j@northeastern.edu`

December 5, 2024

Contents

1 Project Scope

The aim of this project is to design a system that can explore a graph to retrieve objects throughout the graph. The robots begin exploring the graph while sending the node data back to the controller that is mapping the graph. When a robot comes across an object, it retrieves the object back to the control node. The project objectives are:

The project's main objectives are:

- To randomly create a graph with different nodes such as blocking nodes, object nodes, and control node.
- To allow the robots to map the graph by sending node data to a controller.
- To create an algorithm for the robots to get back to the controller node with the object efficiently.

The expected results include a fully functional algorithm, a detailed technical report, and a final presentation summarizing the project's findings.

2 Project Plan

2.1 Timeline

The project is divided into phases, each with specific deliverables:

- **Week 1**: (10/14 - 10/20): Start development, set up repository and development environment.
- **Week 2**: (10/21-10/28): Verify graph generation, create classes to represent objects for attributes such as location and status (retrieved or not retrieved). Implement basic functionality for robots to traverse the graph, update the controller with node information, and mark the explored nodes.
- **Week 3**: (10/29 - 11/3): Verify robot graph mapping works, begin robot object retrieval system, verify that new classes are working as expected.
- **Week 4**: (11/4 - 11/10): Establish functionality for robot collaborative object retrieval.
- **Week 5**: (11/11 - 11/17): Ensure that all aspects of the projects are functional and cohesive. Work on the final presentation and report.

2.2 Milestones

Key milestones include:

- Established ability to generate a graph randomly (October 16th)

- Verified graph generation with different node configurations testing (October 17th)
- Created necessary classes to represent objects, ensuring attributes like location and status (retrieved or not) were included (October 21st)
- Established functionality for robots to traverse the the graph and update the controller with node locations (October 27th)

3 Team Roles

- **Team Member 1**: Jack Gladowsky - Algorithm Development and Optimization, Complexity Analysis, Code Review
- **Team Member 2**: William Fox - Algorithm Development and Optimization, Testing, Debugging, Code Review
- **Team Member 3**: Ishaan Desai - Algorithm Development and Optimization, Documentation, Code Review

4 Methodology

4.1 Pseudocode and Complexity Analysis

We are generating graphs, so this is the primary data structure of choice. The above flowchart provides a brief overview of the graph that is created for the robots to traverse. The nodes will be generated and subsequently, a graph of the nodes will be created by setting "neighbor" nodes for each node. Technically, this is a bidirectional, fully-connected graph, meaning that all nodes are reachable from any other node and that all connections between nodes can be traversed both ways. We determined that this was the best method for our implementation to maintain simplicity and represent a realistic scenario. The robots are controlled by a controller, which will give the robot traversal instructions. The robot traversal algorithm is explained further below.

The graph generation pseudo code is shown in the appendix as algorithm 1. The time complexity of the graph generation is O(n) where n is numNodes, since the maximum number of neighbors per node is bounded by 4, making the nested loops effectively linear.

The robot move pseudo code is shown in the appendix as algorithm 2. The time complexity is O(n2) where n is the number of neighbors of the current node, as we need to check each neighbor against the visited list.

The controller map psuedo code is shown in the appendix as algorithm 3. The time complexity is O(r×n) where r is the number of robots and n is the number of nodes in the graph, as each robot may need to visit most or all nodes in the graph during the mapping process.

4.2 Algorithm Design

Graph Generation:

The graph generation algorithm works by creating a set number of nodes and for each, assigns a type. The different types are clear, obstacle, object, or control and are assigned based on random number probabilities. Each node then connects to a number of other “neighbor” nodes, to for a bidirectional, connected graph that ensures each node has at least one neighbor. This is done using a random selection process that selects nodes to be added as neighbors for each new node, and ensures nodes are not added as neighbors more than once by checking if the node already exists as a neighbor before adding the node as a neighbor. The number of neighbor nodes may be set to a certain number to reduce complexity in the future. Lastly, a control node is set randomly to one of the pre-generated nodes, providing a “home” from which the robots will spawn and begin to explore the graph.

Robot Traversal:

Each robot starts from the home node, where the controller is, and initializes its movement to an adjacent, unexplored node. Movement from node to node is determined by a depth-first search approach, with each robot selecting an unexplored neighboring node from its current position. This choice is made by accessing the node’s list of neighbors, which is shared among all robots to prevent duplicate visits. Each robot keeps a local record of visited nodes and, upon reaching a new node, sets a ”visited” or ”explored” flag on that node.

