

Spring 4-2017

Multi-Agent Path Planning for Locating a Radiating Source in an Unknown Environment

Stephanie M. Schwartz
Embry-Riddle Aeronautical University

Follow this and additional works at: <https://commons.erau.edu/edt>



Part of the [Mechanical Engineering Commons](#)

Scholarly Commons Citation

Schwartz, Stephanie M., "Multi-Agent Path Planning for Locating a Radiating Source in an Unknown Environment" (2017). *Doctoral Dissertations and Master's Theses*. 343.
<https://commons.erau.edu/edt/343>

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Doctoral Dissertations and Master's Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

MULTI-AGENT PATH PLANNING FOR LOCATING A
RADIATING SOURCE IN AN UNKNOWN ENVIRONMENT

by

Stephanie M Schwartz

A Thesis Submitted to the College of Engineering Department of Mechanical
Engineering in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Embry-Riddle Aeronautical University
Daytona Beach, Florida
April 2017


MULTI-AGENT PATH PLANNING FOR LOCATING A
RADIATING SOURCE IN AN UNKNOWN ENVIRONMENT

by

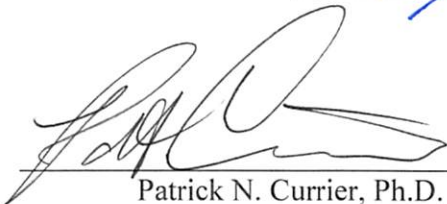
Stephanie M Schwartz

This thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. Eric J. Coyle, Professor, Daytona Beach Campus, and Thesis Committee Members Dr. Patrick Currier, Professor, Daytona Beach Campus, and Dr. Marc Compere, Professor, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Thesis Review Committee:



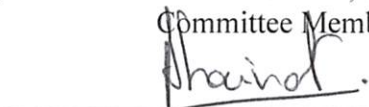
Eric J. Coyle, Ph.D.
Committee Chair



Patrick N. Currier, Ph.D.
Committee Member



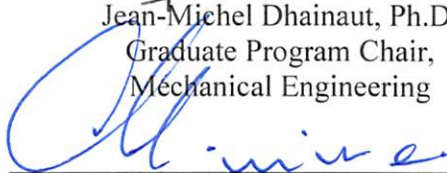
Marc D. Compere, Ph.D.
Committee Member



Jean-Michel Dhainaut, Ph.D.
Graduate Program Chair,
Mechanical Engineering



Charles F. Reinholtz, Ph.D.
Department Chair,
Mechanical Engineering



Maj Mirmirani, Ph.D.
Dean, College of Engineering



Christopher Grant, Ph.D.
Associate Vice President of Academics

4/24/17
Date

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Eric Coyle, for his guidance and support throughout the entire course of this project, from defining the original parameters of the research to finally getting it all down on paper. I particularly appreciate his patience through numerous iterations and his flexibility in allowing me to pursue parts of this research outside our original definition in order to satisfy my own curiosity. His help and advice have been invaluable to me both in pursuing this research and in professional and academic areas in general. I am especially grateful, as well, that he never gave up on me, even though at times it seemed like I would never finish.

I would also like to thank my committee members, Dr. Patrick Currier and Dr. Marc Compere, for their suggestions, and critiques. It is insightful questions that best clarify one's thoughts.

Abstract

Researcher: Stephanie M Schwartz

Title: Multi-Agent Path Planning for Locating a Radiating Source in an Unknown Environment

Institution: Embry-Riddle Aeronautical University

Degree: Master of Science in Mechanical Engineering

Year: 2017

A situation is addressed in which multiple autonomous agents are used to search an unknown environment for a target, the position and orientation of which is known with respect to each agent. A controlling framework is proposed to inform and coordinate the agents' movements in order to reduce the time required to locate the target. Four primary variables are considered: the cost function used to select the agents' paths, the number of agents in a given scenario, the distance over which the agents are assumed to communicate, and the size of the environment in which the agents are operating. It was found that a cost function that balances progress toward the target with exploration of the environment is generally most effective for all combinations of the other variables. More agents and greater communication are beneficial, to a point, in larger environments, although these may be less effective in smaller ones.

Thesis Review Committee	i
Acknowledgements	ii
Abstract	iii
List of Tables	vii
List of Figures	viii
Chapter	
I Introduction	1
Review of Relevant Literature	1
Multi-Agent Mapping	3
Multi-Agent Search	4
Multi-Agent Path Planning	5
Maze solving	6
A Proposed Hybrid Approach	8
Contributions	9
Definitions of Terms	10
II Methodology	13
Search Algorithm Assumptions	14
Agents	14
Communication	15
Map and Environment	15
Search Overview	15
Terminology	20
Search Algorithm	21

	Check for Re-Evaluation of Intermediate Target Node	23
	Continuously Updating Connectivity Graph	23
	Path Calculation and Identification of Unvisited Nodes...	23
	Agent Node Selection	24
	Node Selection Sequence.....	24
	Best Node Criteria.....	27
	A* Cost Function	27
	Dijkstra Cost Function	29
	Greedy.....	30
	Node Selection for Multiple Agents	30
	Update other Agents	31
III	Simulation Environment	32
	Justification	32
	Simulation Environment	33
	Assumptions.....	33
	Simulation Options	34
	Simulation Structure	35
	Agent Model	35
	Agent Starting Pose.....	38
	Obstacle Avoidance	40
	Controllers.....	41
	Termination Criteria.....	43
IV	Results.....	44

	Map Generation	44
	Case Generation	47
	Data Sets and Scenario Statistics	47
	Results and Discussion	49
	Unexpected Behaviors	57
V	Conclusions and Recommendations	61
	Conclusions	61
	Recommendations for Further Investigation	63
	Changes in Assumed Environment	63
	Maximum Efficiencies	64
	Non-Homogenous Agent Groups	64
	Agent Failure Rates	64
	Unclear Target Location	65
	References	66
	Appendices	
A	Graphical Simulation Environment	71
B	Mathematical Models	76
C	Full Map Exploration	79
D	Maps used for Data Collection	82

List of Tables

Table

1	Possible Values for Primary Simulation Variables.....	47
2	Statistics for First Data Set	48
3	Statistics for Second Data Set	49

List of Figures

Figure

1	Simulation Sequence.....	19
2	Overview of a Search in an Unknown Environment using a Continuously Updating Connectivity Graph.....	22
3	Node Selection Flowchart.....	26
4	Assumed Path for Agents when Turning.....	29
5	Agent Configuration	36
6	Agent Orientation.....	37
7	Multi-Agent Start Positions	39
8	Potential Field Calculated for Wall Avoidance	41
9	Example of Smallest Map Size.....	46
10	Example of Largest Map Size.....	46
11	Effect of Cost Function on Solve Time	50
12	Effect of Number of Agents on Solve Time - Data Set 1	51
13	Effect of Number of Agents on Solve Time (A* Cost Function and Specified Communication Distances) – Data Set 1	52
14	Effect of Communication Distance on Solve Time (A* Cost Function) – Data Set 1.....	53
15	Effect of Communication Distance on Solve Time – Data Set 1	53
16	Solve Time vs Communication Distance per Map Size for Specified Numbers of Agents	56
17	Looping Behavior	58

18	Less Efficient Path Behavior	59
19	Placing First Edge	72
20	New Node Horizontal or Vertical with Respect to Existing Node	73
21	Complete Connectivity Graph	73
22	Complete Map.....	74
23	Graphical Simulation Interface after Defining Start Node	75
24	Position and Orientation of a Fixed Wheel.....	76
25	Effect of Cost Function on Nodes Visited	80
26	Effect of Cost Function on Nodes Revealed.....	80
27	Class 4, Map 3 – 229 Total Nodes	91

Chapter I

Introduction

One of the primary uses for robotic systems is to carry out tasks in potentially dangerous environments. One such application, which is investigated in this thesis, is to search out a target at a known location but in an unknown environment so that it may be surveyed, retrieved, destroyed, or otherwise modified. If personnel or other machinery is needed to perform the task, the most efficient known path to the target should also be identified. Such a target is referred to a “radiating source” because it is reporting its location by emitting a signal that is detectable with the appropriate equipment. This signal may be a chemical spill or leak, a fire, a wireless signal, or any other source that emits energy that can be sensed. These sensors, the software necessary to identify and map obstacles, and proper communications equipment enable multiple agents to work in concert to complete an efficient search of the environment. While the particular sensors, obstacle avoidance algorithms, and communications equipment needed are very specific to each application, a general approach to the path planning and agent coordination problems that is extensible to a wide variety of applications can be devised. It is a solution to this aspect of the multi-agent search that is presented in this paper.

Review of Relevant Literature

In search applications the goal is often to minimize the time needed to locate or reach a target. While an intelligent single agent approach is expected to be faster than a random search, a team of agents has the potential to search the space more efficiently. The problems to be solved in such a case include coordinating the agents to achieve a balance of breadth and depth during the search, incorporating the agents’ shared

knowledge into each agent's path planning, minimizing path duplication among the agents, and selection of movement directions that increase the probability of locating the target.

These problems have been addressed individually in works involving exploration and mapping [1], [2], [3], [4], [5], [10], [11], [12], [13], [14], [15], [17], multi-agent search algorithms [18], [19], [20], [21], [22], [23], [31], [32], and path planning for cooperating groups [33], [44], [45]. In addition, this problem shares many characteristics with some approaches to maze-solving [46], [47], [51], [53], [54], [55], [56], [57], [58]. Problems that fall under exploration and mapping generally assume an unknown environment of unknown structure and have to goal of covering as much area or gaining as much information as possible about the environment in an efficient manner. In multi-agent searches, the environment is assumed to be known, at least in a general sense [18], [19], [20]. In these types of searches the target or targets are usually at unknown locations and are sometimes also an unknown quantity. Problems related specifically to path planning for multiple agents usually address the problem of how to move a number of agents from a number of specified starting points to given destinations in an efficient manner without causing the agents to collide, or in some cases, have intersecting paths [33], [44], [45]. Maze-solving problems often assume unknown environments and target locations, and sometimes make use of multiple agents, but most of the techniques employed are based in graph theory and do not allow all knowledge of the known area to be used when making path decisions [46], [47], [51], [53], [54], [55], [56], [57], [58]. The following sections provide a review of recent works in these areas and their relevance to the problem presented here.

Multi-Agent Mapping. Mapping and exploration methodologies involving multiple agents tend to follow one of several general approaches. In the first approach agents are assigned areas of the environment which they are responsible for uncovering [1], [2], [3]. In this approach, there are two levels of logic governing the mapping action. The first partitions the entire area and assigns agents to the partitions, while the second determines how the agents go about exploring their individual sections. Both [1] and [2] make use of a Voronoi diagram based on an uncertainty density distribution in the search space instead of distance. In [1] the nodes are the current positions of the agents, while in [2] the nodes are locations that are expected to reduce the map uncertainty by a maximum amount. In both cases agents move toward these maximum-information locations and each paper addresses a method of determining these locations. In these papers, a ‘search’ is performed by sensing the information present at locations near the agents. This may be done once the agent(s) have reached the intermediate target locations [1], [2] or at each time step while moving between target locations [1].

The second approach focuses on keeping the agents spread out. Although such area coverage is a consequence of the sectioning method described above, these dispersion algorithms have only one level of mapping/exploration logic and the communication between agents is simple. Alian et. al. [4] achieve dispersion through pheromone markers, while Ugur et. al. [5] makes use of the wireless signal intensity between agents. Many dispersion approaches seek to deploy sensors or agents for maximum coverage throughout an environment while maintaining communications [5], [10].

A third approach is to mathematically model the distribution of information or uncertainty throughout the map [1], [2], [11]. Baglietto et. al. [11] assign trajectories to the agents such that the uncertainty for the entire map is reduced by a maximum amount in the subsequent step(s). This is determined based on the predefined uncertainty density distribution. This uncertainty density approach is also used in [1] and [2] to define the map partitions.

A final approach to mapping problems is to have the agents remain on a frontier or boundary between explored and unexplored cells [12], [17]. This can be combined with other criteria such as dispersion [13], [14] or maintaining communication with nearby agents [12], [15].

Multi-Agent Search. One of the most common problems related to multi-agent searches is to locate a (sometimes unknown) number of targets at unknown locations within a known or partially known environment [18], [20]. In some cases [18], [19] the environment is known perfectly and the search algorithms are based on the likelihood of finding a target at a certain location. In [20] the general layout of the environment (a building) is known, but the targets (obstacles imposed by an emergency) alter the available paths. The premise of [20] is that if these obstacles can be found and reported, a central planner can update usable evacuation routes.

Another commonly seen scenario is a search of an unknown environment for one or more targets (or otherwise defined globally optimal solutions) [21], [31]. Many of the solutions to these types of problems use robotic swarms with agents running simple algorithms, but in communication with one another, as in Particle Swarm Optimization (PSO) [21], [22] and its myriad modified versions [23], [31]. Indirect communication

works here, too, as in [32] where agents mark the cells simply as ‘wall’, ‘explored’, and ‘visited’ (ie. dead end) while searching out victims and hazards.

A majority of the works, however, which investigate searching unknown environments for known and unknown targets deal with approaches to maze-solving, which is discussed in the section on Maze solving, below.

Multi-Agent Path Planning. A major area of research related to path planning with multiple agents is the problem of moving multiple agents through a shared environment between unique start and end points without causing a collision between them [33], [44]. This problem, specifically, is known as Multi-Agent Path Planning (MAPP). These problems and their solutions assume that an agent’s size is comparable to the size of the area represented by a map’s nodes or cells and, therefore, multiple agents cannot occupy the same node or cell. This is not always the case, since, in some instances, it may be reasonable for a node or cell to represent a relatively large area such as an intersection between two hallways or an open area between obstacles. In these cases it may still be desirable to plan each agent’s path using information about the other agents’ proposed paths, but the criteria may instead be to minimize time, distance, or some other cost as in some of the maze problems discussed below.

Path planning for multiple agents can also be done in two layers, as shown by Zhong [45]. In this problem multiple transport agents in an industrial setting plan individual routes between points based on a general knowledge of the environment layout. This is done without consideration for other agents. As the agents travel along these pre-determined paths they may encounter other agents or unexpected obstacles. In

these cases, separate algorithms resolve conflicts between agents and allow agents to navigate around localized obstacles.

Maze solving. The approaches to maze-solving address multiple aspects of the problem presented in this paper. Many maze-solving solutions exist for single agents in unknown map environments. Bakar et. al. [46] use combined left-hand and right-hand wall navigation rules to solve a reconfigurable maze. Their approach involves only a single agent and accounts for no information beyond the local wall configuration in making navigation decisions. Many approaches, such as those in [47], [51], use cell-based approaches such as flood-fill, Tarry's algorithm, and Trenaux's algorithm and variations of these methods to explore and solve maze problems with a single agent. Other methods include pheromone marking [51], wall-following [51], [53], and A* methods [54]. With these approaches, the focus is primarily on improving efficiency or on generalizing one of the existing approaches in order to guarantee or improve the likelihood of a solution.

Only a few methodologies deal with multiple agents in a maze of unknown configuration. Rahnama and Elçi [55] use multiple agents to explore an unknown labyrinth. The agents share information about the parts of the maze each agent has uncovered in order to create a master map that includes information about which paths (which direction out of which cells) have already been traveled. The environment representation is cellular and agents choose which direction to take out of their current cells by comparing the direction of the potential next cell with the goal cell and by how many times a particular path has been traveled. This approach facilitates the search function by ensuring that the agents spread out, but since the decision making process

only considers the next cell, it does not necessarily provide for the most intelligent dispersion. For instance, there may be instances where a discovered, but not yet explored, cell-direction would be accessible to a particular agent via an already visited cell-direction. This method does not have logic in place to allow the agent to travel a previously explored cell-direction in order to reach another cell that has a high probability of leading to the target. Instead the agent will move to a potentially less desirable adjacent cell that has not yet been visited. In other words, each agent only considers the local maze configuration when choosing how to move.

Kivelevitch and Cohen [56] use a similar approach to [55]. They use a modified version of Tarry's algorithm [57] to allow a team of agents to explore an unknown labyrinth and locate a target. As in [55], the agents share information about the environment as the maze is explored. The maze representation built in this approach is slightly different in that the values assigned to the directions from a cell can take on values between and including 0 to 1. The value starts at 1 and is reduced for each agent that takes a certain path. A value of 0 is assigned when an agent determines a path to be a dead end. In this way, the team can keep track of how many agents have taken a particular path and, since a higher number is more favorable, force later agents to take other paths even if the directional value for the path is favorable. This means that as more paths are explored, more and more cells will be marked as dead ends as it is determined that the cells they connect to do not lead to the goal. Since this implementation uses a tree-type map where there is only one possible path that leads to the goal, the agents will be forced down this path, given enough time. This method of blocking off cells also adds a level of knowledge beyond the immediate cells to the navigation process, although it

takes time to build up and is done by process of elimination rather than intelligent selection.

Rahnama et. al. [58] use Flood Fill and Modified Flood Fill algorithms adapted for multiple agents to explore and map an unknown maze. Since these approaches also give each cell a value based on the current best estimate of its distance from the goal, the sharing of information amongst agents is especially beneficial as information about wall placement further along a given path will affect the favorability of earlier cells.

A Proposed Hybrid Approach. In order to efficiently locate a target in an unknown environment, that environment should be explored as thoroughly as possible, in a minimum amount of time, and during the exploration paths that are more likely to lead to the target should be prioritized. These objectives can be addressed by using multiple agents, each of which plans its path using information from the entire known map and knowledge of each of the other agents. Each of the works discussed above addresses at least one of these components, but none provides a solution for dispersing multiple agents through a partially known environment in an intelligent manner that allows them to seek out a target at a known position or to uncover information beneficial to that objective.

The approach laid out in this paper uses elements of path planning to calculate paths for each agent that are most productive in seeking out the target. It uses map building techniques to allow agents to acquire, store, and share knowledge about the environment so that it can be used in selecting the best paths. There is also logic in place to prevent agents from duplicating exploration efforts, as far as possible, while giving priority to areas of the map most likely to lead to the target. Unlike the maze-solving solutions discussed above, the approach presented in this paper does not rely on graph

theory. This means that, as in real-world applications, information about the environment and status of the agents can be updated while any of the agents is between waypoints. It also means that the dynamics of each agent can be accounted for and that each agent can have unique parameters.

The work that most closely resembles the problem addressed in this paper is that of Wan et. al. [59]. This work assumes a single agent in an environment populated by obstacles of arbitrary shape and size placed at unknown locations within the environment. The work goes on to develop a method by which a mobile agent can identify and locate these obstacles. As the obstacles are revealed to the agent the location and size information is used to calculate node locations such that the nodes and edges are as far as possible from the obstacles. As possible nodes are computed an A* approach is used to choose an unvisited node that has the best chance of leading towards a predefined target and not encountering obstacles. The cost function considers: 1) distance between current node and next potential node, 2) estimated Euclidean distance between the current node and the target, and 3) distances between the current node and the known obstacles along the prospective path. This last consideration imposes a penalty on prospective intermediate targets that have more obstacles associated. The lowest cost node is chosen as an intermediate goal and this process is repeated until the global goal is reached.

Contributions. This paper presents a methodology similar to that of Wan et. al. [59], but addresses the use of multiple agents in the search of an unknown environment for a radiating target. Given that there are multiple agents, this thesis shows how different levels of communication between the agents effects the time required to reach the target. Expanding on Wan et. al., this method also examines the effects of different cost

functions, in addition to an A*-based cost function, for determining the most profitable path for an agent to take. Unlike many of the cell-based approaches to exploration and maze-solving, it uses all of an agent's knowledge about the environment to select the agents' paths in their search for the target. Finally, this thesis also examines the effect of the environment size on solve time, as it is related to the other factors such as the number of agents and communication range.

The following sections describe the assumptions and methods used for the single and multi-agent cases and the simulation environment used to validate the approach. These are followed by the results of simulations run using several different environment set-ups and numbers of agents. These are compared using several efficiency metrics and are also measured against single-agent and ideal-path baselines.

Definitions of Terms

Intermediate Target Node: graphical representation of a waypoint chosen during the search to aid an agent in reaching the radiating source

Radiating Source: a feature of interest that is reporting its location either through a communications broadcast or emitting a detectable energy or substance

Target Node: graphical representation of the radiating source

\mathbf{A} : the set of agents searching the environment

A_i : the i^{th} agent in \mathbf{A}

$D(\tau_i, T)$: Euclidean distance between τ_i and T divided by the average speed of A_i

e_θ : error between the agent's current heading and the desired heading

\mathbf{E}_τ : set of edges connecting an intermediate node τ to any node not contained in \mathbf{P}

G : a connectivity graph, containing a subset of nodes from \mathbf{P}

i, j : subscripts denoting a graph, node or set specific to agent A_i or A_j

j : the current timestep

k_i : integral constant

k_p : proportional constant

$L()$: edge distance between consecutive nodes in \mathbf{Q}

N_c : list of connected nodes

\mathbf{P} : set of nodes that have been previously discovered by the agents in \mathbf{A}

\mathbf{Q} : a set of nodes representing path between τ_i and $\mathbf{U}_i\{j\}$, as calculated previously using Dijkstra's algorithm

\mathbf{R}_i : set of nodes connected to τ_i , and not contained in \mathbf{P}

s : average speed of A_i

$s(x)$: field strength at the agent's current location

$s(1)$: maximum possible field strength

T : the (estimated location of) target node

t_s : magnitude of the timesteps in seconds

\mathbf{U} : subset of \mathbf{P} representing all known, but unvisited nodes

$\mathbf{U}_i\{j\}$: j^{th} node in \mathbf{U}_i

\mathbf{V} : subset of \mathbf{P} representing all visited nodes

Δv : increment applied to left and right wheel speeds

v_{max} : maximum allowable wheel velocity

v_l : left wheel velocity

v_r : right wheel velocity

τ : Intermediate target nodes; The set of nodes from U chosen as agent destinations

θ_{cmd} : desired heading, without accounting for obstacles/walls; direction of the next node with respect to the agent's current position

θ_{field} : direction of the wall potential field at the agent's current position

θ_{new} : new desired heading

Chapter II

Methodology

This paper presents a solution to the problem of searching an unknown environment for a radiating source using multiple unmanned agents. The approach involves continuously updating the environment map as new information is acquired. As more of the map is uncovered, agent paths are re-planned in an effort to improve each individual's chance of finding the target and to cover as much of the map as possible during exploration.

One of the reasons that many path planning approaches cannot be used directly to address the problem of searching in an unknown environment is that they require complete knowledge of the environment in order to determine a best path. However, node-based path-planning methods can be used with an incomplete knowledge of the environment if an agent instead navigates to an intermediate target chosen within the scope of the known map.

When exploring unknown environments it is common to assign the agents goals, areas, or waypoints to navigate towards, in order to reveal more information about the environment, as in [1], [3]. Here, the ultimate goal is not to cover the entire environment or gather as much information as possible, but rather to find a path to a given target in a minimum amount of time. This means that although any additional knowledge of the environment is likely to be beneficial, it is logical to make the exploration pattern biased towards areas that are more likely to lead to the target in a timely manner. This, also, can be accomplished though appropriate selection of intermediate target nodes.

In the approach to searching for a radiating source in an unknown environment presented in this paper, each agent chooses an intermediate target node based on a minimum cost criteria that aims to balance progress toward the target with adequate exploration of the environment and then uses a node-based path-planning method to navigate to it, updating its map of the environment as new information becomes available. The following sections present a method that allows multiple agents to balance environment exploration with efficient target location.

Search Algorithm Assumptions

This section explains the assumptions used in formulation of the search algorithm. These include the assumptions related to environment representation, agent node selection, and agent path planning.

Agents. The exact configuration of the agents should be immaterial to the generalized search algorithm. The agents may be identical to one another or may be a heterogeneous group with variations in speed, size, or mobility characteristics. The only requirements for the agents are as follows:

- 1) Each agent must be able to move.
- 2) Each agent must be able to determine its position and orientation. This may be done through the use of some global/local positioning system or with an inertial sensor system relative to the starting point.
- 3) Each agent must use reactive obstacle avoidance as there may be new obstacles (such as other agents) on the planned path to intermediate nodes.
- 4) Each agent must be able to identify new nodes and connectivity of these nodes in real-time.

- 5) Each agent must have knowledge of the target location. This may be determined using an on-board sensor, relayed from an outside source, or be prior knowledge.

Communication. It is assumed that each agent can communicate with every other agent within a set distance. The exact quantity of this distance is not a factor in the generalized version of the search algorithm. It is further assumed that distance is the only factor affecting communication between agents. Communication is assumed not to require line-of-sight between agents, though this does change the communication distance that can be achieved by a given system.

Map and Environment. The exact configuration of the environment has no direct bearing on the generalized search algorithm, but it is assumed that the explored environment can be represented by a connectivity graph. It is generally expected that the nodes of the connectivity graph represent intersections in available travel directions or dead ends, as shown in the simulations of Chapter III. However, this is not a requirement and the algorithm would still function if nodes were placed in locations with no significant environmental features. This last circumstance may occur when no intersection or dead ends can be detected within the range or field of view of the agent sensor. It is further assumed that the agents have no prior knowledge of the environment and that the environment does not change except for small obstacles (such as other agents) that do not obstruct the established paths.

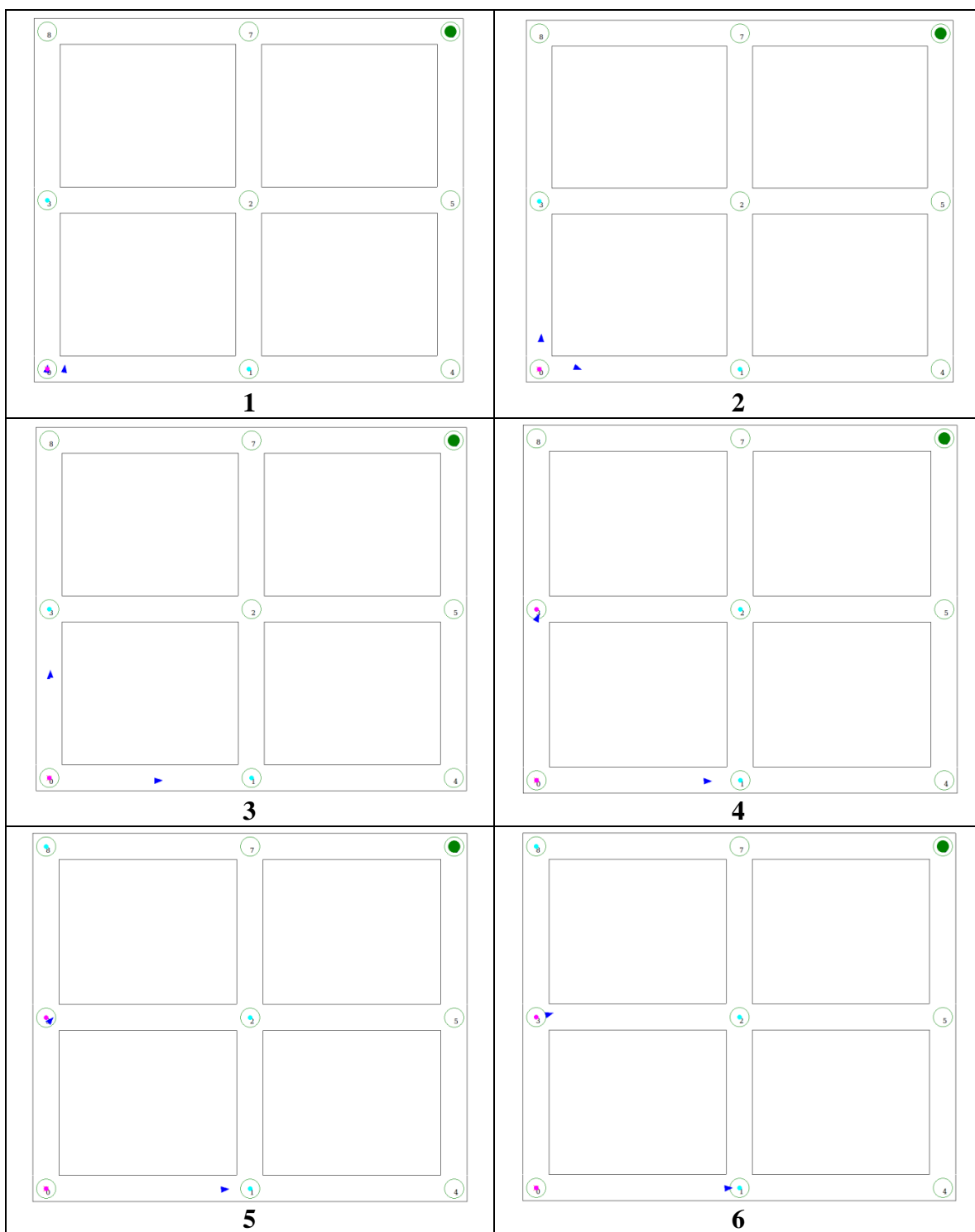
Search Overview

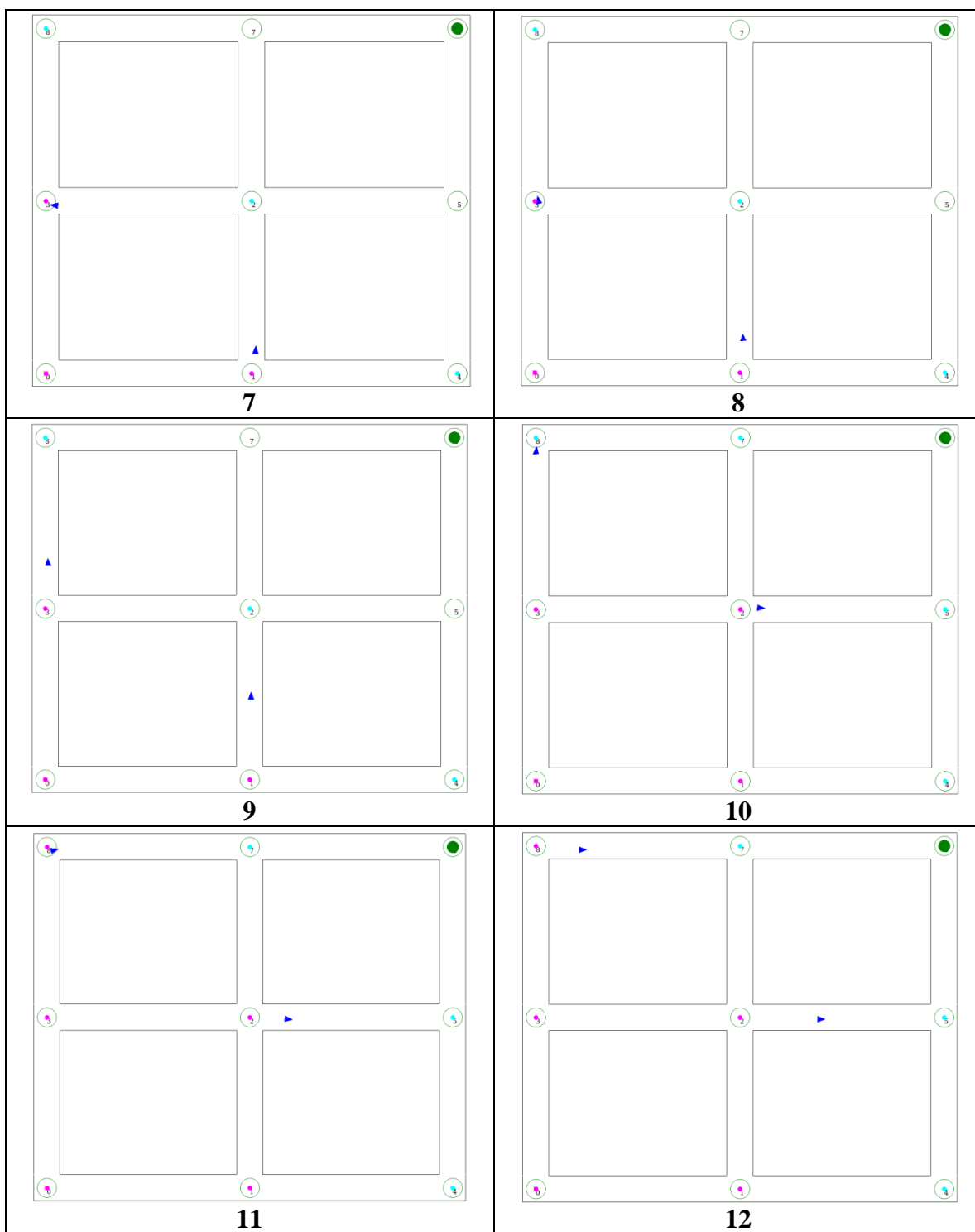
This section is intended to give a brief overview of the function of the search algorithm and the environment in which simulations for the results shown in Chapter IV

are run. Details relating to the method by which the agents navigate through the environment are given beginning on page 21 and a discussion of how simulations are run is presented in Chapter III.

All agents start at the same position within the environment. They move through this environment with the ultimate goal of reaching the target beacon and with the short-term directive to move towards locations that are most likely to lead to that target beacon. The positions of the agents are updated at each timestep during the simulation. As agents reach nodes represented in the connectivity graph these nodes are marked as “known” in the agents’ environment models. Any nodes connected to a node at which an agent is located is assumed to be seen by that agent and is added to the agent’s environment map and marked as “revealed”. Whenever agents are within the range for communication, they will share any new information with one another.

In the simulation maps, known and revealed nodes are shown with colored dots. All unmarked nodes represent regions of the environment as yet unknown to the agents. Since it is possible to have more than one unique set of known and visited nodes due to communication limits, the known and visited nodes shown to the user in the simulation maps represent those nodes that are known to any agent. Known nodes are indicated by a blue dot node marker, and visited nodes are indicated by a red dot. The target location is shown by a green dot. A sequence of a representative simulation is shown below.





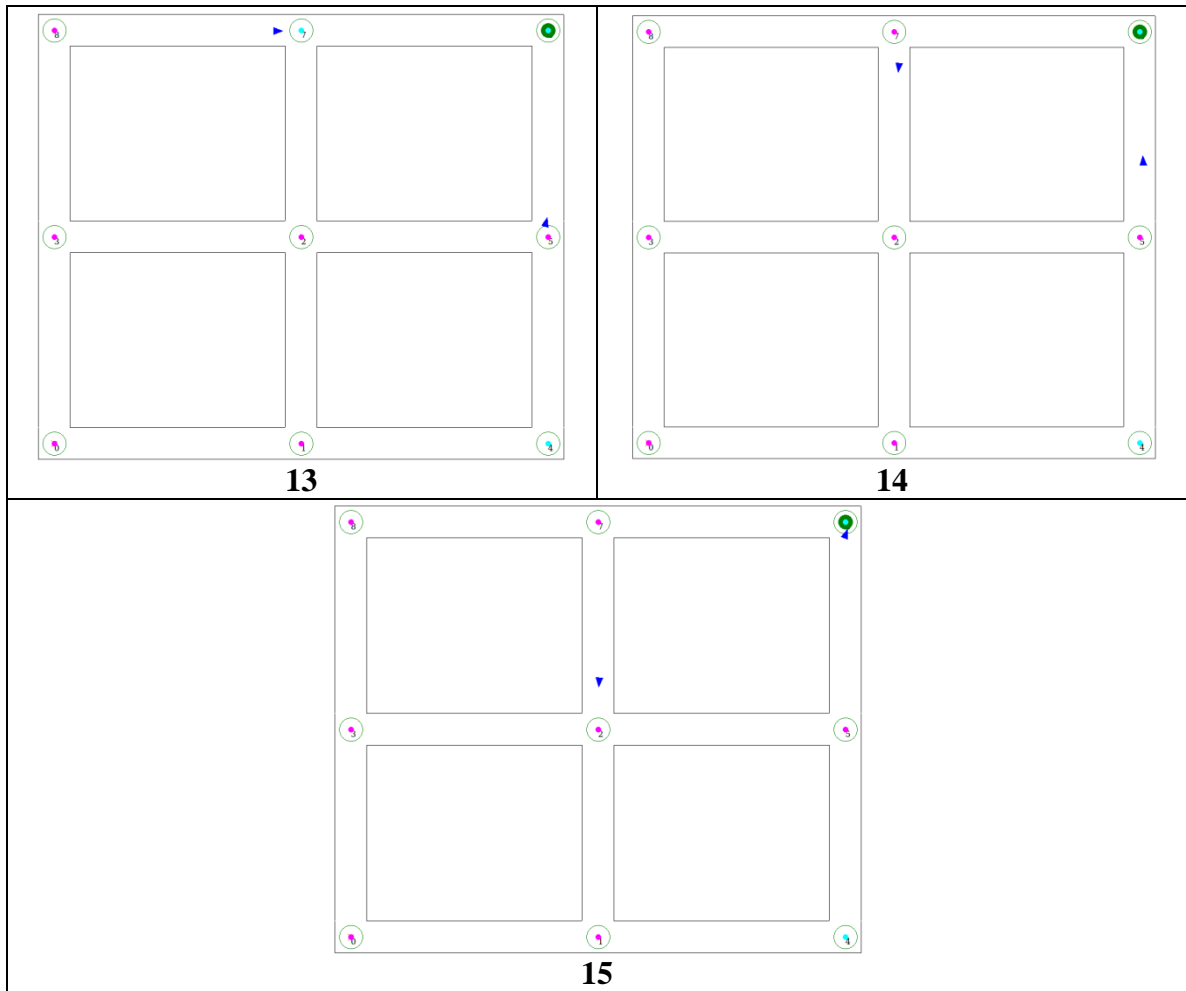


Figure 1: Simulation Sequence

Step 1: Both agents are at the start node. The start node has been visited (as shown by the pink marker) and the two adjacent nodes (nodes 1 and 3) are known, but not visited (as shown by the blue marker).

Steps 2-3: One agent heads toward each known, but unvisited, node (nodes 1 and 3, shown in blue).

Step 4-5: One agent arrives at node 3. Node 3 has now been visited and is shown with a pink marker. Nodes 2 and 8 are now discovered and are shown with blue markers.

Step 6: The agent at node 3 begins to turn toward node 2. The other agent arrives at node 1.

Step 7: Since the agent at node 3 is still within that node's region of influence, it is eligible to have its intermediate target node, τ , reassigned. The agent at node 1 is determined to have a lower cost with respect to node 2, so that agent begins to head toward node 2 while the agent at node 3 is redirected to node 8. Node 1 is now visited (pink) and node 4 has been identified (blue).

Steps 8-9: The agents travel from node 3 to node 8 and from node 1 to node 2.

Step 10: One agent arrives at node 2. This node becomes known (pink) and the two previously unknown adjacent nodes (nodes 5 and 7) are now known (blue). The agent heads toward node 5.

Step 11: One agent arrives at node 8. This node is now visited and no new nodes are added to the map since all nodes adjacent to node 8 are already known.

Step 12: The agents head toward node 5 and node 7.

Step 13: One agent arrives at node 5. This node is now visited and node 6 (the target node) is revealed. The agent at node 5 heads toward node 6.

Step 14: One agent arrives at node 7. Since the other agent is already headed toward node 6, the agent at node 7 begins to travel toward the only remaining unvisited node (node 4) by heading toward node 2.

Step 15: One agent arrives at the target node (node 6), ending the search.

Terminology

A : the set of agents searching the environment

A_i : the i^{th} agent in A

i, j : subscripts denoting a graph, node or set specific to agent A_i or A_j

T : the target node

P : set of nodes that have been previously discovered by the agents in A

U : subset of P representing all known, but unvisited nodes

V : subset of P representing all visited nodes

τ : Intermediate target nodes; The set of nodes from U chosen as agent destinations

G : a connectivity graph, containing a subset of nodes from P

R_i : set of nodes connected to τ_i , and not contained in P

E_τ : set of edges connecting an intermediate node τ to any node not contained in P

Search Algorithm

The search algorithm presented in this thesis is designed to be executed continuously by each agent until it either acquires the target or receives confirmation that another agent has done so. Each agent begins with at least three pieces of information: its own initial pose, estimated location of the target T , and the locations of agents A . The environment is initially unknown, making G empty for all agents. As the search progresses A_i will acquire knowledge about the environment (e.g. intersections and dead ends) necessary to add new nodes to its connectivity graph, G_i , and will share this information, plus its current position and intermediate target node, τ_i , with the other agents. A high-level overview of the search algorithm is shown in Figure 2. The key processes of this search are numbered 1 through 5 (shown in red) and are detailed below.

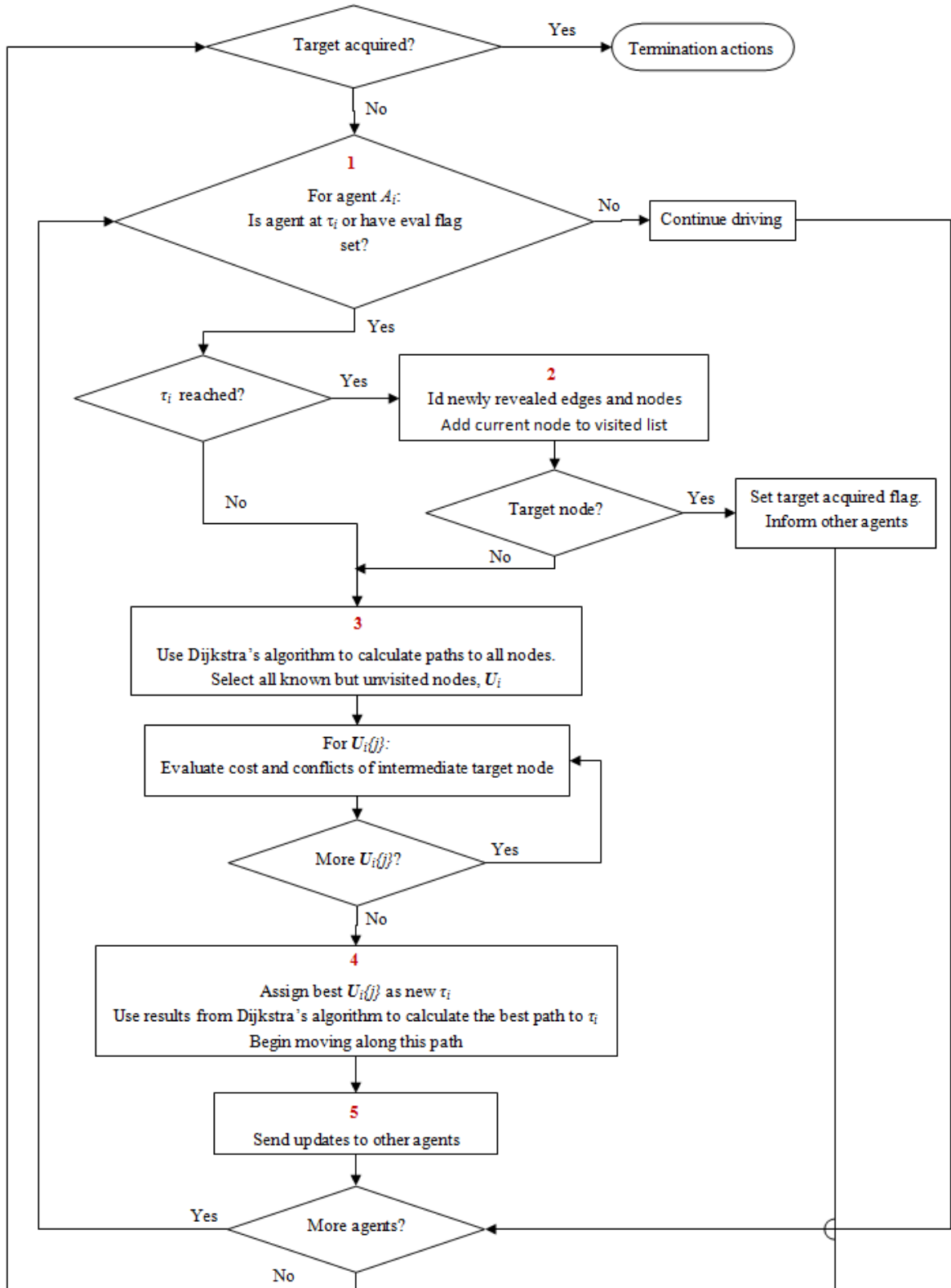


Figure 2: Overview of a Search in an Unknown Environment using a Continuously Updating Connectivity Graph
(red numbers relate to paragraphs below)

Check for Re-Evaluation of Intermediate Target Node. Once it has been determined that no agent has reached the target T , each agent A_i , must decide to keep its intended destination, defined by the intermediate target node τ_i or select a new τ_i . A_i will select a new τ_i if the previous τ_i has been reached or if a flag is set by one of two conditions. These conditions are 1) the agent receives map updates from other agents and 2) an agent A_j determines it can get to τ_i faster than A_i , causing A_j to select $\tau_j = \tau_i$.

If A_i has not reached τ_i and has not been instructed to re-evaluate τ_i , it will continue driving along its current path to τ_i .

Continuously Updating Connectivity Graph. Before the search starts, all agents have knowledge only of the starting node, which is added to \mathbf{P} and \mathbf{V} , and which is also all agents' first τ_i . When agent A_i is at an intermediate target node, τ_i , it evaluates the environment for nodes connected to τ_i . All of the newly revealed nodes, \mathbf{R}_i , along with the edges, \mathbf{E}_τ , connecting them to τ_i , are added to the known connectivity graph, G_i , stored by A_i .

Path Calculation and Identification of Unvisited Nodes. When agent A_i has reached any τ_i it must select its new τ_i from \mathbf{U}_i , a set of unvisited nodes, where

$$\mathbf{U}_i = \mathbf{P}_i - \mathbf{V}_i \quad (1)$$

This is done in part by calculating a cost for each node in \mathbf{U}_i , as discussed below, beginning on page 27. Since the path to any node in \mathbf{U}_i may be a factor in its cost and since the shortest path will need to be computed for the new τ_i before A_i can begin to move toward it, it is advantageous to calculate the shortest paths to all nodes in \mathbf{U}_i at this stage. Dijkstra's algorithm [61] is used for this calculation because will find the shortest

paths to all nodes with a lower computational cost than would be needed to calculate all the paths individually. Since paths to all nodes in U_i must be calculated for several of the possible cost function (see page 27), the cost of using Dijkstra's method is likely to be minimal compared to the cost of calculating paths on an as-needed basis. The exception, of course, would be the case of a large known map where all the unvisited nodes are concentrated in a relatively small section.

Agent Node Selection. In this process intermediate target nodes τ are selected. The choice of selection criteria can be case dependent, and several are described beginning on page 27, but in general τ should be selected to satisfy the following guidelines:

- 1) An intermediate target node must not have been previously visited.
- 2) An intermediate target node should reduce the time required to reach the target.
- 3) Multiple agents should have the same intermediate target node only if there are more agents than the number of unvisited nodes in U .
- 4) In the case where there are more agents than the number of unvisited nodes in U , agents should distribute themselves as evenly as possible between the nodes.