To mark each node’s exploration status, robots use a shared data structure (stored at the controller) where they update each visited node’s state to “explored.” As each robot moves to a new node, it first checks if the node is already flagged as “explored” in this shared map to prevent revisiting nodes. When encountering a node, the robot inspects its type, updating the node status as ”clear,” ”obstacle,” or ”object.” For obstacles, the robot marks the node as impassable, which other robots will read and avoid, optimizing their exploration routes.

This shared map will be updated by the controller whenever a robot reports back to keep a record of the graph’s state. Robots will continue moving outward from their last explored node, revisiting nodes only when they need to backtrack. This exploration process repeats until all accessible nodes in the graph are marked, allowing the swarm to build a complete, dynamically updated map that serves as the foundation for further object retrieval or navigational tasks.

5 Results

We successfully developed an algorithm to generate random graph environments that consist of various node types. The algorithm is very efficient and can generate a graph with 100,000 nodes in under 3 seconds. Each node in the graph also successfully gets assigned its neighbors and holds references to each neighbor node internally as shown in Figure 1 in Appendix B. We then developed robots that can traverse through the graph environment while simultaneously mapping it. The system also can handle mapping with multiple robots through the use of a system controller that handles storing the representation of the map that all the robots use to navigate through the environment. The controller is also in charge of commanding the robots and telling them when and where they are able to navigate to.

6 Discussion

The implementation of the swarm robot system achieved the core objectives outlined in the project scope, with several notable observations and implications. The graph generation algorithm demonstrated excellent performance, successfully creating complex environments with different node types (clear, obstacle, object, and control) while maintaining O(n) complexity. This efficiency is particularly significant for scaling the system to larger environments. The robot traversal and mapping system showed effective collaboration between multiple robots through the centralized controller. The depth-first search approach for exploration proved efficient in mapping unknown environments, though there were some trade-offs. While the algorithm ensures complete coverage of accessible nodes, the random selection of unvisited neighbors can sometimes lead to suboptimal paths. However, this randomness also helps distribute robots across different sections of the graph, potentially improving overall exploration speed. The system's ability to handle multiple robots simultaneously while maintaining a consistent shared map through the controller demonstrates the scalability of the architecture. The O(r×n) complexity of the controller mapping algorithm shows a linear scaling with the number of robots, suggesting good efficiency for larger swarms. However, this could potentially become a bottleneck in very large environments with many robots due to the centralized nature of the controller. When comparing our results with the initial objectives, the system successfully achieved:

- Random graph generation with different node types
- Efficient robot exploration and mapping
- Coordination between multiple robots

7 Conclusion

This project successfully demonstrated the implementation of a multi-robot exploration and mapping system using a graph-based environment representation. The algorithms developed show promising results in terms of both performance and scalability, with linear time complexity for graph generation and efficient exploration strategies for multiple robots. The graph-based approach to environment representation proved to be a flexible and efficient solution, allowing for easy expansion and modification of the environment while maintaining connectivity information. The centralized controller architecture, while potentially limiting in extremely large-scale deployments, provided a reliable mechanism for coordinating multiple robots and maintaining a consistent map of the environment. Future improvements could focus on:

- Implementing more sophisticated path planning algorithms to optimize robot movement
- Adding dynamic obstacle handling capabilities
- Optimizing the object retrieval process through cooperative robot behaviors

The project provides a solid foundation for further research in swarm robotics and multi-agent exploration systems, particularly in applications requiring coordinated exploration and object retrieval in unknown environments.

8 References

References

[1] IEEE, "Swarm Robotics Applications," [Online]. Available: `https://ieeexplore.ieee.org/document/`.