Node Selection Sequence. There are four basic steps an agent follows in selecting its next intermediate target node. They are:

- 1) Identify unvisited nodes (Figure 2, Block 2).
- 2) Evaluate costs of unvisited nodes (Figure 3, Blocks 1 and 4).

- 3) Check lowest cost node against other agents' τ and resolve conflicts to minimize duplicate τ selections (Figure 3, Blocks 2 and 5).
- 4) Assign most suitable node, based on steps 2) and 3) (Figure 3, Block 6).

As there is usually more than one unvisited node, steps 2) and 3) are repeated for each node in U_i until all the costs for all nodes have been compared and the most suitable node identified. Figure 3, below, shows a flow chart of the node selection process. The primary steps are numbered in red and are discussed in detail below, beginning on page below. Steps 2) and 3) are discussed in detail below, beginning on page 30.

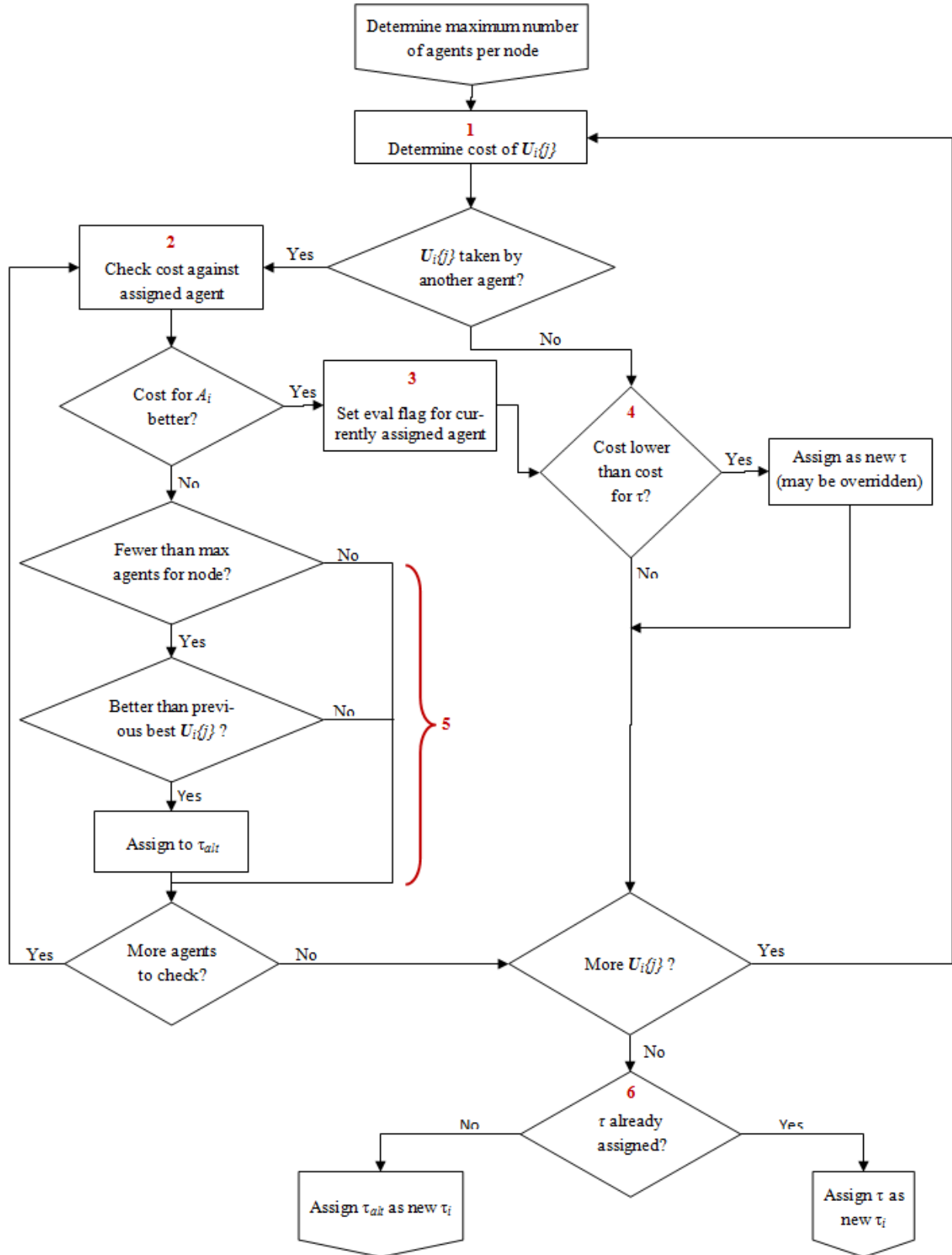


Figure 3: Node Selection Flowchart
(red numbers related to paragraphs below)

Best Node Criteria. Potentially, any cost function (Figure 3, Block 1) could be used to select τ_i . However, given the goal of locating a target of known position within a minimum amount of time, it seems reasonable that the criteria should consider some combination of a cost to get to the node and a cost to get from the node to the target. To this end, three node cost criteria are tested. The A* cost function uses both time to the node and estimated time from the node to the target, the Dijkstra cost function considers only time to the node, and the greedy cost function considers only the estimated time from the node to the target. Each method is described in detail below.

A* Cost Function. A* path planning [60] approaches have two parts: an edge cost and a heuristic. The cost function, C , used here operates on a similar principle and is given by:

$$C(A_i, \mathbf{U}_i\{j\}) = E(\mathbf{Q}) + H(\mathbf{U}_i\{j\}, T, \mathbf{Q}) \quad (2)$$

Where:

$\mathbf{U}_i\{j\}$: j^{th} node in \mathbf{U}_i

T : the estimated location of the target node

\mathbf{Q} : a set of nodes representing path between τ_i and $\mathbf{U}_i\{j\}$, as calculated previously using Dijkstra's algorithm

The edge cost, $E(\mathbf{Q})$, is simply the time it would take A_i to travel the distance of the path represented by \mathbf{Q} and is given by:

$$E(\mathbf{Q}) = \frac{\sum L(\mathbf{Q}\{j\}, \mathbf{Q}\{j+1\})}{s} \quad (3)$$

Where:

$L()$: edge distance between consecutive nodes in \mathbf{Q}

s : average speed of A_i

The heuristic, $H(\mathbf{U}_i\{j\}, T, \mathbf{Q})$ is composed of two parts: $D(\tau_i, T)$, the estimated time it will take A_i to travel from $\mathbf{U}_i\{j\}$ to T , and $F(\mathbf{Q})$, the estimated time it will take A_i to execute each turn in \mathbf{Q} .

$$H(\mathbf{U}_i\{j\}, T, \mathbf{Q}) = D(\tau_i, T) + F(\mathbf{Q}) \quad (4)$$

Where:

$D(\tau_i, T)$: Euclidean distance between τ_i and T divided by the average speed of A_i

The estimated time to execute a turn is based on the angle of the turn and on the width of the opening. In making this estimate the agent is assumed to track through the middle of turn so that the turn may be represented by an arc of a circle with a diameter equal to the width of the opening. The determination of the assumed path is shown in Figure 4.

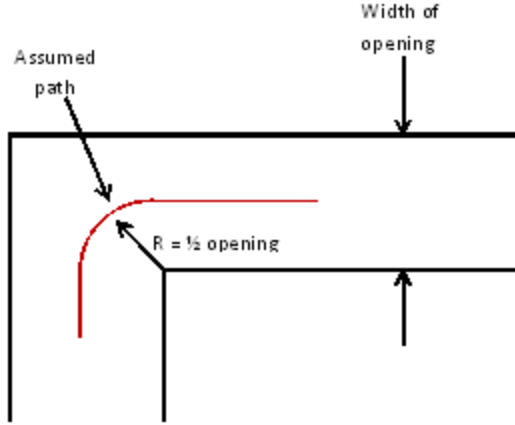


Figure 4: Assumed Path for Agents when Turning

As can be seen from the figure, the estimated distance the agent must travel to execute the turn can be determined from the arc length, which is in turn a function of the radius of the circle and the angle through which the agent must travel. $F(\mathbf{Q})$ can be calculated by summing the time it will take A_i to cover the distance incurred by each of the n turns in \mathbf{Q} , as shown below.

$$F(\mathbf{Q}) = \frac{\sum_n (\theta r)_n}{s} \quad (5)$$

Dijkstra Cost Function. Dijkstra's algorithm, first presented by E. W. Dijkstra in [61], is a special case of A* where the heuristic is zero. As such, this Dijkstra-based cost function will result in selecting the nearest unvisited node as τ_i . Since the agents are prevented, where possible, from choosing the same nodes and are not biased toward the target or any other point, this cost function is expected to result in a broader search than the A* cost function. The Dijkstra cost function is given by:

$$C(A_i, \mathbf{U}_i\{j\}) = \frac{\sum L(\mathbf{Q}\{j\}, \mathbf{Q}\{j+1\})}{s} \quad (6)$$

Greedy. The greedy cost function uses only the straight-line distance between τ_i and T divided by the average speed of A_i . The current position of A_i therefore has no influence on node selection when this criterion is used. The Greedy cost function is given by:

$$C(A_i, \mathbf{U}_i\{j\}) = D(Nt_i, T) \quad (7)$$

The agents are still prevented from selecting the same intermediate nodes as before, but since they are drawn to the known nodes closest to the target a search using this node selection criterion will tend to more closely resemble a depth-first search.

Node Selection for Multiple Agents. In order to minimize redundancy in the agents' search efforts and to encourage dispersion of the agents through the environment additional criteria are required in the multi-agent case to resolve conflicts that arise when two agents choose the same τ (Figure 3, Block 2). When A_i evaluates the suitability of each node in \mathbf{U}_i it first considers the cost of the node and then checks to see if any other agent has $\tau = \mathbf{U}_i\{j\}$. If this is the case, A_i compares its cost for $\mathbf{U}_i\{j\}$ with the cost for the other agent, assigning $\tau_i = \mathbf{U}_i\{j\}$ if the cost for A_i is lower and instructing the other agent to set its evaluation flag (Figure 3, Block 3), or selecting the next best $\mathbf{U}_i\{j\}$ if the cost for A_i is higher. When the number of agents executing the search is less than or equal to the number of unvisited nodes, this process is straightforward, with any given node assigned to a single agent. There are instances, however, when there are more agents than

unvisited nodes and it will be necessary for several agents to choose the same τ (Figure 3, Blocks 5). In this case a maximum number of agents allowed for any one node is defined as:

$$maxAgents = \left\lceil \frac{nAgents}{nNodes} \right\rceil \quad (8)$$

A_i will now assign τ_i to be the lowest cost $\mathbf{U}_i\{j\}$ (Figure 3, Blocks 4 and 6) for which the number of other agents with $\tau = \mathbf{U}_i\{j\}$ and a lower cost is less than the maximum allowed number of agents.

Update other Agents. Once an agent A_i has selected a new τ_i , A_i must communicate τ_i and its newly updated G_i to all agents within communication range (Figure 3, Block 5). Communicating the entirety of G_i instead of just any newly revealed information ensures that any agents previously out of communication range receive the most complete information. This map update will in turn cause each of these agents to re-evaluate its τ based on the new information. This entire process of node discovery and intermediate target selection is repeated for each agent at each timestep until the target has been found.

Chapter III

Simulation Environment

In order to test the efficacy of the proposed search algorithm a custom simulation environment has been developed. The simulation environment is written in Python and includes functionality needed to create maps and edit simulation parameters such as the number of agents, start and target nodes, and the cost function used to determine the agents' intermediate nodes.

Justification

While several robotic simulation environments exist for both specific applications and for general development purposes, many cannot be applied to the proposed scenario or would require extensive low-level set-up. Simulation environments such as Webots [62], Microsoft Robotics Developer Studio [63], and anyCode Marilou [64] are intended to model robotic sub-systems and components, rather than operating environments and may be costly. Similarly, other simulators target specific aspects of simulation, such as realistic physics modeling (EZ Physics [65]), controller implementation (Lpz Robots [66]), or environment modeling (ANVEL [67]). ANVEL, in particular, is also quite graphics-intensive, which is not desirable when attempting to generate results from a large number of simulations. While many of these have qualities that could be useful here, their specificity would make integration with other facets of the simulation less practical.

Another drawback of many of the simulators mentioned above is that they require detailed low-level set-up of the environment and robots, down to the level of environment surface or motor characteristics. This level of detail would add unnecessary complexity to

the simulation and would make it difficult to build many and varied scenarios in which to test the proposed algorithm.

Morse [68] and STDR [69] are the two simulation frameworks closest to what was needed to test the proposed algorithm. However, neither of these is user friendly and both have a steep learning curve with a multitude of unnecessary features.

Overall, given that the goal of the simulator is to test a high-level algorithm, it was deemed preferable to build a specialized simulation environment with representative high-level physics and controllers and having accessible control for those simulation variables necessary to provide a thorough proof of concept for the proposed search algorithm.

Simulation Environment

Assumptions. This section provides an overview of the assumptions taken in constructing the simulation environment. These assumptions have no bearing on the search algorithm itself and most are made either for the sake of simplicity and limitation of unnecessary processing, or because a more realistic model would be highly application-dependent.

- 1) The environment is maze-like. A maze-like map lends itself well to a nodal representation and is also a reasonable approximation of an arbitrary building floor plan – a reasonable environment for this type of search.
- 2) The corridors are assumed to be perpendicular and of an equal width. This minimizes the computation needed when generating the graphical representation of the environment, extracting map information from graphical input, and calculating intersections.

- 3) The agents are modeled as differential drive vehicles. They are assumed to be 4 inches in diameter.
- 4) The agents are assumed to have nominal speeds of 32 in/sec. Other speeds can be accommodated, but no scenarios were run with speeds other than 32 in/sec. Motor and wheel physics are not modeled
- 5) The environment is scalable to any size. However, for the purposes of this research the smallest simulated environment size is approximately 12ft x 20ft and the largest is approximately 37ft x 61ft.
- 6) The agents are assumed to have sensors and processes to establish pose, identify walls and other obstacles, and the target location. The sensors and data processing are not modeled.
- 7) The agents are assumed to have communications equipment that allows sharing of information within a specified distance. This distance may be set before running any simulation. The communications hardware is not modeled.
- 8) The agents are assumed to have 360° visibility with regard to walls/obstacles and the target.
- 9) The agents are assumed to have software processes necessary to determine when to add a new node to the graph. These processes are not modeled.
- 10) The agents are assumed to have adequate on-board processors and memory. No limits are placed on these, aside from those imposed on the entire simulation by the system on which it is run.

Simulation Options. Two versions of the simulation code are used. The first is graphical and is primarily for demonstration purposes. A command-line version of the

simulation code contains the same calculations, has no graphics, and is configured for batch process of simulations to quickly produce many simulation results. The following sections discuss the agent and environment models, controllers, and the calculations necessary to demonstrate the movement of the agent through the environment. These are common to both simulation options. An overview specific to the graphical option can be found in 0.

Simulation Structure. In practice, it is common for each individual in any multiple-agent robotic team to run an independent instance of the same or similar code and to carry out all calculations for itself using onboard resources. This results in a parallel system, where all sharing of information is done on a broadcast or request basis. One of the comparisons made in evaluating the proposed search algorithm is the performance differences between agent groups of various sizes. In order to do this, the simulation environment has to allow the agent population to be extended to a reasonably large size. If the simulation processed the agents in parallel the total number of agents would be limited by the number of processing cores available on the system. On a majority of personal computers this would be relatively few (2-4). Considering this limitation, the simulation environment is written to process each agent in turn on each timestep.

Agent Model. Agents are modeled as differential drive vehicles with a wheel base of 4 inches and a wheel radius of 0.5 inches. The agents are assumed to have circular bodies 4 inches in diameter. The agents' maximum speeds are 32 in/sec.

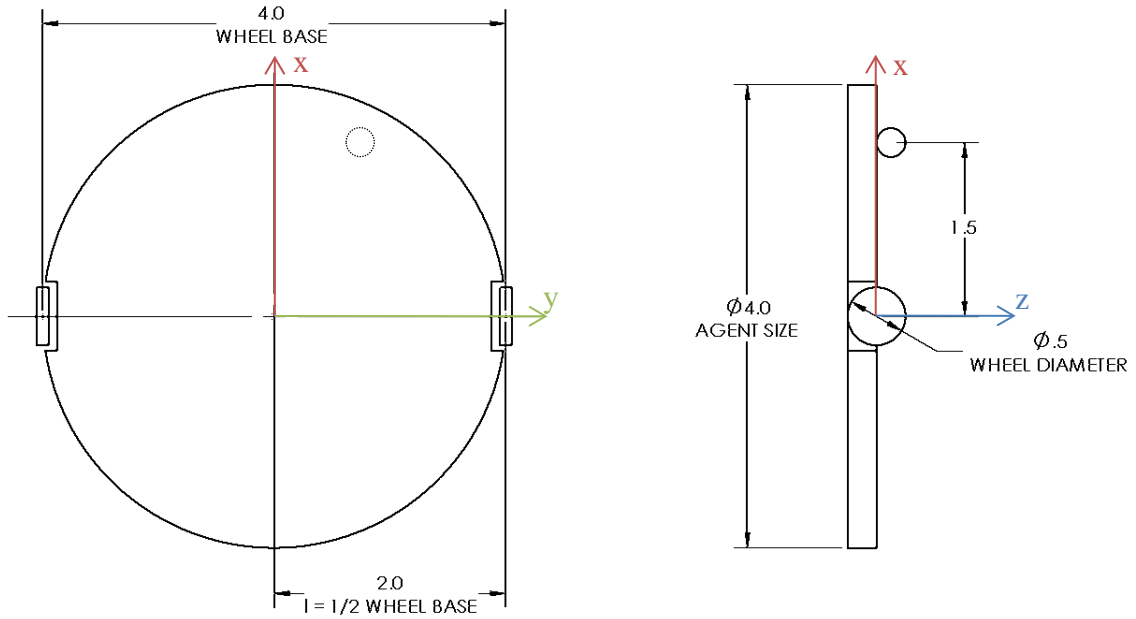


Figure 5: Agent Configuration
(all measurements in inches)

A ground vehicle operating normally in a 2D environment has three degrees of freedom – it can translate in the x- and y-directions, and rotate around the z-axis. The vehicle has a reference frame that is fixed to the vehicle with its y-axis through the drive wheels and its x-axis through the free wheel, as shown in Figure 5. The vehicle's pose is calculated as the position and orientation of this reference frame and is represented by:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \quad (9)$$

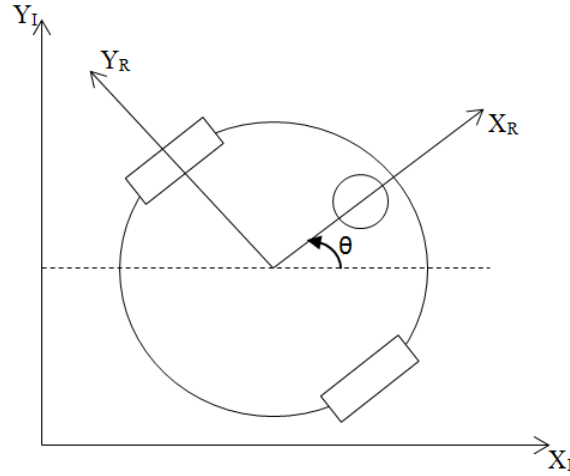


Figure 6: Agent Orientation

The transformation matrix, $R(\theta)$, relates the orientation of the robot reference frame to the inertial reference frame.

$$R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (10)$$

The vehicle's pose in its own reference frame is related to its pose in the inertial reference frame by:

$$[\xi_R] = \{R(\theta)\}[\xi_I] \quad (11)$$

A differentially driven vehicle has two drive wheels and one free wheel. Since the free wheel is unpowered and allowed to move in any direction, only the drive wheels need be considered when determining how the vehicle will move. Assuming that the drive wheels remain vertical and do not slide, each wheel has a forward velocity given by

$$v = r\dot{\psi} \quad (12)$$

where r is the wheel radius and $\dot{\psi}$ is the angular velocity of the wheel. By applying standard rolling and sliding constraints and accounting for the orientations of the wheels, the following equation is derived to describe the motion of a differentially driven vehicle in a two-dimensional environment.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & -l \\ 1 & 0 & l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} r_L \dot{\psi}_L \\ r_R \dot{\psi}_R \\ 0 \end{bmatrix} \quad (13)$$

Where l is half the wheel base, as shown in Figure 5.

Further details on the mathematical model used to calculate the agents' movements are shown in 0.

Agent Starting Pose. When multiple agents are used in the simulation they must be positioned around the start node so as not to be in collision with one another or with any walls. This is done by aligning their starting positions vertically or horizontally around the start node. Where possible, the agents will be equally arranged horizontally or vertically to either side of the start node (Figure 7(e-g)). When this is not possible the agents are arranged to the open side of the start node (Figure 7(a-d)). Arrangement to both sides of the start node is given precedence when possible, regardless of direction. When both horizontal and vertical arrangements are possible and equal in terms of positioning agents to one or both sides of the start node, the horizontal arrangement is the default

(Figure 7(a), (f)). All agents are oriented towards the “top” of the simulation frame, or at -90° with respect to the global coordinate system. The agents are spaces such that there is half of an agent diameter of space between any two adjacent agents.

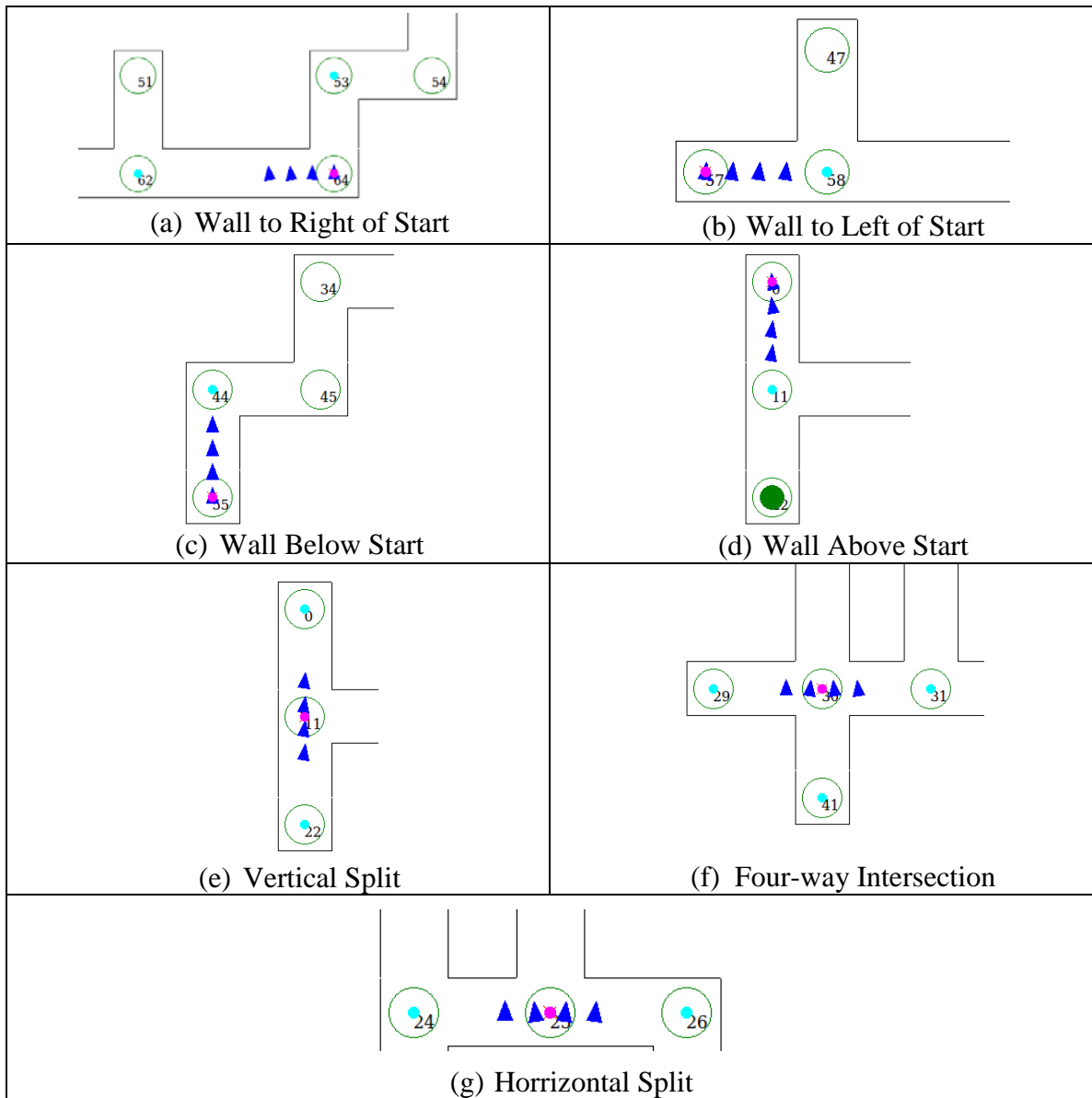


Figure 7: Multi-Agent Start Positions

Obstacle Avoidance. Two types of obstacles are considered in this simulation. The first are the walls, which are represented by a pre-calculated potential field. The second are the agents themselves. The agents are represented by a potential field of fixed strength originating from the agent's current position. The field strength is constant and is applied out to a distance equal to twice the radius of a circle completely enclosing the agent. The field generated by an agent replaces the field vectors generated by the walls in the region of the agent.

Obstacle avoidance for an agent is accomplished by incorporating the direction and strength of the potential field at the agent's current location into the calculation of its new desired heading. The stronger the field or the larger the discrepancy between the agent's current heading and the direction perpendicular to the potential field, the more influence the field will have on the new heading. For more detail on the calculation of an agent's desired heading, see the section on Controllers, beginning on 41.

The wall potential field is the means by which the agents avoid the walls. Vectors are established perpendicular to the walls at specified intervals along their length and are summed at specified intervals throughout the map space. For the wall width used in these simulations (9 simulated inches) the field strength is defined by

$$s(x) = \begin{cases} x > e, & -20 \ln x \\ x \leq e, & 0 \end{cases} \quad (14)$$

where x is the distance away from any given wall. An example of the potential field is given in Figure 8, below.

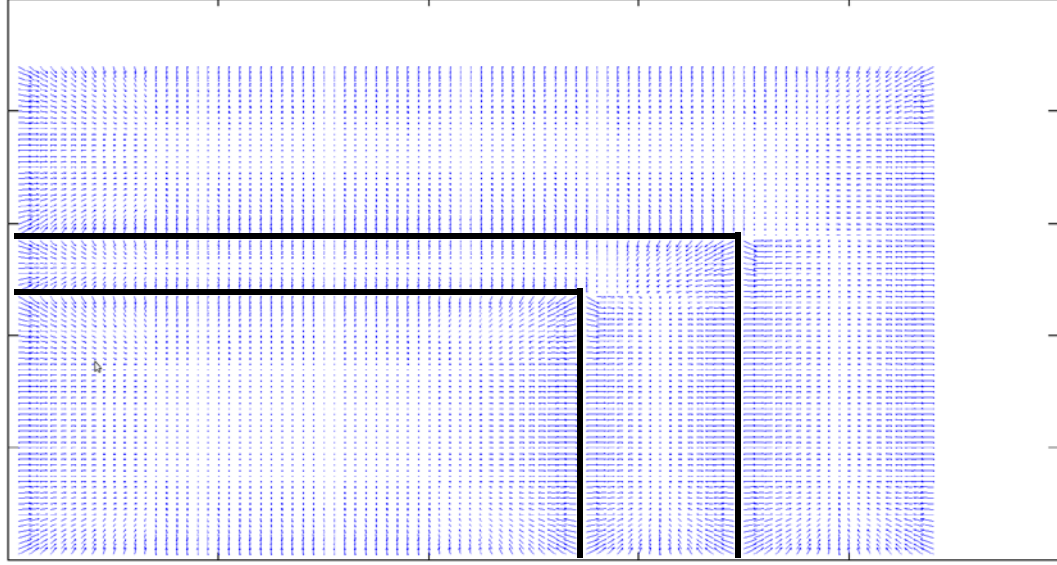


Figure 8: Potential Field Calculated for Wall Avoidance

Controllers. Agent heading control is accomplished using one of two proportional-integral (PI) controllers applied to the left and right wheel velocities. Each of these controllers corresponds to one of the two allowed agent speeds (32 in/sec). The form of the controllers is as follows:

$$\Delta v_{(j)} = k_p e_{\theta(j)} + k_i (e_{\theta(j)} - e_{\theta(j-1)}) t_s \quad (15)$$

$$\begin{aligned} v_{r(j)} &= v_{max} + \Delta v \\ v_{l(j)} &= v_r - 2\Delta v \end{aligned} \quad (16)$$

Where:

Δv : increment applied to left and right wheel speeds

e_{θ} : error between the agent's current heading and the desired heading

j : the current timestep

k_i : integral constant

k_p : proportional constant

t_s : magnitude of the timesteps in seconds

v_{max} : maximum allowable wheel velocity

v_l : left wheel velocity

v_r : right wheel velocity

The error used in the controller is based on the desired agent heading. This is determined by a combination of the direction of the agent's next node with respect to the agent's current orientation (θ_{cmd}) and the direction perpendicular to the potential field at the agent's current location with respect to the agent's current orientation (θ_{field}). The difference between these two values is weighted by the field strength so that this difference has a greater influence on the desired heading when the agent is close to a wall. The expression for the new heading is:

$$\theta_{new} = \frac{|s(x)|}{s(1)} (\theta_{field} - \theta_{cmd}) + \theta_{cmd} \quad (17)$$

Where:

$s(x)$: field strength at the agent's current location

$s(1)$: maximum possible field strength

θ_{cmd} : desired heading, without accounting for obstacles/walls; direction of the next node with respect to the agent's current position

θ_{field} : direction of the wall potential field at the agent's current position

θ_{new} : new desired heading

Termination Criteria. The termination criterion for this simulation is that any agent reaches the target node. This criterion is used instead of one requiring only line-of-sight to the target because it will cause all connections to the target node to be revealed. This allows the quickest known path to be calculated in the case that such a path does not approach the target from the same node as the agent that found the target.

Chapter IV

Results

The goal of the algorithm presented in Chapter II and Chapter III is to minimize solve time for the search problem presented. With the given algorithm there are three main factors affecting the solve time: the number of agents, the cost function used to calculate the agents' paths, and the range in which the agents can communicate. The following sections describe the setup and scenarios used to examine the effects of each of these factors, followed by the simulation results and discussion.

Map Generation

The maps used in these simulations are created using a modified depth-first maze generation algorithm. The algorithm works on a grid of a given size and begins at a given node (n). From this node the algorithm randomly selects one of the adjacent nodes to which to connect. The maze-building process continues with the new node n , and the preceding node, n_p . This process is repeated for each subsequent node, with two provisions. The first is that the new connection for n cannot be n_p , and the second is that the new connection cannot be to a node which connects to any other node. When a connection is made to a new node, that node is added to a list of connected nodes (N_c). When it is determined that a node n cannot be connected to any new nodes without violating these two provisions, the node is removed from N_c . When the algorithm reaches such a node, a node from N_c is selected as a new starting point and the process is repeated until no nodes remain in N_c .

This method for maze generation ensures that the maze uses the entirety of the given area, but has two characteristics that make it less than ideal for use in the scenario

presented in this paper. First is that the maze generator does not distinguish between directions and it is therefore possible to have straight-line segments with no intersections that are made up of multiple nodes. Second is that every node is connected to every other node by one, and only one, path. This last characteristic of a single path between any two nodes makes the maze “perfect”. Two additional steps are added to the basic depth-first maze generator to address these gaps.

First, in order to remove any unnecessary nodes that are in a straight-line, the algorithm steps through each node and checks if the x- or y-coordinate is the same as the corresponding coordinate for both the preceding and succeeding nodes along a path in the perpendicular direction, and if the node has no other connections. If both these criteria are met the node is removed.

Second, since in many real-life scenarios there is the possibility that multiple possible paths to the target exist, the maps should not be perfect mazes. In order to turn the perfect mazes generated by this method into non-perfect maps, a number of nodes are selected to be connected to adjacent nodes to which they were not previously connected. For the purpose of these simulations this number was set at 10% of the total number of nodes.

Nodes are randomly selected and connections are selected randomly from a list of the node’s possible new connections. If a node has no possible new connections another node is selected in its place. This process continues until the requisite number of new connections has been made.

The maps used for the data collection presented below are generated in five different sizes ranging from a simulated size of approximately 12ft x 20ft containing an

average of 31 nodes to a simulated size of approximately 37ft x 61ft with an average of 372 nodes. Examples of these map sizes are shown in Figure 9 and Figure 10.

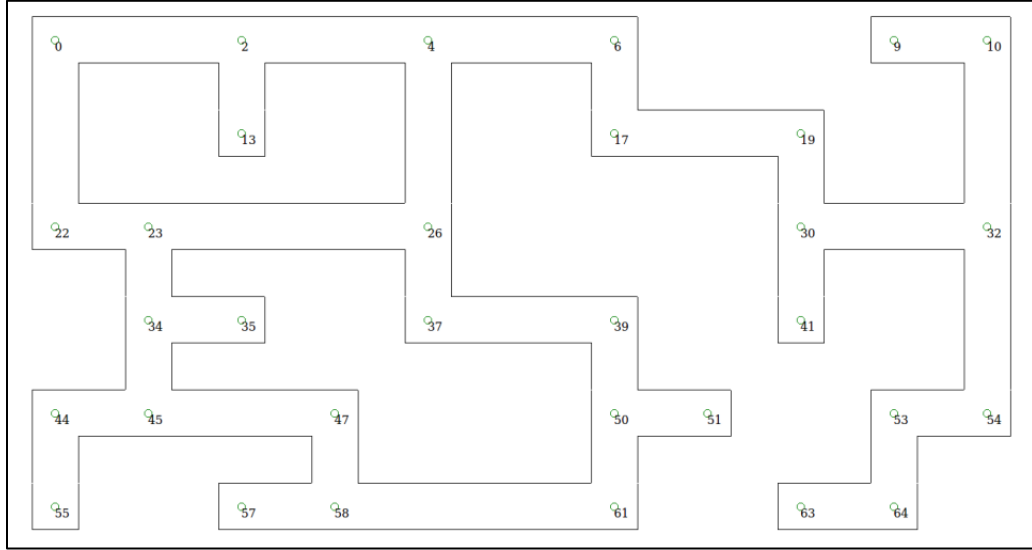


Figure 9: Example of Smallest Map Size

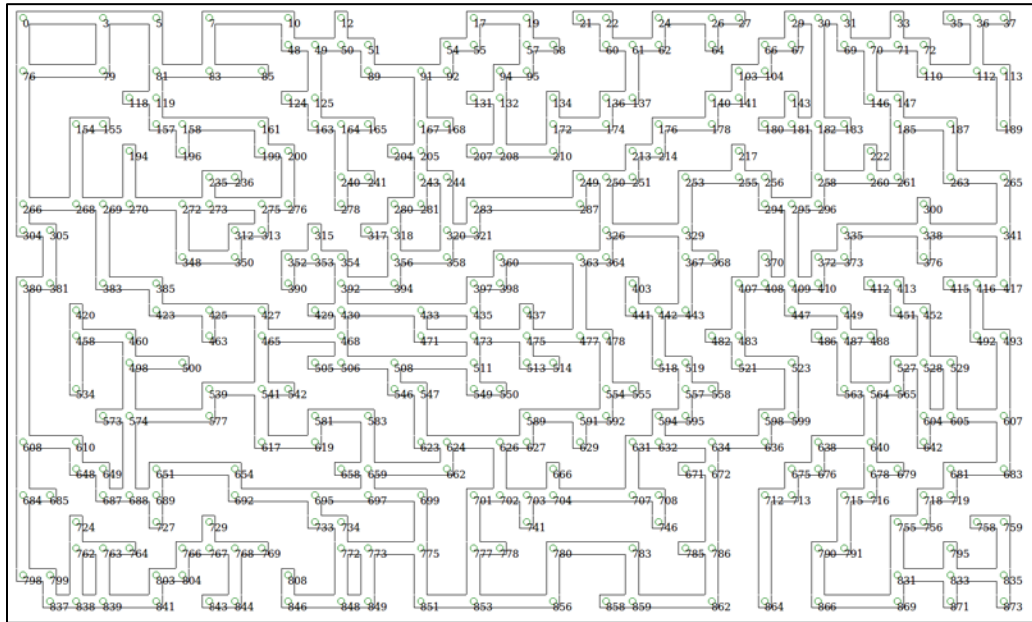


Figure 10: Example of Largest Map Size

Case Generation

The parameters for each case are assembled using an input file generation code.

The generator assembles input files from the following list of parameters:

- A supplied list of maps
- A fixed number of unique start and target combinations to be made for each map
- A minimum ideal path length between and start and target points
- A supplied list of communication distances
- A supplied list of number of agents
- A supplied list of cost functions

Each of the lists is looped such that every combination is accounted for at least once in the data set.

Data Sets and Scenario Statistics

As discussed above, both the maps used and the individual cases are semi-randomly generated. The maps can be found in 0 and details of the cases used for this analysis are described below.

Four major variables are considered. These are the map (or map size) on which a given case is run, the number of agents, the cost function used, and the communication range. The values used for each of these variables are shown in Table 1.

Table 1:

Possible Values for Primary Simulation Variables

Variable	Possible Values
Map	One of 25 unique maps; 5 maps in each of 5 sizes
Number of Agents	1, 2, 4, or 6
Cost Function	A*-, Greedy-, or Dijkstra-based
Communication Range	0, 20%, 50%, or 100% of given map size OR 0-100%, in increments of 5%, of the largest map size

As can be seen from the table, there are two different schemes for the communication range. This reflects two different data sets. For the first, the communication range was allowed to vary with map size so that communication for an individual agent would be allowed over a specified percentage of the given map's area. For the second set of cases the simulated distance assumed for communication did not vary with the size of the map used in a given case. These fixed communication distances were selected such that the entire range from zero to 100% of the largest maps' areas would be covered. A greater number of discrete distances were also used, with increments set at 5% of largest maps' areas. This results in 21 possible values for communication distance. Also, only the A* selection criteria is considered for the second data set. This second data set was designed to refine and clarify certain trends seen in cases from the first data set. The reasons for this are discussed below in the Results and Discussion.

Table 2 and Table 3 show the numbers of cases and the numbers of options for the variables for each data set. Note that for each possible combination of variables, ten unique start and target combinations are also generated, with a minimum path length of 10% of the total number of nodes in the specified map.

Table 2:

Statistics for First Data Set

Total Number of Cases: 12,000		
Paths per Variable Combination: 10		
Variable	Number of Possible Values	Cases per Value
Map	25	360
Number of Agents	4	3,000
Cost Function	3	4,000
Communication Range	4	3,000

Table 3:
Statistics for Second Data Set

Total Number of Cases: 21,000		
Paths per Variable Combination: 10		
Variable	Number of Possible Values	Cases per Value
Map	25	840
Number of Agents	4	5,250
Cost Function	1	21,000
Communication Range	21	1,000

Results and Discussion

This section looks at the effects on solve time of the three major variables (cost function, number of agents, and communication distance) both individually and as inter-dependent parameters. The efficiency of the three cost functions is discussed first. This is followed by an examination of the number of agents, both alone and as related to the other factors. The effects of communication distance are discussed toward the end of this section. All results shown in this section refer to the time required for the agents to reach the target as a percentage longer than the ideal time. In these cases the ideal time is the time that would be required for an agent to travel the shortest path between the starting point and the target, given that the entire environment (and therefore the absolute shortest path) is known and the agent travels at its maximum speed.

Figure 11 shows the effect of the cost function on the overall solve time for each map size, but does not account for possible correlations with any other parameter.

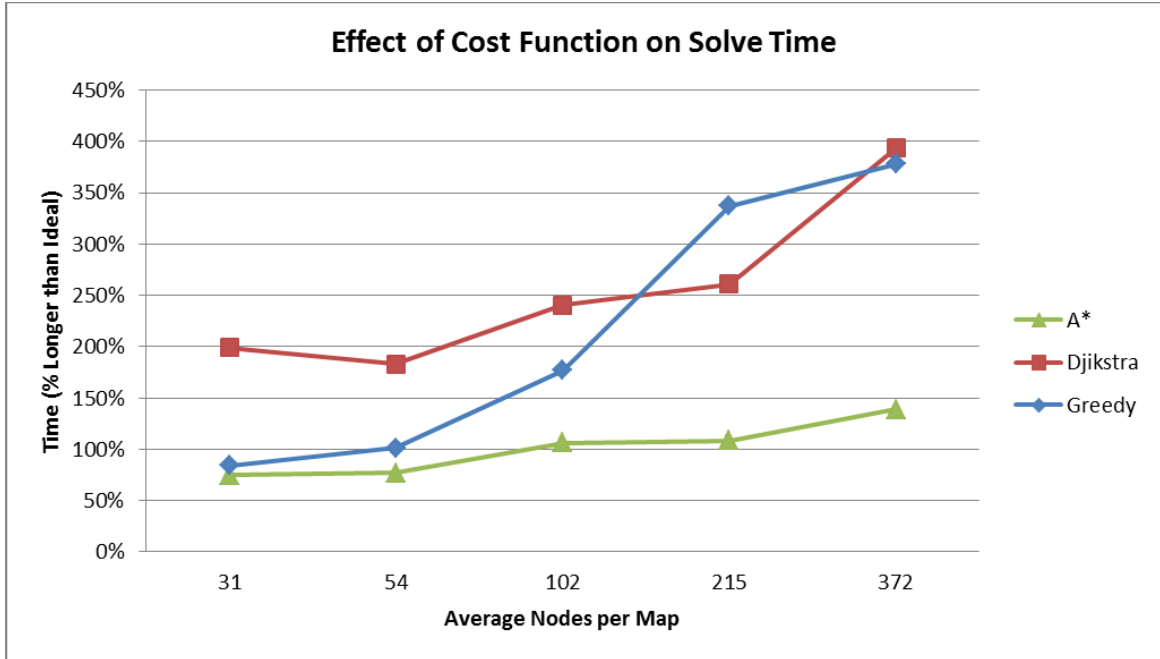


Figure 11: Effect of Cost Function on Solve Time

As expected, the A* cost function consistently shows the quickest solve time. It also has the most consistent performance across the range of map sizes. The greedy cost function mimics a depth-first search and the Dijkstra cost function has more in common with a breadth-first search. The A* cost function combines these two in an attempt to find an effective balance between depth and breadth. If the goal of the search is changed to be the efficient exploration of the map, the Dijkstra cost function may be more effective, as the results from these test cases show that it uncovers more nodes during the search than either of the other cost functions (see 0 for details).

The correlation between number of agents and solve time shown in Figure 12, again neglecting any possible interaction with other parameters.

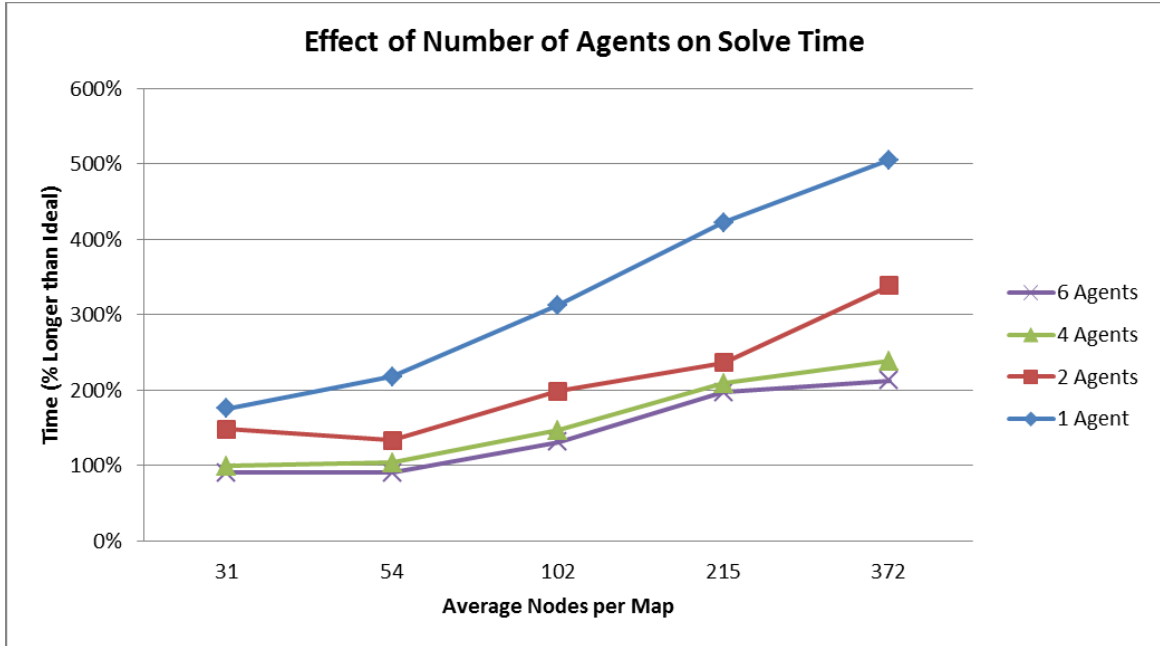


Figure 12: Effect of Number of Agents on Solve Time - Data Set 1

As expected, solve time generally decreases with an increased number of agents. The difference in solve time with respect to number of agents increases significantly with increased map size, but becomes less significant as more agents are added. For any given map there likely exists a certain number of agents for which time efficiency is nearly maximized and where the addition of more agents results in little or no significant decrease in solve time. This must be at least theoretically true since each map has a discrete number of nodes. However, the maximum useful number of agents is likely to be much less than this number, given the near-convergence seen in Figure 12 between four and six agents.

Figure 13 shows the effects of the number of agents on solve time when the communication distance is held to the consistent percentage of the map sizes.