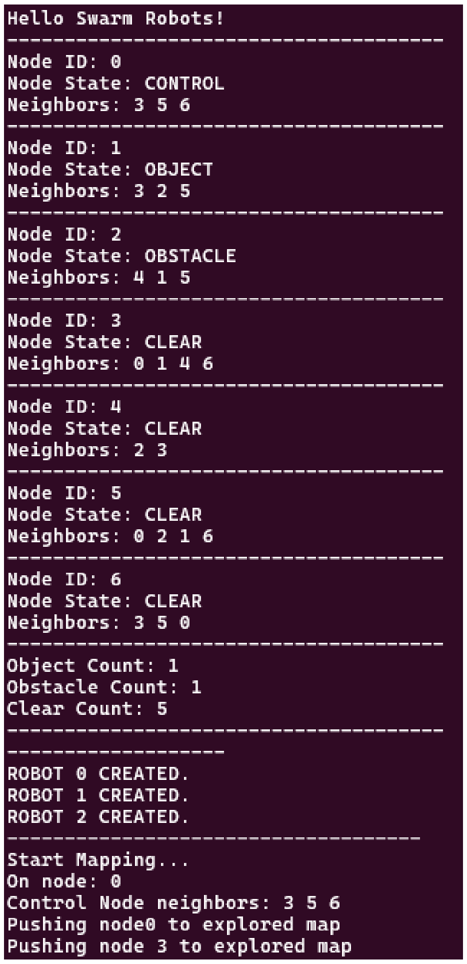
[2] IoT For All, "Swarm Robotics: Applications, Benefits, and Challenges," [Online]. Available: `https://www.iotforall.com/swarm-robotics-applications`.

A Appendix A: Code and Psuedocode

B Appendix B: Additional Figures

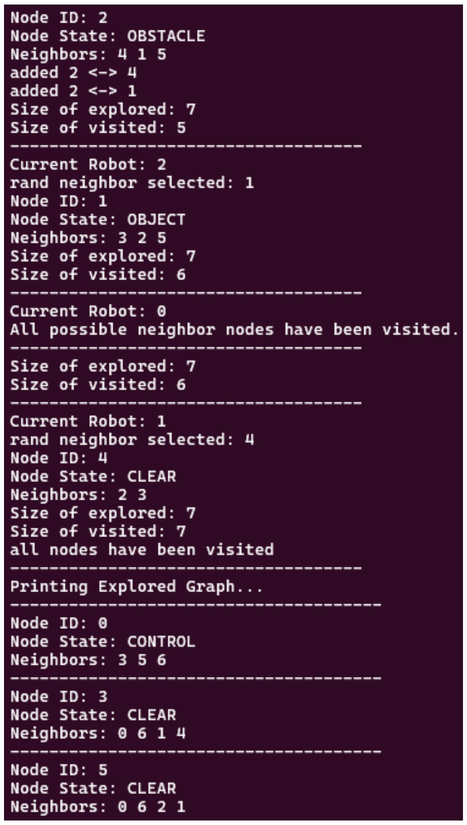
Algorithm 1: Graph Constructor

```
Input: Number of nodes, numNodes
Output: Constructed graph with randomly assigned states and
  neighbors
Begin Algorithm: Graph Constructor
Initialize empty list  nodes
obsCount, clrCount, objCount ← 0
for $i = 0$ to $numNodes - 1$ do
    $random \leftarrow \mathrm{random}(1, 100)$
    if $random \leq 10$ then
        $nodes[i] \leftarrow \mathrm{new\ GraphNode}(i, OBSTACLE)$
        $obsCount \leftarrow obsCount + 1$
    else if $random \leq 90$ then
        $nodes[i] \leftarrow \mathrm{new\ GraphNode}(i, CLEAR)$
        $clrCount \leftarrow clrCount + 1$
    else
        $nodes[i] \leftarrow \mathrm{new\ GraphNode}(i, OBJECT)$
        $objCount \leftarrow objCount + 1$
    end
end
$controlPointIndex \leftarrow \mathrm{random}(0, numNodes - 1)$
$nodes[controlPointIndex].state \leftarrow CONTROL$
$controlPoint \leftarrow nodes[controlPointIndex]$
for $i = 0$ to $numNodes - 1$ do
    $numNeighbors \leftarrow \mathrm{random}(1, 4)$
    for $j = 0$ to $numNeighbors - 1$ do
        $neighbor \leftarrow \mathrm{random}(0, numNodes - 1)$
        if $neighbor \neq i$ then
            $exists \leftarrow false$
            for $k = 0$ to size of nodes[i].neighbors $- 1$ do
                if $nodes[i].neighbors[k].NodeID = neighbor$ then
                    $exists \leftarrow true$
                    break
                end
            end
            if $\neg exists$ then
                $nodes[i].neighbors.\mathrm{append}(nodes[neighbor])$
                $nodes[neighbor].neighbors.\mathrm{append}(nodes[i])$
            end
        end
    end
end
Return nodes
End Algorithm: Graph Constructor
```



```
Hello Swarm Robots!
--------------------------------------
Node ID: 0
Node State: CONTROL
Neighbors: 3 5 6
--------------------------------------
Node ID: 1
Node State: OBJECT
Neighbors: 3 2 5
--------------------------------------
Node ID: 2
Node State: OBSTACLE
Neighbors: 4 1 5
--------------------------------------
Node ID: 3
Node State: CLEAR
Neighbors: 0 1 4 6
--------------------------------------
Node ID: 4
Node State: CLEAR
Neighbors: 2 3
--------------------------------------
Node ID: 5
Node State: CLEAR
Neighbors: 0 2 1 6
--------------------------------------
Node ID: 6
Node State: CLEAR
Neighbors: 3 5 0
--------------------------------------
Object Count: 1
Obstacle Count: 1
Clear Count: 5
--------------------------------------
-------------------
ROBOT 0 CREATED.
ROBOT 1 CREATED.
ROBOT 2 CREATED.
------------------------------------
Start Mapping...
On node: 0
Control Node neighbors: 3 5 6
Pushing node0 to explored map
Pushing node 3 to explored map
```