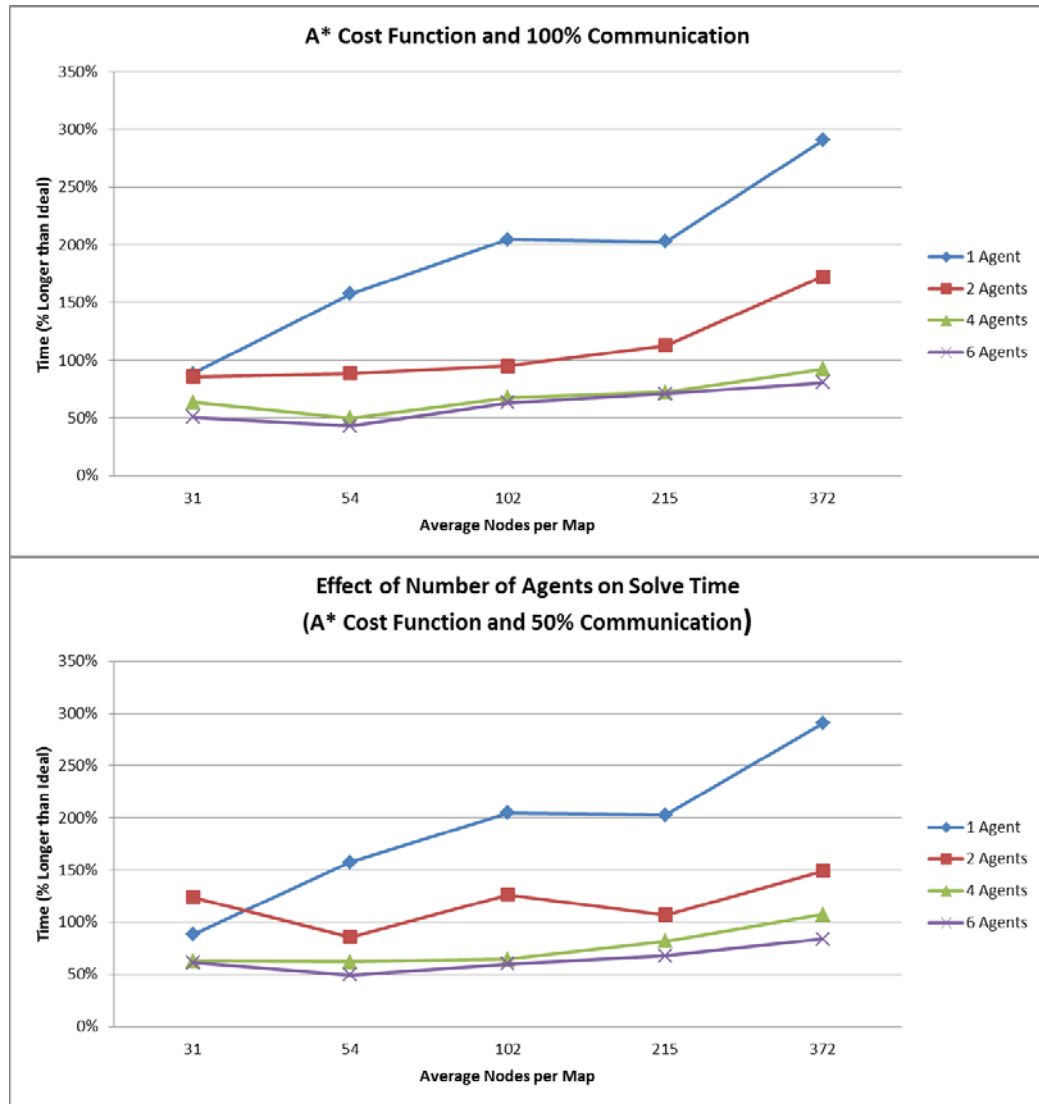


Figure 13: Effect of Number of Agents on Solve Time (A* Cost Function and Specified Communication Distances) – Data Set 1

The results of Figure 13 are similar to those shown in Figure 12, with increasing numbers of agents correlating to a decreasing solve time. This trend is most significant for larger maps, which is intuitive and expected. However, the second plot of Figure 13 shows that communication distance has a minimal effect on these trends.

This apparent consistency between the communication distances is shown more clearly when the number of agents used is not accounted for, as shown in Figure 14, and

when communication distance is examined without reference to any other factor (Figure 15).

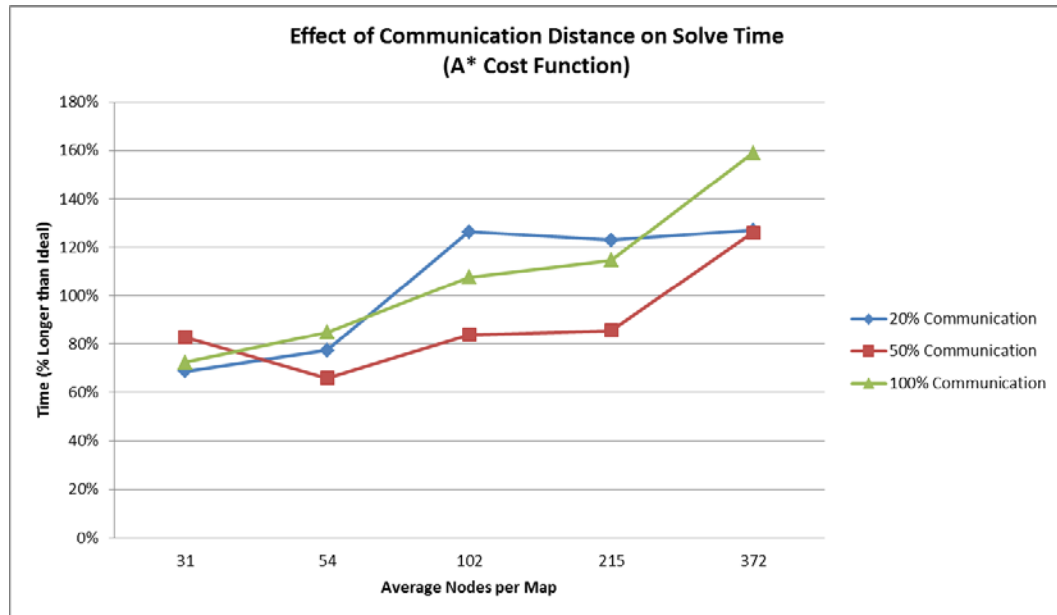


Figure 14: Effect of Communication Distance on Solve Time (A* Cost Function) – Data Set 1

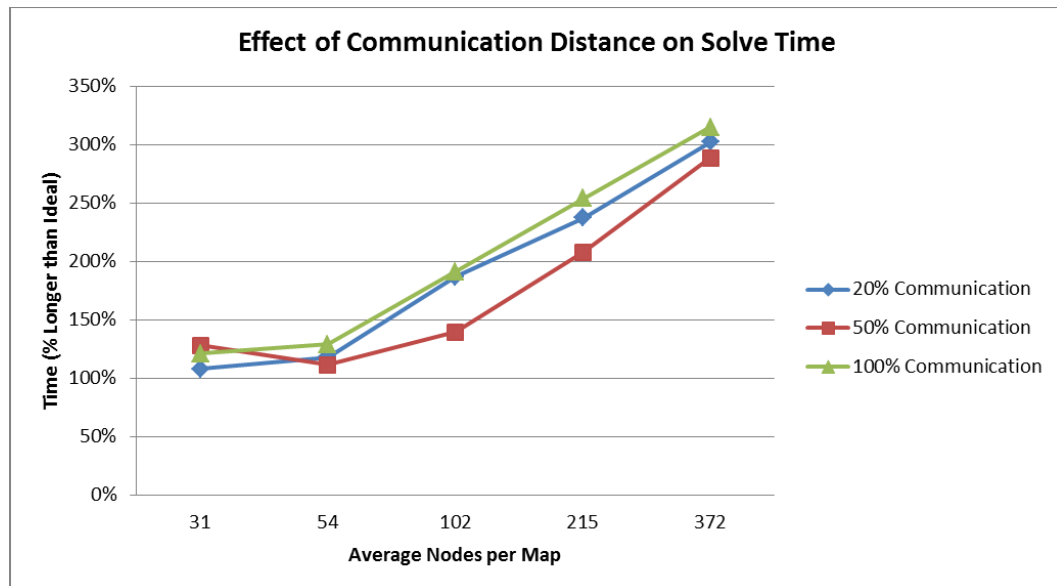


Figure 15: Effect of Communication Distance on Solve Time – Data Set 1

In these figures, although a moderate communication distance appears to be generally most effective, there is no clear indication that any one of the proposed communication coverages is most efficient for all map sizes.

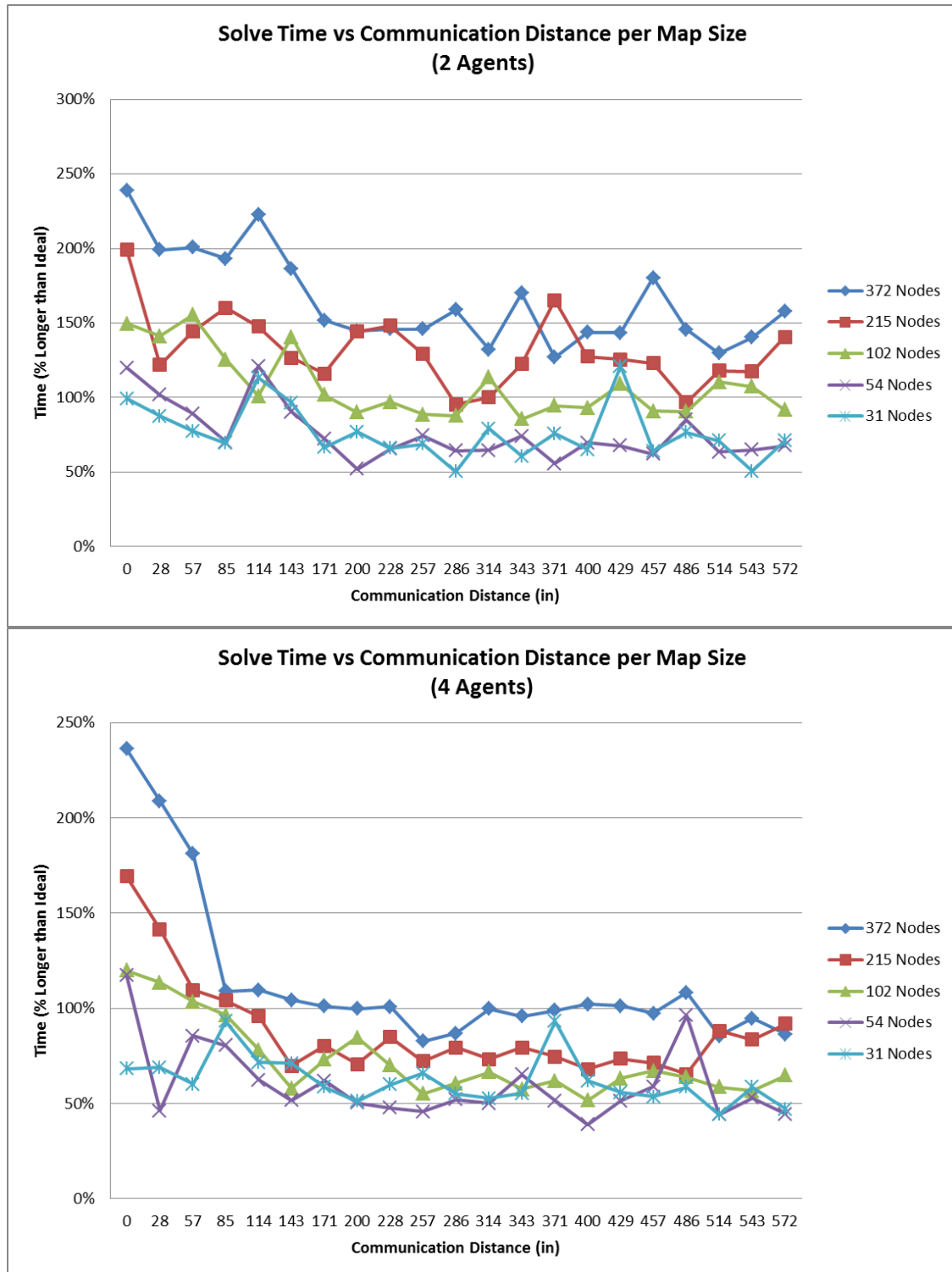
It is theorized that the possible preference for the moderate communication distance (50% of map area) may be due to the fact that with limited communications, each agent rarely updates its intermediate target based on information from other agents, while full coverage can lead continuously changing their intermediate targets without making any real progress towards the goal.

Figure 14 and Figure 15 also seem to indicate that communication distance has the greatest effect on the mid-sized maps. Since the map sizes are somewhat arbitrary, with the minimum and maximum sizes having no correlation to any particular real-life scenarios, this result is somewhat suspect.

Due to the overall lack of significant correlation between communication distance and solve time, and to the suspect result that the mid-size maps are most sensitive to communication distance, it was decided that the communication distances used should be refined in order to better understand these conclusions.

As discussed at the beginning of this chapter, this second data set uses communication distances at finer intervals and has those distances held constant across all map sizes (as opposed to being a function of map size). These attributes were chosen in order to make any potential trend more apparent and to allow identification of the most efficient communication distance (if any) for each map size. Also, since the chosen distances are based on the largest map, there should be a clear cut-off on the smaller maps where the solve time levels off as the allowable communication distance exceeds

100% of the map area. The results from this section data set are shown in Figure 16 (A*-based cost function, only).



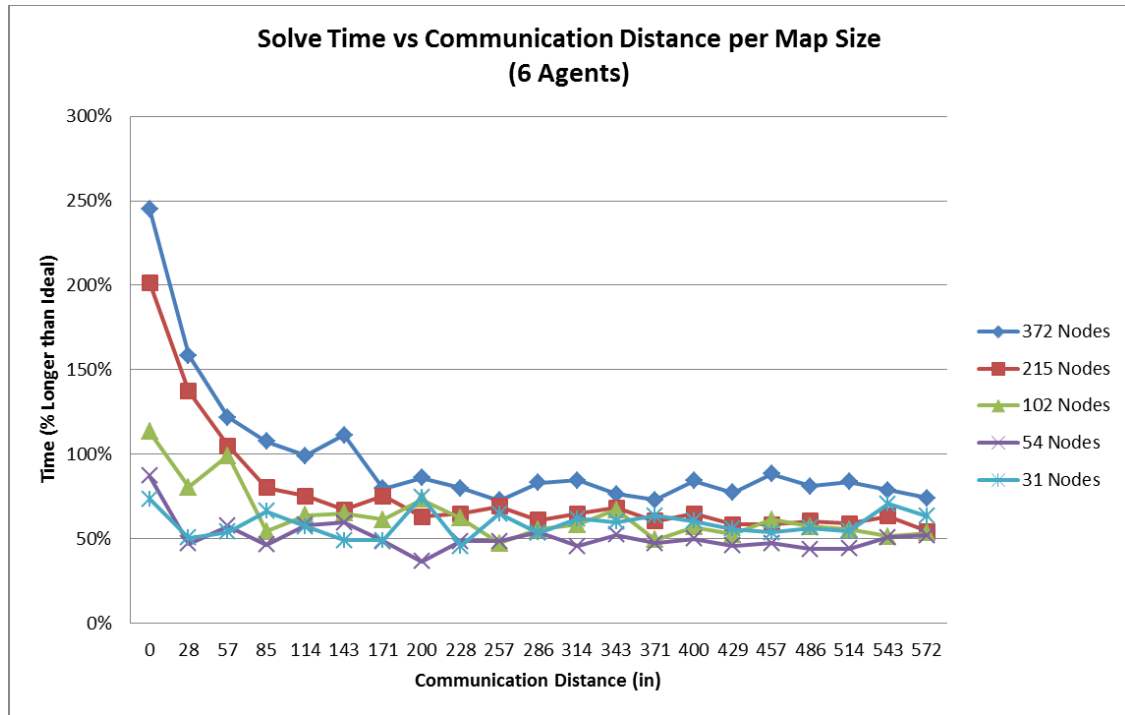


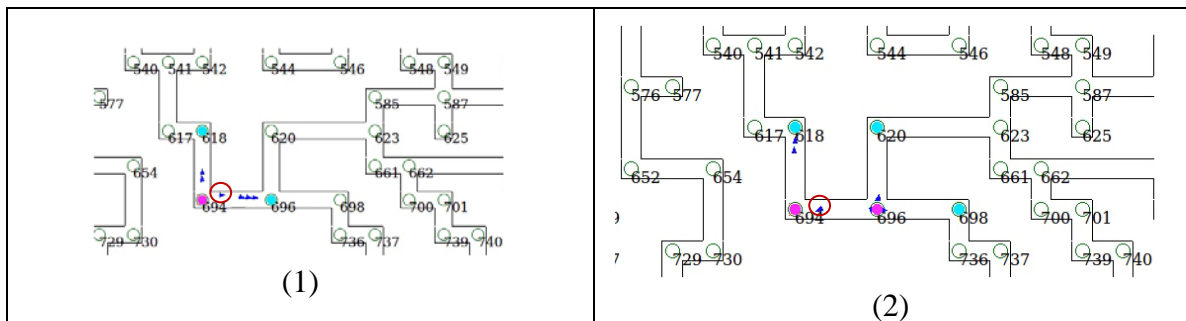
Figure 16: Solve Time vs Communication Distance per Map Size for Specified Numbers of Agents

These three charts demonstrate the correlation between the communication distance and number of agents. In each multi-agent case considered, the solve time decreases as the communication distance is increased. The data sets for all three numbers of agents show some decreasing trend in solve time with increasing communication distance, particularly between zero and the smaller distances. Each also shows decreasing solve time with decreasing map size. However, as the number of agents increases the significance of greater communication increases. Furthermore, the solve times for the larger maps decrease significantly with increased communication distance while the times for the smaller maps remain relatively constant. The effect of the number of agents and communication distance is also most significant for the largest maps. The solve times for these maps seem to settle around a communication distance of about 171 inches or less. At communication distances beyond this, the solve times remain relatively constant,

with further increases in communication having no greater benefit to solve time. The distance of 171 inches corresponds to the radius of a circle with an area equal to approximately 30% of the area of the largest set of maps.

Unexpected Behaviors

Although many aspects of this method have yielded expected results in terms of a greater number of agents, more communication, and use of the A*-based cost function being generally more effective, there are some instances in which the agents behave in an unexpected or less than ideal manner. These cases are related to the fact that any given agent can be forced to reevaluate its intermediate target node at any time if it is determined that it is more beneficial for another to make use of the agent's current intermediate target node. When agents spread out in several directions to explore different sections of the map, it is possible that the lowest cost node for one particular agent changes continually from one section to another as other agents discover new nodes and then determine that they, being closer, are more suited to explore them. This results in a looping behavior that ties up one agent and causes it to not be a contributor to the search. An example of this looping behavior is shown in Figure 17 with the agent circled.



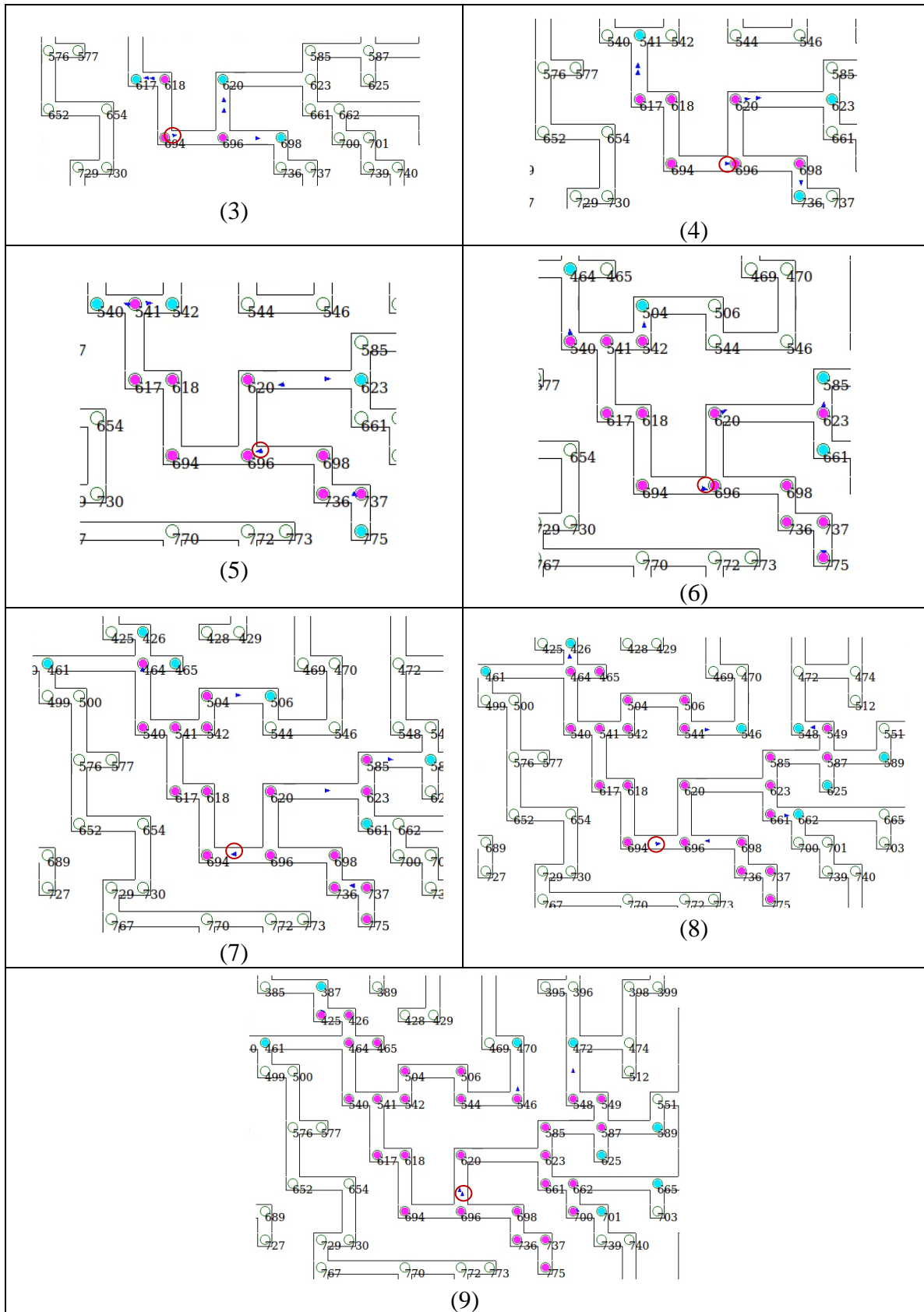


Figure 17: Looping Behavior

A second type of undesirable behavior can also result from an agent being redirected, but causes that agent to take a less efficient path in exploring new nodes. This is shown in Figure 18.

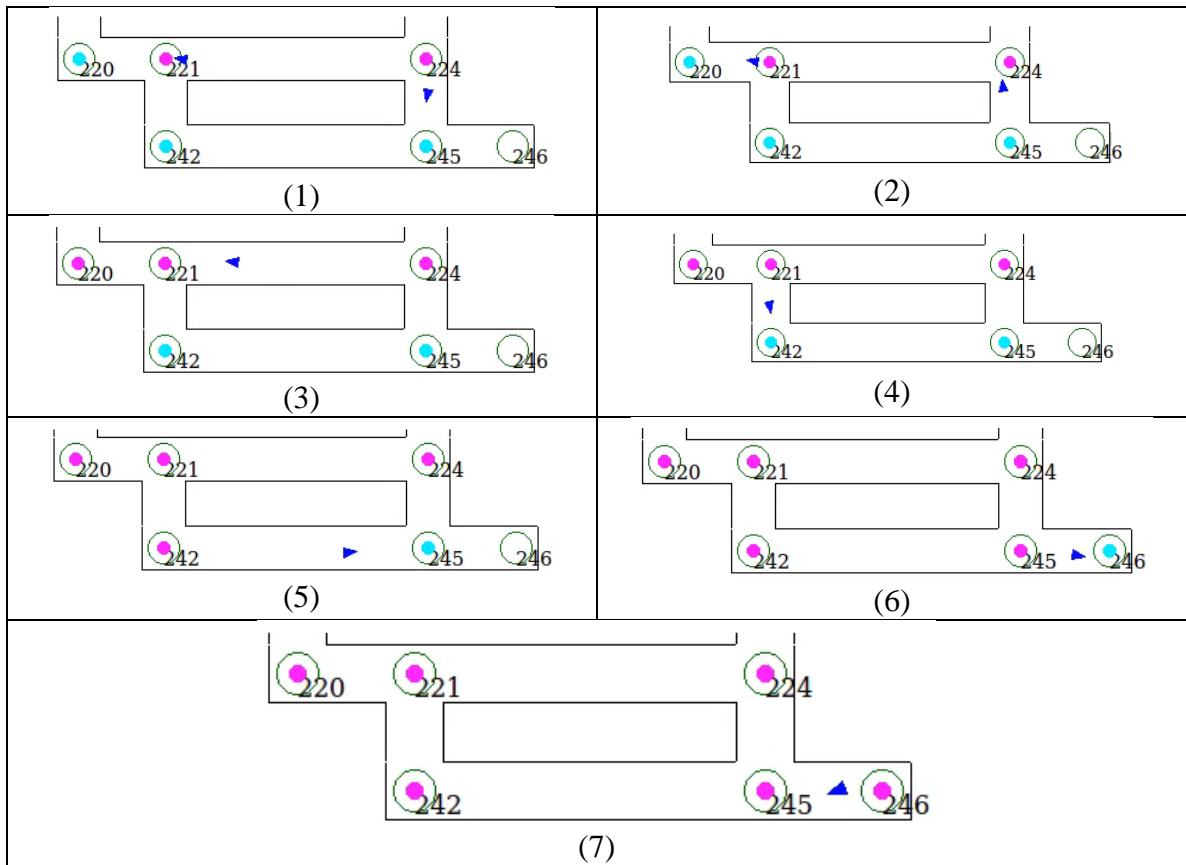


Figure 18: Less Efficient Path Behavior

In this scenario, an agent is initially heading towards node 245 when another agent discovers nodes 220 and 242. This second agent selects to head toward node 220, leaving node 242 free to be explored by the first agent. The first agent, determining that node 242 is more beneficial than node 245, which it is already heading towards, turns around and drives the only currently known path to node 242. This path ends up being longer than it would have been had the agent continued driving towards node 245 and

discovered the connection between nodes 245 and 242. Had the agent continued to node 245 until that node was reached, its path would have been 245, 246, 242 instead of 224, 221, 242, 245, 246 and the agent would have travelled a lesser distance.

Chapter V

Conclusions and Recommendations

While the trends identified in Chapter IV are suggestive in terms of the general effects of the cost function, number of agents, and communications range used on various map sizes on the time required to locate a target, there are many more factors that need to be considered for practical applications. These include deviations from the assumed environment, differences among agents and the mechanical limitations of the agents, and the particular demands of

Conclusions

In general, three basic conclusions can be drawn from the results presented above about how the various parameters affect the time required for an agent to reach the target. Firstly, that for the given problem of minimizing the time to acquisition of a specified target, the A*-based cost function performs better than the others addressed; second, that a greater number of agents tends to reduce solve time, particularly for larger environments; and finally, that providing some level of communication between agents also reduces the solve time. Additionally, it is concluded that it would be beneficial to investigate methods by which the undesirable behavior described in the section on Unexpected Behaviors can be mitigated.

The A*-based cost function attempts to balance breadth and depth in selecting agents' paths. It is overall both the quickest for getting agents to the target and shows the least variation in solve times when other factors such as map size and number of agents are considered. The cost function that is most appropriate for other types of problems should be chosen to match the particular challenges and goals of those problems. As

mentioned above in the Results and Discussion, there is some evidence that the Dijkstra-based cost function may be more efficient in completing a complete exploration of the maps used in this investigation. Cost functions other than those considered here may be more appropriate to problems addressing other types of goals or using different environment configurations.

Although the use of additional agents tends to decrease the solve time for all map sizes, it should be noted that the spread-out start positions of multiple agents (see the section on Agent Starting Pose, page 38) likely exaggerates the efficiency to a small degree in the smaller maps. This effect, while still present, is much less significant on the larger maps since the overall distance to be traveled by any agent is greater and since there is a higher probability that the agent that eventually finds the target did not start with a clear positional advantage. Even given this source of error, it is clear that using a larger number of agents is significantly more effective when larger maps are considered. Also, for any given environment size, it is likely that a maximum number of agents can be found beyond which there is no significant benefit to performance.

A similar trend is seen when considering communication distance. Overall, communication over a greater area of the environment results in lower solve times. At the same time, scenarios using larger maps and greater numbers of agents benefit more for an increase in communication distance than do those run on smaller maps and/or with fewer agents. There is also generally a steep increase in performance when progressing from no communication through the smaller distances. Again, this is more significant with larger maps and more agents.

In order to mitigate the undesirable behaviors that cause looping and inefficient path selection it is suggested that an agent be required to reach its intermediate target node before selecting a new one. It may not be beneficial for an agent to reevaluate while driving, as this can lead to the looping and inefficient path behaviors discussed in the section on Unexpected Behaviors (page 57). By forcing an agent to drive to its node, that agent will be closer to the unexplored areas of the map and therefore will be able to fully participate in the search. Since, in some cases, the agent in question may have a significant distance to travel to reach its intermediate target node, it may be necessary to create a provision for other agents to explore the paths connected to that node in order to allow the search to progress while the agent in question is travelling.

Recommendations for Further Investigation

Changes in Assumed Environment

It is assumed in this investigation that the environments conform to a specific format. The maps are all regular and grid-based, all locations of interest are intersections, all nodes denote intersections, and there are no changes to the environment except for small obstacles (such as other agents) that are easily navigated around and do not impair an agent's knowledge of its surroundings. While this model is appropriate an initial investigation of the factors influencing the path-planning method presented in this paper, many practical applications cannot be well described using this narrow environmental model. It would be useful to understand how changes in these assumptions about the nature of the environment affect the efficiency of the path-planning method. This might then be expanded to include investigations into how the effects of the number of agents and communications range change when considering these other types of environments.

Maximum Efficiencies

As noted the Results and Discussion, for any given map there likely exists a certain number of agents and a certain range of communication for which time efficiency is nearly maximized and where the addition of more agents and/or an increase in communications range has little or no benefit. It would be interesting to further describe the correlation between environment size, number of agents, and communication distance and to determine if it is possible to define sets of parameters that will maximize efficiency for given map sizes.

Non-Homogenous Agent Groups

In all the scenarios considered in this paper, each agent is identical to every other agent in every way, including size, speed, maneuverability, and sensing capabilities. In practical applications this may not always be the case, whether due to inherent differences between individual pieces of hardware or due to some intentionally sought benefit gained by using agent with different characteristics. It would be useful to understand the impact of slight inconsistencies between agents when the agents are assumed to be exactly identical (ie. no compensation in the node selection method). It would also be interesting to modify the node selection algorithm to account for differences in agent speeds, communication ranges, or other characteristics with the aim to improve performance under particular circumstances.

Agent Failure Rates

In practical application, all hardware is subject to failure, the rate of which can be estimated using published historical data or life-cycle testing. A multiple-agent system inherently has some level of redundancy, but the loss of one or more agents may reduce

the efficiency of the group as a whole. An investigation into the costs and benefits of using additional agents beyond the theoretical maximum suggested in the Results and Discussion and the section on Maximum Efficiencies would be useful in addressing this. It may also be possible to design a method by which specific agents are designated as ‘recovery agents’ and positioned at strategic locations throughout the known environment.

Unclear Target Location

It is likely that in practical applications it may not be possible to initially get a reliable or accurate target location. It may be that the best that can be done is to determine a general direction or a general direction and imprecise distance. It would be of interest to construct a method by which the target’s location could be refined as agents get closer and uncover more information about the environment. This would result in a changing target location, although the feature of interest would not itself be moving.

References

- [1] Steven Kibler and Dejan Raskovic, "Coordinated Multi-Robot Exploration of a Building for Search and Rescue Situations " in *44th IEEE Southeastern Symposium on System Theory*, 2012, pp. 159-163.
- [2] K. Guruprasad and D. Ghose, "Automated multi-agent search using centroidal voronoi configuration," *Automation Science and Engineering, IEEE Transactions on*, vol. 8, no. 2, pp. 420-423, April 2011.
- [3] F. Zuiani and M. Vasile, "Multi Agent Collaborative Search Based on Tchebycheff Decomposition," in *Computational Optimization and Applications*, 2013, pp. 189-208
- [4] Shadi Alian, Nazeeh Ghatasheh, and Mua'ad Abu-Faraj. "Multi-Agent Swarm Spreading Approach in Unknown Environments." *International Journal of Computer Science Issues*, vol. 11, issue 2, no 2, pp. 160-168, March 2014.
- [5] Emre Ugur, Ali E. Turgut, and Erol Şahin. "Dispersion of a Swarm of Robots Based on Realistic Wireless Intensity Signals.", *22nd International Symposium on Computer and Information Sciences*, IEEE, 2007.
- [6] Andrew Howard, Maja J. Matarić, and Gaurav S. Sukhatme. "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks." *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, vol. 13, pp. 113-126, Sept 2002
- [7] Marcelo Oscar Sztainberg. "Algorithms for Swarm Robotics." Applied Mathematics and Statistics, State University of New York at Stony Brook, 2003.
- [8] Geunho Lee and Nak Young Chong. "Self-configurable Mobile Robot Swarms with Hole Repair Capability." in *IEEE International Conference on Intelligent Robots and Systems*. 2008, pp. 1403-1408.
- [9] Dimos V. Dimarogonas and Kostas J. Kyriakopoulos. "An Inverse Agreement Control Strategy with Application to Swarm Dispersion." In *46th IEEE Conference on Decision and Control*, 2007, pp. 6148-6153.
- [10] Zhao Pengchong, Liang Alei, Liu Liang, Chen Ying, Guan Haibing, and Yan Xinan. "An Exploration Algorithm for a Swarm of Homogeneous Robots." in *IEEE International Conference on Computational Intelligence and Software Engineering*, 2009, pp. 1-6.
- [11] M. Baglietto, M. Paolucci, L. Scardovi, and R. Zoppoli. "Information Based Multi-Agent Exploration." in *IEEE International Workshop on Robot Motion and Control*, 2002, pp. 173-179
- [12] Ionel Muscalagiu, Cosmina Illes, and Horia Emil Popa. "Large Scale Multi-Agent-Based Simulation using NetLogo for the Multi-Robot Exploration Problem." in *IEEE 11th International Conference on Industrial Informatics*, 2013, pp. 225-230.
- [13] Yiheng Wang, Alei Liang, and Haibing Guan. "Frontier-based Multi-Robot Map Exploration Using Particle Swarm Optimization." in *IEEE Symposium on Swarm Intelligence*, 2011, pp. 1-6.
- [14] Christian G. Quintero M., José A. Oñate López, Francisco A. Bertel R. "Coordination Mechanisms for a Multi-Agent Robotic System Applied to Search and Target Location." in *9th Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control*, 2011, pp. 1-6

- [15] Julian de Hoog, Sephen Cameron, and Arnoud Visser. "Role-Based Autonomous Multi-Robot Exploration." in *International Conference on Advanced Cognitive Technologies and Applications*, November 2009
- [16] Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. "Collabrative Multi-Robot Exploration." in *IEEE International Conference on Robotics & Automation*, 2000, pp. 476-481.
- [17] Alejandro Torreño, Eva Onaindia, and Óscar Sapena. "FMAP: Distributed Cooperative Multi-Agent Planning." *Applied Intelligence*, vol. 41, no 2, pp. 606-626, 2014. <http://link.springer.com/article/10.1007/s10489-014-0540-2#>
- [18] Jean Berger and Nassirou Lo. "An Innovative Multi-Agent Search-and-Rescue Path Planning Approach". *Computers & Operations Research*, vol. 53, pp. 24-31, 2015
- [19] Timothy H. Chung and Joel W. Burdick. "Multi-Agent Prbabilistic Search in a Sequential Decision-Theoretic Framework." in *IEEE International Conference on Robotics and Automation*, 2008, pp. 146-151.
- [20] Marcelo Cardoso Silva, Ana Cristina Bicharra Garcia, and Aura Conci. "A Multi-Agent System for Dynamic Path Planning." in *Second Brazilian Workshop on Social Simulation*, 2010, pp. 47-51.
- [21] James Kennedy and Russell Eberhart. "Particle Swarm Optimization." in *IEEE International Conference on Neural Networks*, vol. 4. pp. 1942-1948, 1995.
- [22] Russell Eberhart and James Kennedy. "A New Optimizer Using Particle Swarm Theory." in *Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39-43.
- [23] Helbert Eduardo Espitia and Jorge Iván Sofrony. "Vortex Particle Swarm Optimization". in *IEEE Congress on Evolutionary Computation*, 2013, pp. 1992-1998.
- [24] Aleksandar Jevtić, Peymon Gazi, Diego Andina, Mo Jamshidi. "Building a Swarm of Robotic Bees." in *World Automation Congress*, 2010, pp. 1-6
- [25] Sanjay Sarma O. V., Vishwanath Lohit T., and Deepak Jayaraj. "Path Planning in Swarm Robots using Particle Swarm Optimization on Potential Fields." *International Journal of Computer Applications*, vol. 60, no. 13, December 2012.
- [26] Adham Atyabi, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. "Navigating a Robotic Swarm in an Uncharted 2D Landscape." *Applied Soft Computing*, vol. 10, pp 149-169, 2010
- [27] Ma Qian-zhi and Lei Xiu-juan. "The Application of Hybrid Orthogonal Particle Swarm Optimization in Robotic Path Planning." in *Sixth International Conference on Natural Computation*, vol. 7, 2010, pp. 3536-3540.
- [28] Ganesh K. Venayagamoorthy, Lisa L. Grant, and Sheetal Doctor. "Collective Robotic Search using Hybrid Techniques: Fuzzy Logic and Swarm Intelligence Inspired by Nature." *Engineering Applications of Artificial Intelligence*, vol. 22, pp. 431-441, 2009.
- [29] James M. Hereford. "A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications." presented at *IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, July 2006.
- [30] Adham Atyabi and David M. W. Powers. "The Use of Area Extended Particle Swarm Optimization (AEPSO) in Swarm Robotics." presented at *International*

- Conference on Control, Automation, Robotics, and Vision*, Singapore, December 2010.
- [31] James M. Hereford, Michael Siebold, Shannon Nichols. "Using the Particle Swarm Optimization Algorithm for Robotic Search Applications." in *IEEE Swarm Intelligence Symposium*, 2007, pp. 53-59.
 - [32] Ettore Ferranti, Niki Trigoni, and Mark Levene. "Brick&Mortar: an On-line Multi-Agent Exploration Algorithm." in *IEEE International Conference on Robotics and Automation*, 2007, pp. 761-767.
 - [33] Khalid Al-Mutib, Mansour AlSulaiman, Muhammad Emaduddin, Hedjar Ramdane, and Ebrahim Mattar. "D* Lite Real-Time Multi-Agent Path Planning in Dynamic Environments." in *Third International Conference on Computational Intelligence, Modelling and Simulation*, 2011, pp. 170-174.
 - [34] Gireesh Kumar T., K. J. Poornaselvan, and M. Sethumadhavan. "Fuzzy Support Vector Machine-based Multi-agent Optimal Path Planning Approach to Robotics Environment." *Defense Science Journal*, vol. 60, no. 4, pp. 387-391, July 2010.
 - [35] Zhao Lihui. "On the Techniques of Multi-agent Path Planning and the Collaboration." in *International Conference on Computer Application and System Modeling*, vol. 13, 2010, pp. 430-433.
 - [36] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. "Conflict-based Search for Optimal Multi-Agent Pathfinding." *Artificial Intelligence*, vol. 219, pp. 40-66, 2015.
 - [37] Qichen Wang and Chris Phillips. "Cooperative Path-Planning for Multi-Vehicle Systems." *Electronics*, vol. 3, no. 4, pp. 636-660, 2014.
 - [38] Wenje Wang and Wooi-Boon Goh. "A Meta-Heuristic Approach for Energy-Efficient Multi-Agent Path Planning." In *International Conference on Artificial Intelligence*, Athens, 2013.
 - [39] Wenje Wang and Wooi-Boon Goh. "A Stochastic Algorithm for Makespan Minimized Multi-Agent Path Planning in Discrete Space." *Applied Soft Computing*, vol. 30, pp. 287-307, 2015.
 - [40] Jingjin Yu and Steven M. LaValle. "Multi-agent Path Planning and Network Flow." *Algorithmic Foundations of Robotics X. Springer Berlin Heidelberg*, pp. 157-173, Jan 2013.
 - [41] Jingjin Yu. "Shortest Path Set Induced Vertex Ordering and its Application to Distributed Distance Optimal Multi-agent Formation Path Planning." arXiv preprint arXiv: 1205.0207 Sep 2012
 - [42] Ko-Hsin Cindy Wang and Adi Botea. "MAPP: a Scalable Multi-Agent Path Planning Algorithm with Tractability and Completeness Guarantees." *Journal of Artificial Intelligence Research*, vol. 42, pp. 55-90, 2011.
 - [43] Heba Gaber, Safaa Amin, and Abdel-Badeeh M. Salem. "A Combined Coordination Technique for Multi-Agent Path Planning." in *10th International Conference on Intelligent Systems Design and Applications*, 2010, pp. 563-568.
 - [44] David Silver. "Cooperative Pathfinding." in *AIIDE*, 2005, pp. 117-122.
 - [45] Jianlin Zhong and Andrzej Maslowski. "Control and Path Planning for Multi-Agent Mobile Vehicles System." in *International Conference on Measuring Technology and Mechatronics Automation*, vol. 1, 2010, pp. 841-844.

- [46] Abu Bakar Sayuti Saman and Issa Abdramane. "Solving A Reconfigurable Maze using Hybrid Wall Follower Algorithm." *International Journal of Computer Applications*, vol. 82, no. 3, Nov 2013.
- [47] Hongshe Dang, Jinguo Song, and Qin Guo. "An Efficient Algorithm for Robot Maze-Solving." in *Second International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 2, pp. 79-82, 2010.
- [48] A. Francy Golda, S. Aridha, and D. Elakkiya. "Algorithmic Agent for Effective Mobile Robot Navigation in an Unknown Environment." in *International Conference on Intelligent Agent & Multi-Agent Systems*, 2009, pp. 1-4.
- [49] Zhuang Cai, Lu Ye, and Ang Yang. "FloodFill Maze Solving with Expected Toll of Penetrating Unknown Walls for Micromouse." in *High Performance Computing and Communication & 9th International Conference on Embedded Software and Systems*, 2012, pp. 1428-1433.
- [50] George Law. "Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms." *International Journal of Computer Theory and Engineering*, vol. 5, no. 3, June 2013
- [51] Matthias Becker, Florian Blatt, and Helena Szczerbicka. "Agent-Based Approaches for Exploration and Pathfinding in Unknown Environments." in *17th Conference on Emerging Technologies and Factory Automation*, 2012, pp. 1-4.
- [52] Abu Bakar Sayuti Saman and Issa Abdramane. "Solving a Reconfigurable Maze using Hybrid Wall Follower Algorithm." *International Journal of Computer Applications*, vol. 82, no. 3, November 2013.
- [53] Adil M. J. Sadik, Maruf A. Dhali, Hasib M. A. B. Farid, Tafhim U. Rashid, and A. Syeed. "A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory." in *IEEE International Conference on Artificial Intelligence and Computational Intelligence*, vol. 1, pp.52-56, 2010.
- [54] Xiang Liu and Daoxiong Gong. "A Comparative Study of A-star Algorithms for Search and Rescue in Perfect Maze." in *International Conference on Electric Information and Control Engineering*, 2011, pp. 24-27.
- [55] Behnam Rahnema and Atilla Elçi. "Design and Implementation of a Novel Weighted Shortest Path Algorithm for Maze Solving Robots." in *IEEE 37th Annual Computer Software and Applications Conference Workshops*, 2013, pp. 328-332.
- [56] Elad H. Kivelevitch and Kelly Cohen. "Multi-Agent Maze Exploration." *Journal of Aerospace Computing, Information, and Communication*, vol. 7, no. 12, pp. 391-405, Dec 2010.
- [57] G. Tarry. "La Problem des Labyrinths." *Nouvelles Annales de Mathématiques*, vol. 14, pp. 187-190, 1895.
- [58] Behnam Rahnema, Atilla Elçi, and Cankat Özermen. "Design and Implementation of Cooperative Labyrinth Discovery Algorithms in Multi-Agent Environment." in *International Conference on Technological Advances in Electrical, Electronics and Computer Engineering*, 2013, pp. 573-578.
- [59] T. R. Wan, H. Chen, and R. Earnshaw. "Real-time Path Planning for Navigation in Unknown Environment." in *Proceedings of the Theory and Practice of Computer Graphics*, 2003, pp. 138-145.

- [60] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determinination of Minimum Cost Paths." *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, Issue 2, pp. 100-107, 1968.
- [61] E. W. Dijkstra. "A Note on Two Problems in Connection with Graphs". *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
- [62] Webots
- [63] Mocrosoft Robotics Developer Sudio
- [64] anyKode Marilou
- [65] EZ Physics
- [66] Lpz Robots
- [67] ANVEL
- [68] Morse
- [69] STDR

Appendix A

Graphical Simulation Environment

Required Input

Regardless of whether the simulation is run in command-line or demonstration mode, several pieces of information are required from the user. These are the map of the simulation environment, the number of agents and their speeds, the start node, the target node, the assumed communication distance, and the desired cost function. These variables are either input using the appropriate fields in the GUI or as data read from an input file.

Map Definition

Map information is represented by a connectivity graph. All maps must be created using the GUI, but may be saved and called later in either the demonstration simulation or the command-line process. When the map is created in the GUI, the user defines edges by selecting node pairs. Clicking anywhere in the environment space establishes the first node and clicking again establishes the second node. The first node may be placed anywhere within the environment space, but the second node is forced to be either horizontal or vertical with respect to the first node. Any number of edges may be placed within the environment space. The placement of an initial edge (first and second node) is shown in Figure19.

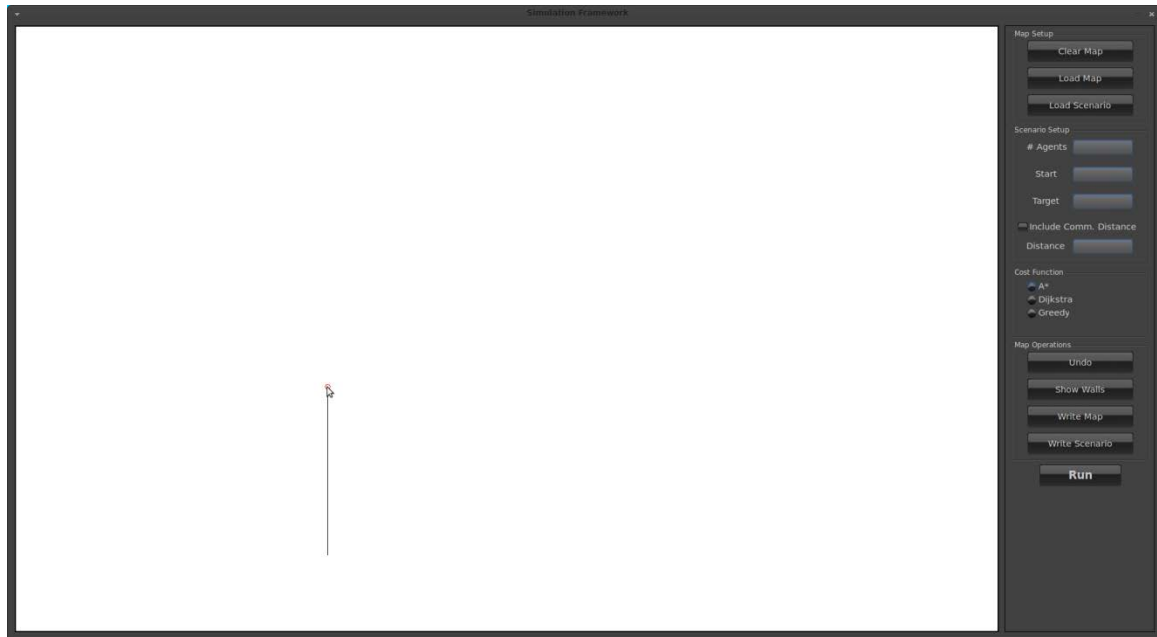


Figure19: Placing First Edge

To create intersections, the first node selected when creating a new edge must be coincident with a previously established node. When the cursor is within a specified distance of an established node, a red circle will appear around that node, indicating that a node placed by clicking at this position will result in an intersecting edge (Figure 20). A similar feature exists to indicate that a new node will be horizontal or vertical with respect to an existing node, as shown in Figure 20. Figure 21 shows an example of a complete connectivity graph.