Figure 1: Graph Generation Output



```
Node ID: 2
Node State: OBSTACLE
Neighbors: 4 1 5
added 2 <-> 4
added 2 <-> 1
Size of explored: 7
Size of visited: 5
------------------------------------
Current Robot: 2
rand neighbor selected: 1
Node ID: 1
Node State: OBJECT
Neighbors: 3 2 5
Size of explored: 7
Size of visited: 6
------------------------------------
Current Robot: 0
All possible neighbor nodes have been visited.
------------------------------------
Size of explored: 7
Size of visited: 6
------------------------------------
Current Robot: 1
rand neighbor selected: 4
Node ID: 4
Node State: CLEAR
Neighbors: 2 3
Size of explored: 7
Size of visited: 7
all nodes have been visited
------------------------------------
Printing Explored Graph...
--------------------------------------
Node ID: 0
Node State: CONTROL
Neighbors: 3 5 6
--------------------------------------
Node ID: 3
Node State: CLEAR
Neighbors: 0 6 1 4
--------------------------------------
Node ID: 5
Node State: CLEAR
Neighbors: 0 6 2 1
```

Figure 2: Graph Traversal and Mapping Output

```
Algorithm 2: Robot Move
Input: List of visited nodes, visited
Output: Boolean indicating successful move
Begin Algorithm: Robot Move
Initialize empty list  unvisitedNeighbors
for $i = 0$ to size of currNode.neighbors $- 1$ do
    if currNode.neighbors[i] $\notin$ visited then
        $unvisitedNeighbors$.append(currNode.neighbors[i])
    end
end
if size of unvisitedNeighbors $> 0$ then
    $randNeighborNum \leftarrow \text{random}(0, \text{size of unvisitedNeighbors} - 1)$
    $randNeighbor \leftarrow unvisitedNeighbors[randNeighborNum]$
    $prevNode \leftarrow currNode$
    $prevNodeStack$.push($prevNode$)
    $currNode \leftarrow randNeighbor$
    Return true
end
Return $false$
End Algorithm: Robot Move
```

Algorithm 3: Controller Map

```
Input: Number of robots, $numRobots$
Output: Explored map of environment
Begin Algorithm: Controller Map
$createRobots(numRobots)$
$mapping \leftarrow true$
$controlPoint \leftarrow$
  new GraphNode($graphMap.controlPoint.NodeID, graphMap.controlPoint.state$)
$exploredMap.controlPoint \leftarrow controlPoint$
$exploredMap.nodes$.append($controlPoint$)
$visitedNodes$.append($controlPoint$)
for $i = 0$ to $size\ of\ graphMap.controlPoint.neighbors - 1$ do
    $currNeighborCopy \leftarrow$
      new GraphNode($graphMap.controlPoint.neighbors[i].NodeID,$

      $graphMap.controlPoint.neighbors[i].state$)
    $controlPoint.neighbors$.append($currNeighborCopy$)
    $exploredMap.nodes$.append($currNeighborCopy$)
    $currNeighborCopy.neighbors$.append($exploredMap.controlPoint$)
end
$count \leftarrow 0$
while $mapping\ AND\ count < 10$ do
    for $i = 0$ to $size\ of\ robots - 1$ do
        if $NOT\ \ robots[i].move(visitedNodes)$ then
            $robots[i].moveBack()$
        end
        $updateExploredMap(robots[i])$
        if $size\ of\ visitedNodes = size\ of\ exploredMap.nodes$ then
            Return
        end
    end
    $count \leftarrow count + 1$
end
End Algorithm: Controller Map
```