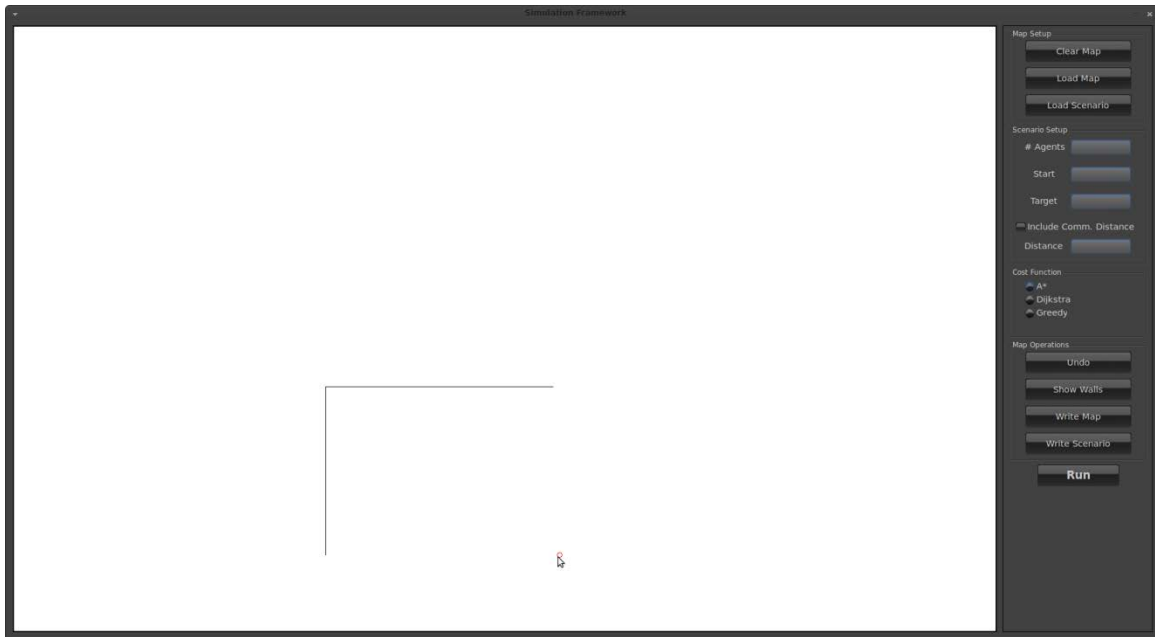


Figure 20: New Node Horizontal or Vertical with Respect to Existing Node

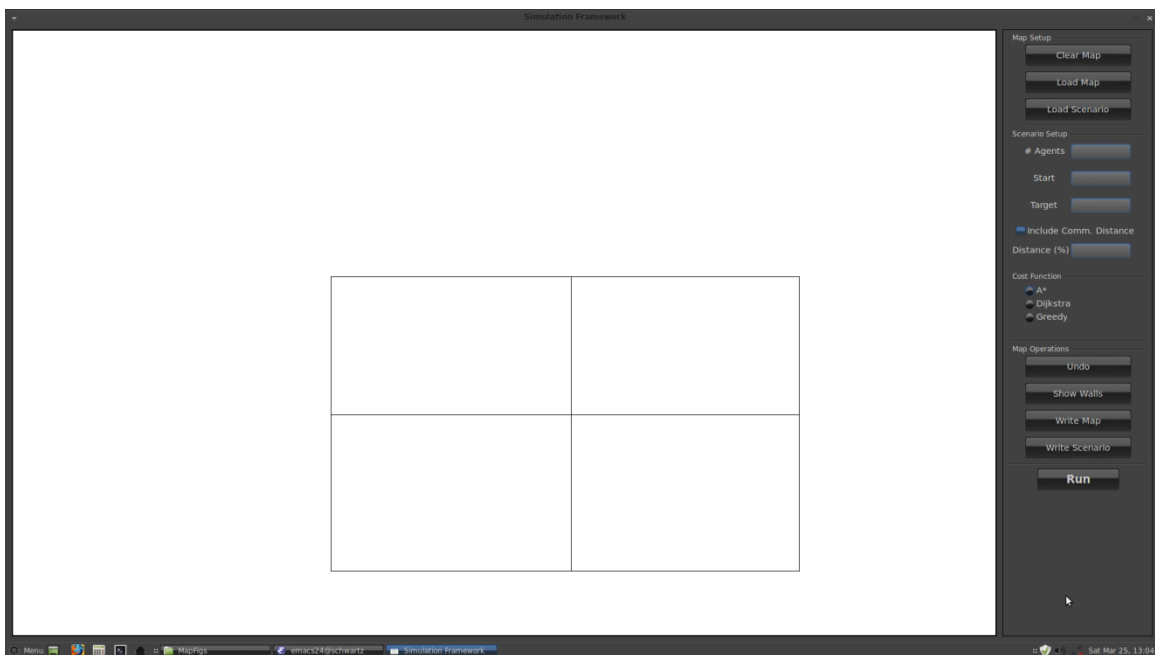


Figure 21: Complete Connectivity Graph

Once the map has been drawn, the user must click the “Show Walls” button. This will calculate and display the walls, and generate a wall potential field based off the newly created connectivity graph. The walls are set to a fixed width and are centered

around the established nodes and edges. The nodes have a radius of influence within which the agents are considered to have reached the node. This is set to be three-eighths of the wall width. A circle of influence for each node as well as assigned node number are also displayed when the “Show Walls” button is clicked. An example of the walls calculated from a defined connectivity graph is shown in Figure 22.

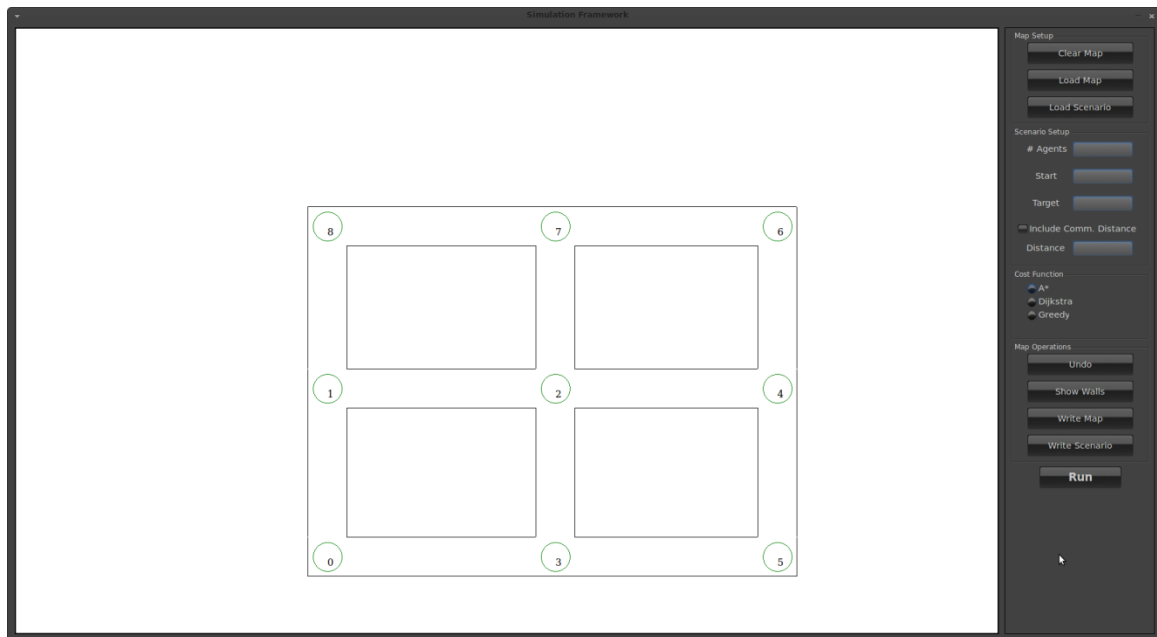


Figure 22: Complete Map

After defining a new map it may be saved as a *.map* file for future use in both demonstrative and command-line simulations. A *.map* file stores the connectivity graph, the wall definitions, the potential field, the wall width, the size of the map space and the scaling factor used to convert inch measurements to a representative number of pixels.

Running a Simulation

Simulation Set-Up. A simulation run using the GUI takes the same input as a command-line simulation. The user may either define a map as outlined above in the

section on Map Definition, or may select a map from any that have been previously saved. Once the map has been generated, the user must input the start and target nodes by referencing the appropriate node numbers shown in the environment space (see Figure 23). Once these have been selected, a red “X” will appear at the start node and a green circle at the target node. The start and target nodes may be changed at any time before beginning the simulation.

Before running the simulation, the user must also input the number of agents running at each speed, the assumed communication distance as a percentage of map area, and the cost function that should be used in planning the agents' paths.

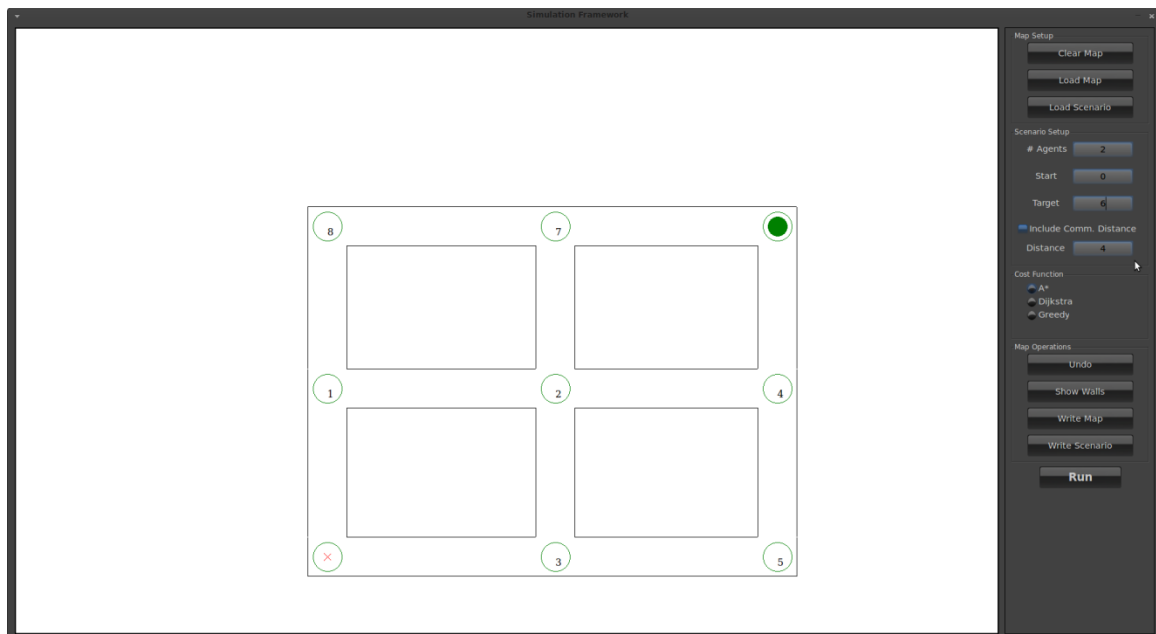


Figure 23: Graphical Simulation Interface after Defining Start Node

Appendix B

Mathematical Models

In the robot frame, the position and orientation of an individual wheel are expressed using the parameters l , α and β (Figure 24). l and α are the polar coordinate of the center of the wheel with respect to the vehicle frame, with α measured from the positive x -axis. β is the angle between the wheel's positive rotation axis and the direction of l . These parameters can be used to express the in-line and transverse components of the $[\xi_R]$ velocities.

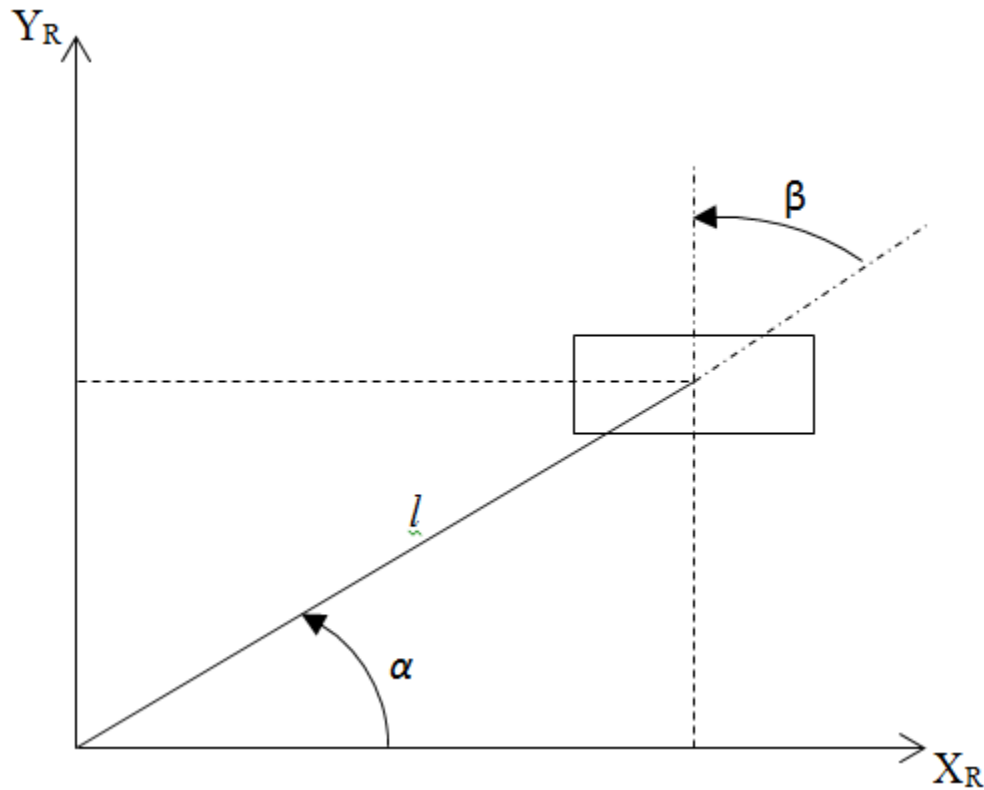


Figure 24: Position and Orientation of a Fixed Wheel

Since the wheel is assumed to undergo pure rolling, the components of velocity in the rolling direction must equal the total wheel velocity, which give rise to the rolling constraint equation.

$$[\sin(\alpha + \beta) \quad -\cos(\alpha + \beta) \quad (-l) \cos(\beta)]\{R(\theta)\}[\dot{\xi}_I] = r\dot{\psi} \quad (18)$$

Likewise, all the velocity components in the sliding direction must equal zero.

$$[\cos(\alpha + \beta) \quad \sin(\alpha + \beta) \quad (l) \sin(\beta)]\{R(\theta)\}[\dot{\xi}_I] = 0 \quad (19)$$

The expression for the vehicle's movement combines the rolling and sliding constraints for each wheel. Since the wheels are identical and located along the same axis the sliding constraints are the same and one (for example, the right) can be dropped. Combining the three remaining equations gives:

$$\begin{bmatrix} \sin(\alpha_L + \beta_L) & -\cos(\alpha_L + \beta_L) & -l \cos(\beta_L) \\ \sin(\alpha_R + \beta_R) & -\cos(\alpha_R + \beta_R) & -l \cos(\beta_R) \\ \cos(\alpha_L + \beta_L) & \sin(\alpha_L + \beta_L) & l \sin(\beta_L) \end{bmatrix} \{R(\theta)\}[\dot{\xi}_I] = \begin{bmatrix} r_L \dot{\psi}_L \\ r_R \dot{\psi}_R \\ 0 \end{bmatrix} \quad (20)$$

Finally, the above equation can be simplified to account for the fact that both drive wheels lie on the vehicle's y-axis. Since the left wheel lies on the positive y-axis and the right wheel on the negative y-axis, α_L and α_R are $\frac{+\pi}{2}$ and $\frac{-\pi}{2}$, respectively. The angle β is measured from the positive end of the line drawn through the wheels' axis of rotation.

This makes β zero for the left wheel and π for the right wheel. Since the wheels are not steerable β is constant.

Substituting these values and solving for $[\dot{\xi}_I]$ gives the final form of the kinematic equations for a simple differentially driven vehicle, as shown in Equation (13).

Appendix C

Full Map Exploration

While the A*-based node selection criteria may be the most efficient when the goal is to locate a given target in a minimum amount of time (see the Results and Discussion), other cost functions may be more appropriate when the objective of the search is changed. Figure 25 and Figure 26 show the effects each of the three cost functions discussed in the section on Best Node Criteria (page 27) have on the percentage of nodes visited and the percentage of nodes revealed, respectively. These show that during trials aimed at finding a given target, the Djikstra-based cost function caused the agents to visit and uncover significantly more nodes than either of the other cost functions. This is reasonable given that the Djikstra-based cost function will bias an agent toward those unvisited nodes which are nearest its current location. The Djikstra-based cost function does not consider the location of the target in any way. The Greedy-based cost function, by contrast, is only influenced by the target location and will bias an agent toward unvisited nodes that are nearest the target, while the A*-based cost function attempts to balance both of these approaches.

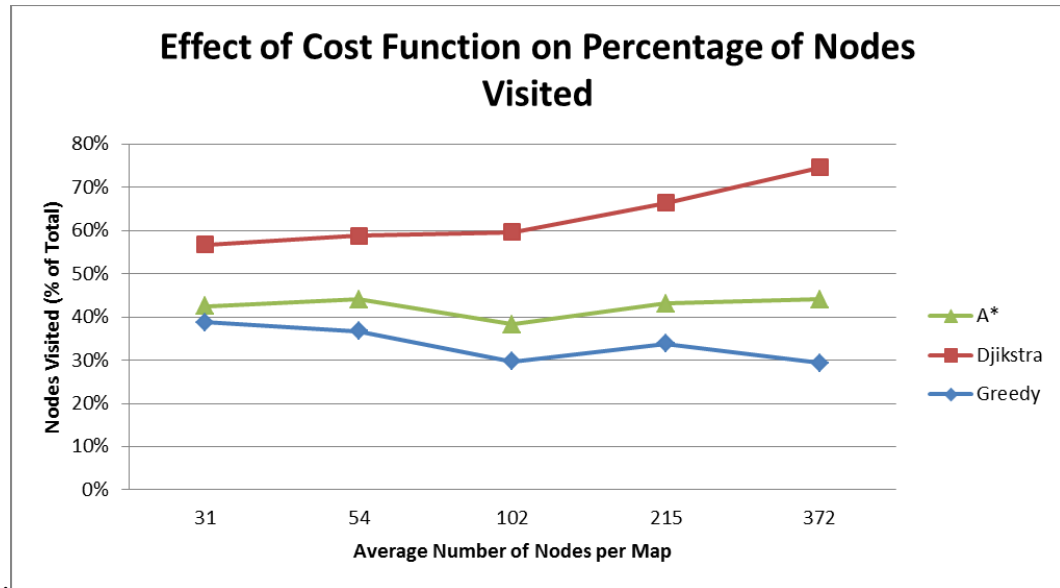


Figure 25: Effect of Cost Function on Nodes Visited

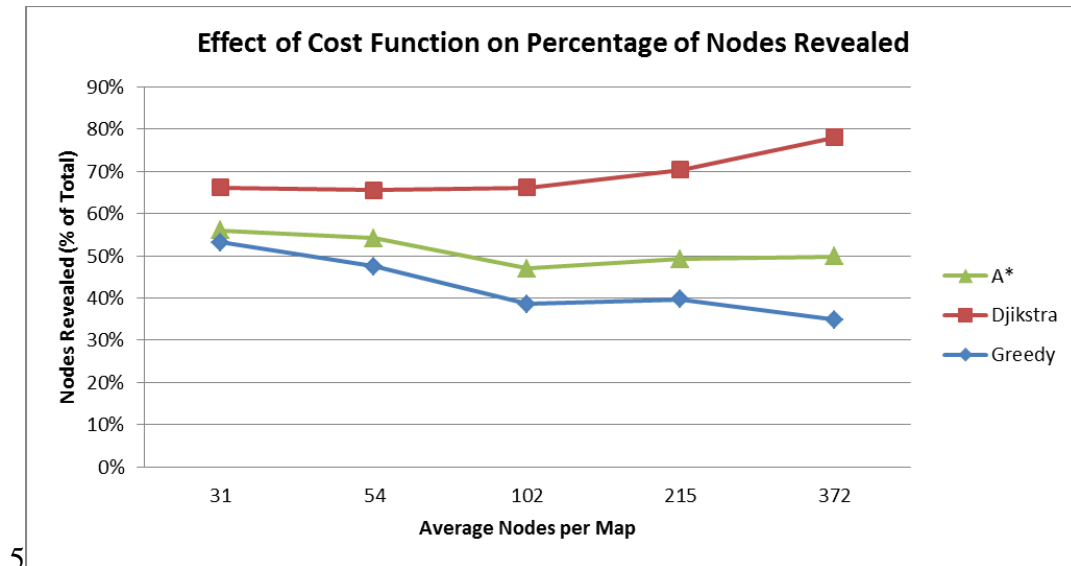


Figure 26: Effect of Cost Function on Nodes Revealed

It should be noted that these results were obtained while running scenarios where the agents were searching for a given target. Few of the maps were completely explored in the time it took to locate the target and the other two cost functions were set up to be biased towards the target. For scenarios where the goal is to explore the entire map, both of these cost functions would likely be modified to give better performance under that

circumstance. Lastly, it should also be noted that much research has been done in the area of mapping unknown environments and some of the other approaches may be better suited to this type of application (see the section on Multi-Agent Mapping, page 3). While the results shown here may provide a basis for speculation, scenarios should be designed to specifically test the efficacy of this method for map exploration and those results should be compared to those obtained using other search methods

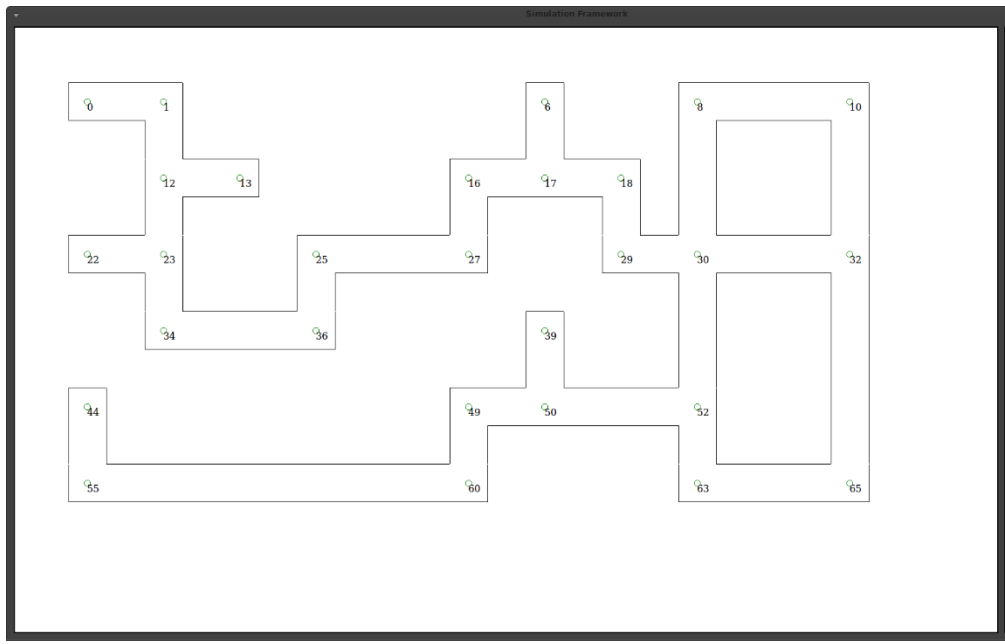
Appendix D

Maps used for Data Collection

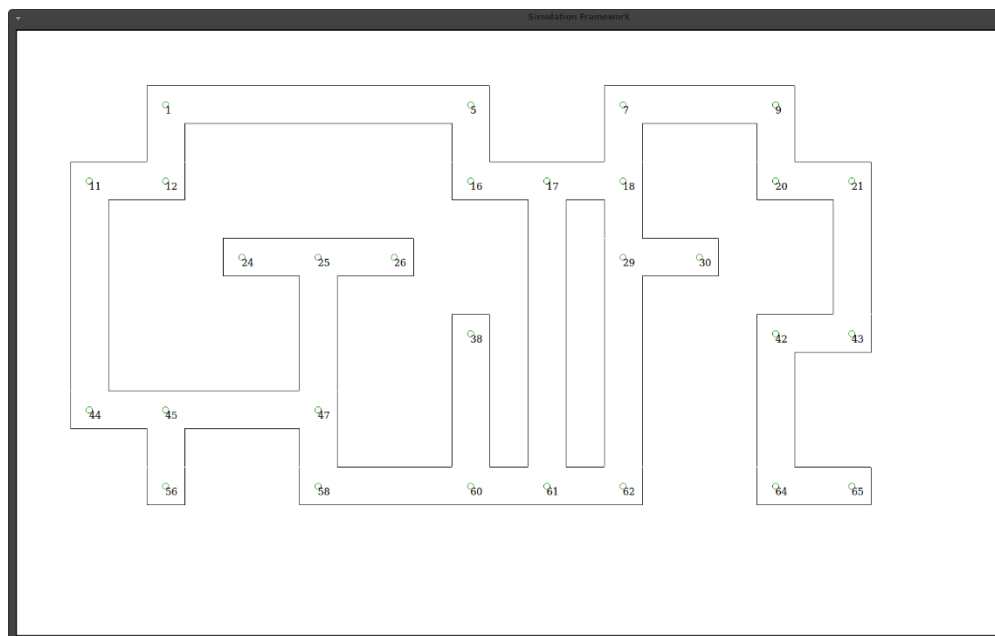
This appendix shows the maps used to collect the data summarized in the Results section (Chapter IV). There are five maps in each of five sizes for a total of twenty-five maps. Each section in this appendix gives details for each size class, beginning with statistics of that size group as a whole and following with figures and details for the individual maps.

Size Class 1 – 235 sq-ft

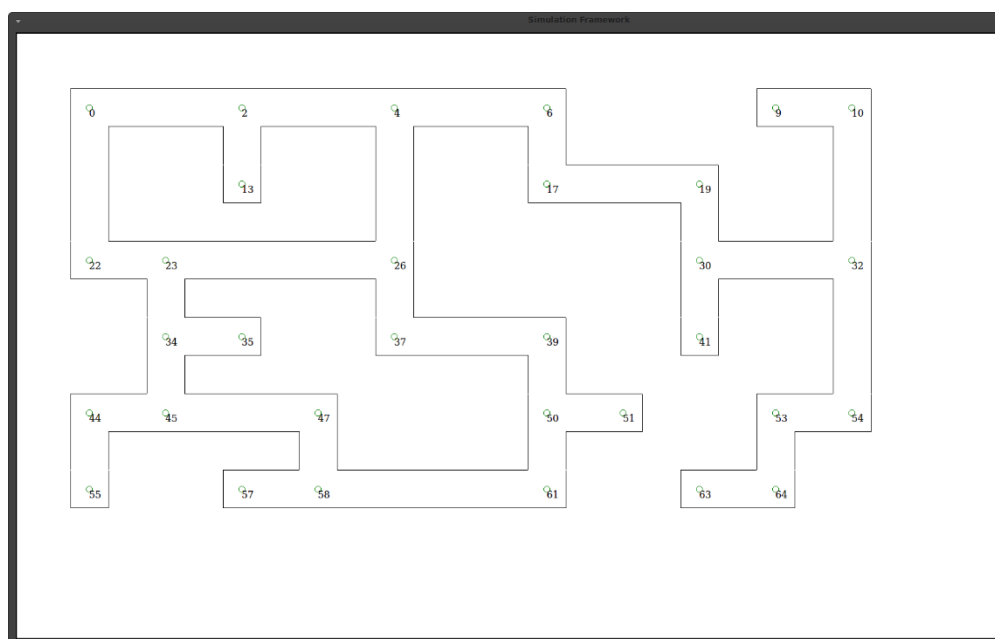
Scale (pixels/inch)	7
Area (sq-ft)	235
Width (ft)	20
Height (ft)	144
Average Nodes	31



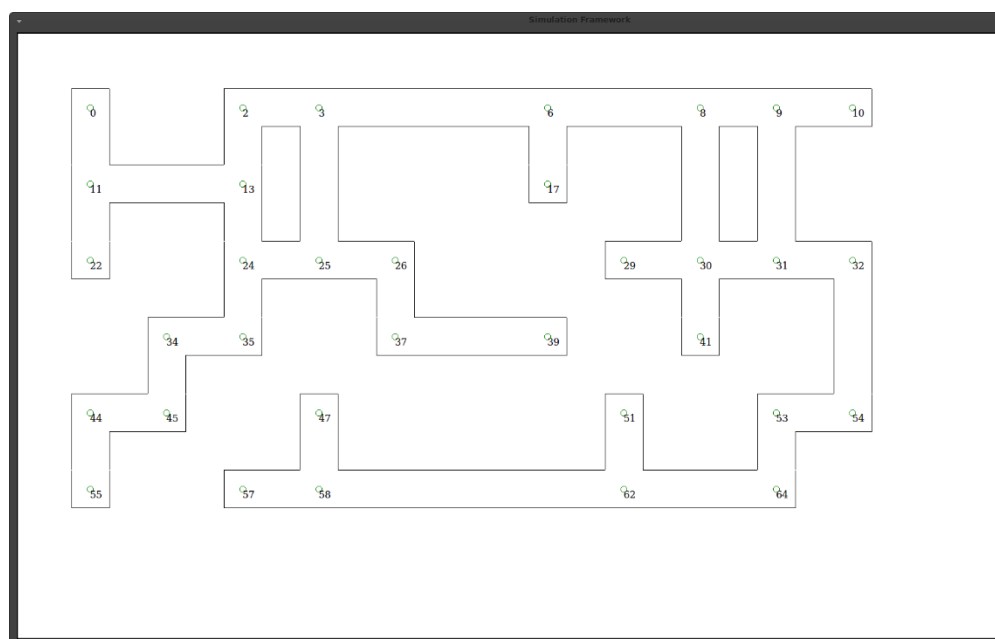
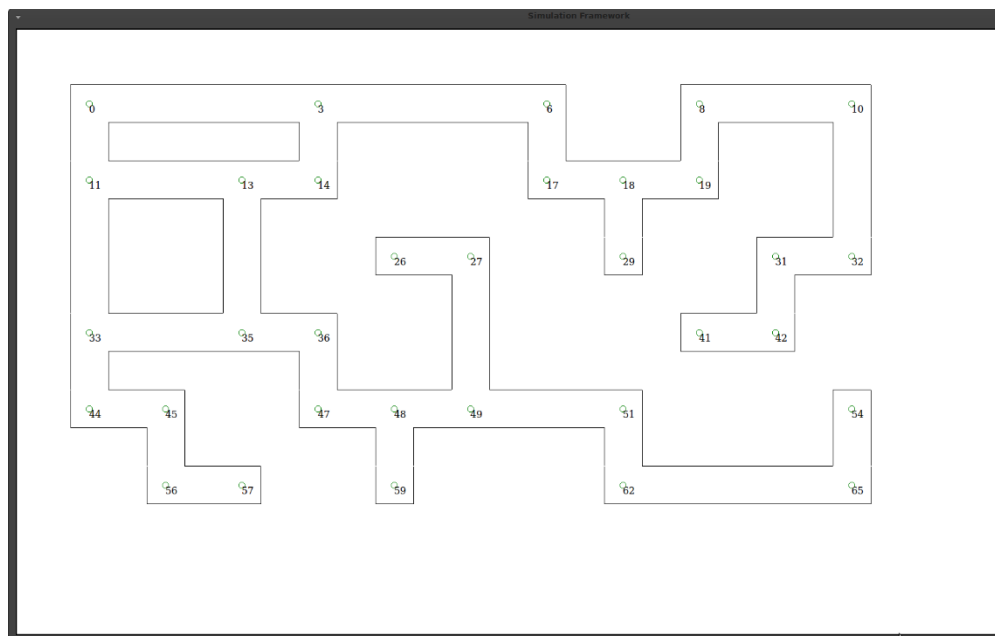
Class 1, Map 1: 28 Total Nodes



Class 1, Map 2 – 29 Total Nodes

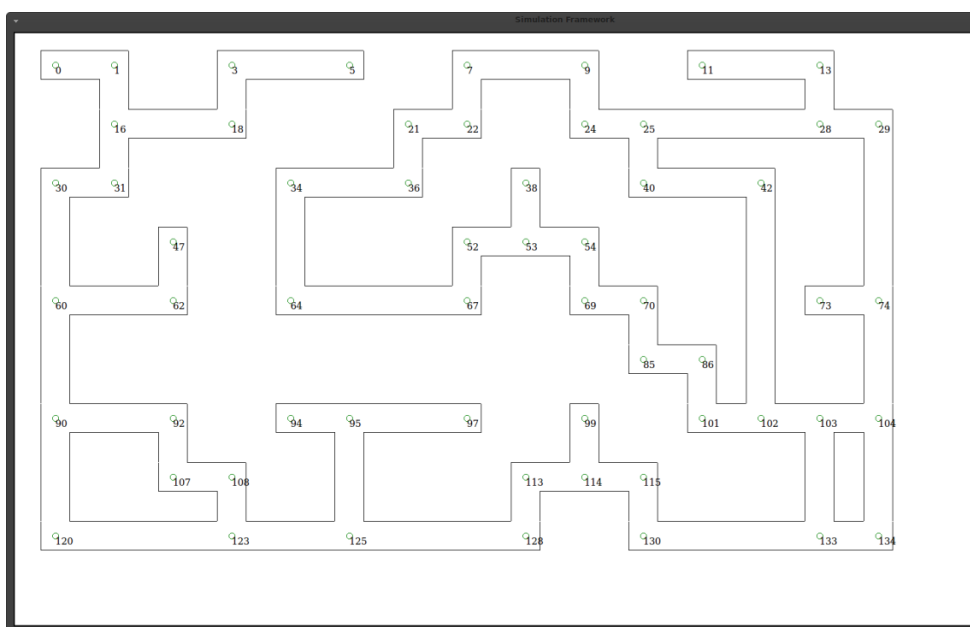


Class 1, Map3 – 32 Total Nodes

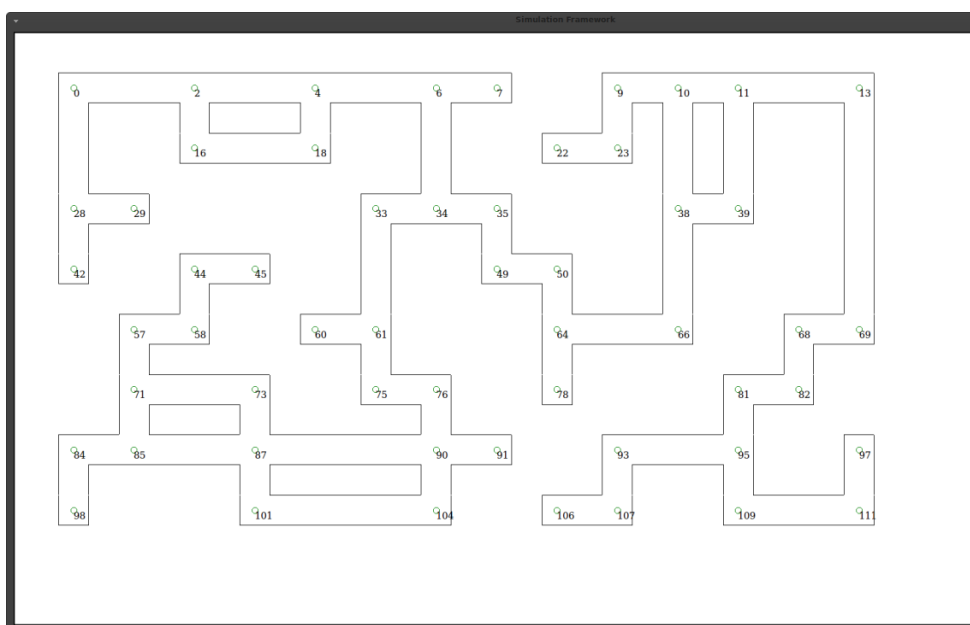


Size Class 2 – 380 sq-ft

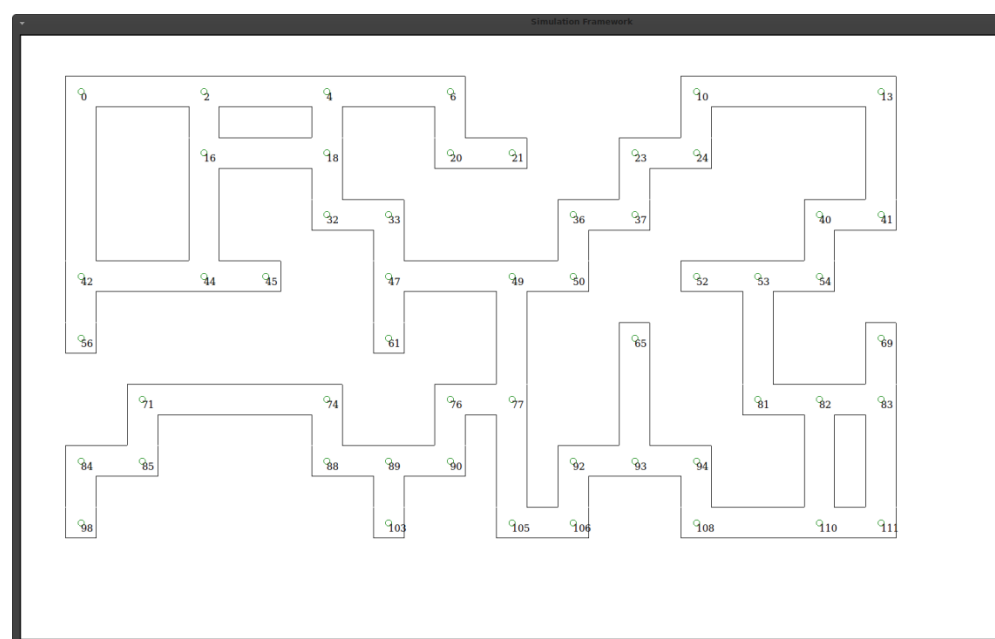
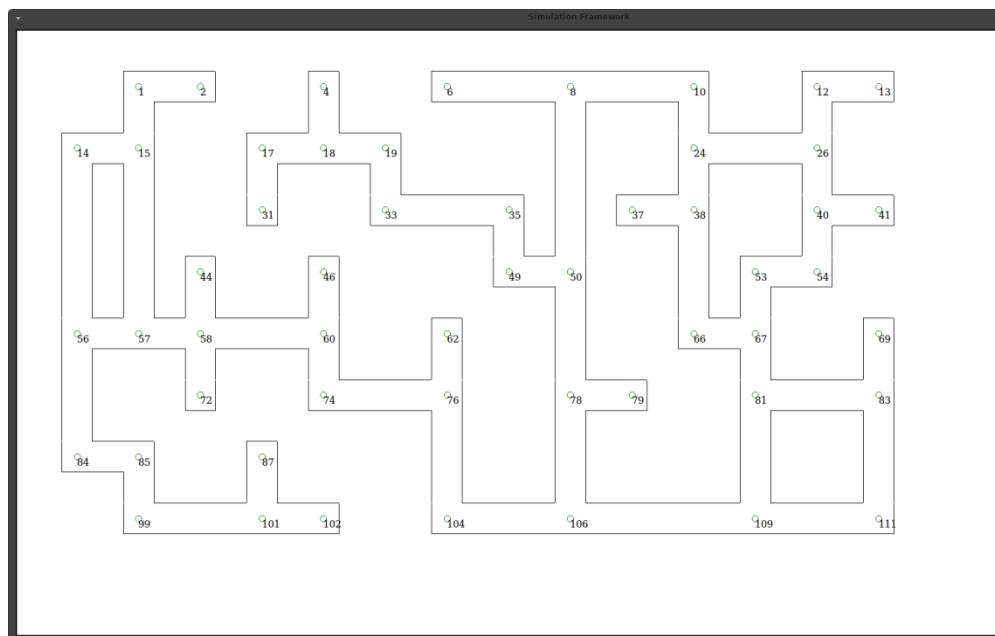
Scale (pixels/inch)	5.5
Area (sq-ft)	380
Width (ft)	25
Height (ft)	15
Average Nodes	54

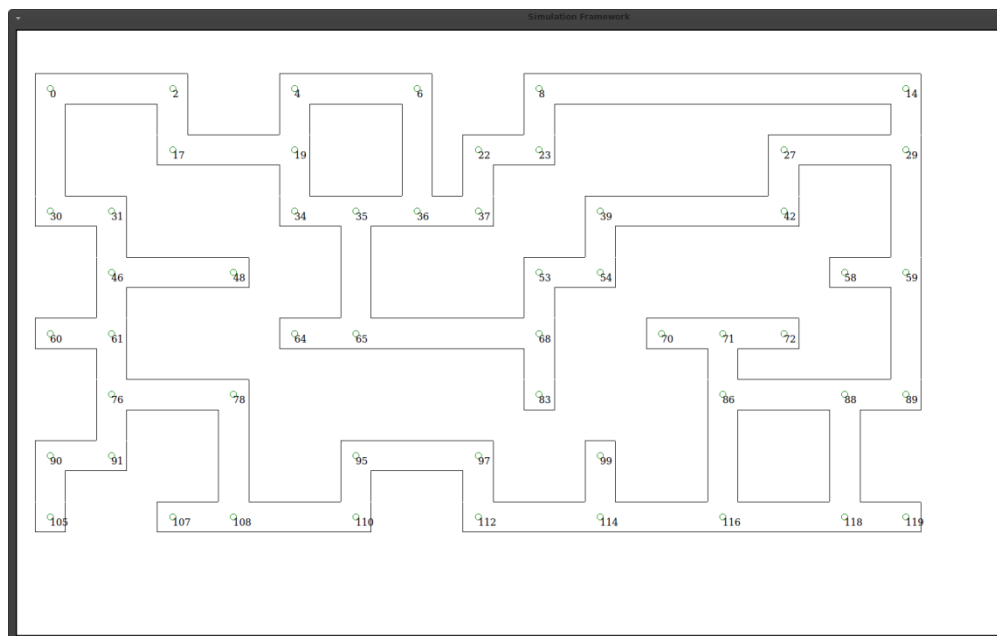


Class 2, Map 1 – 59 Total Nodes



Class 2, Map 2 – 51 Total Nodes

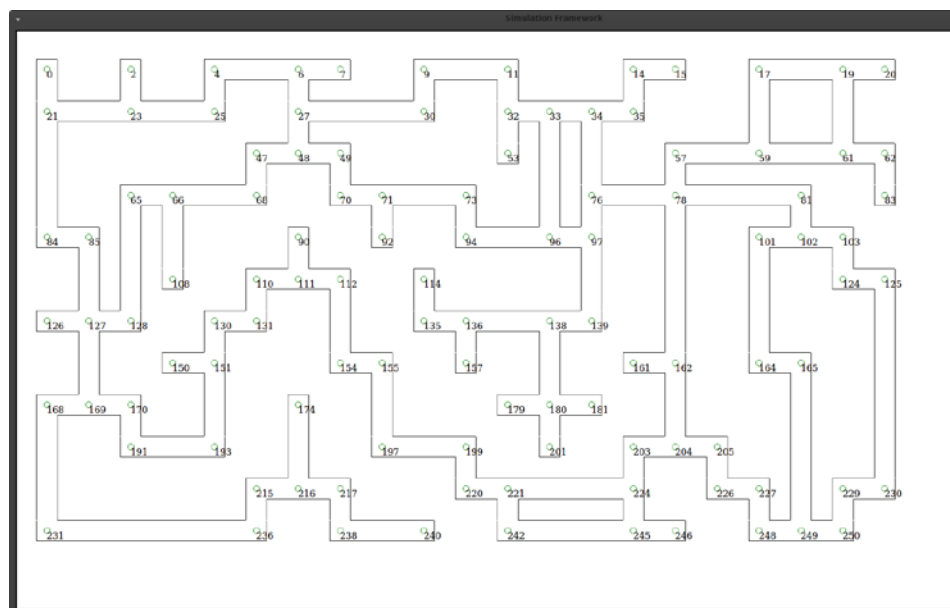




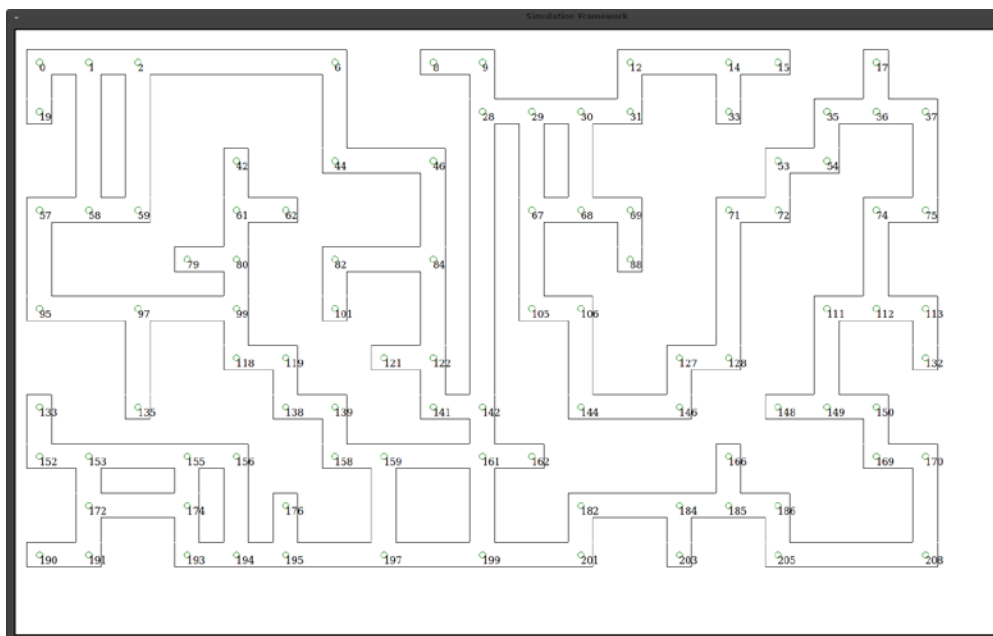
Class 2, Map 5 – 54 Total Nodes

Size Class 3 – 637 sq-ft

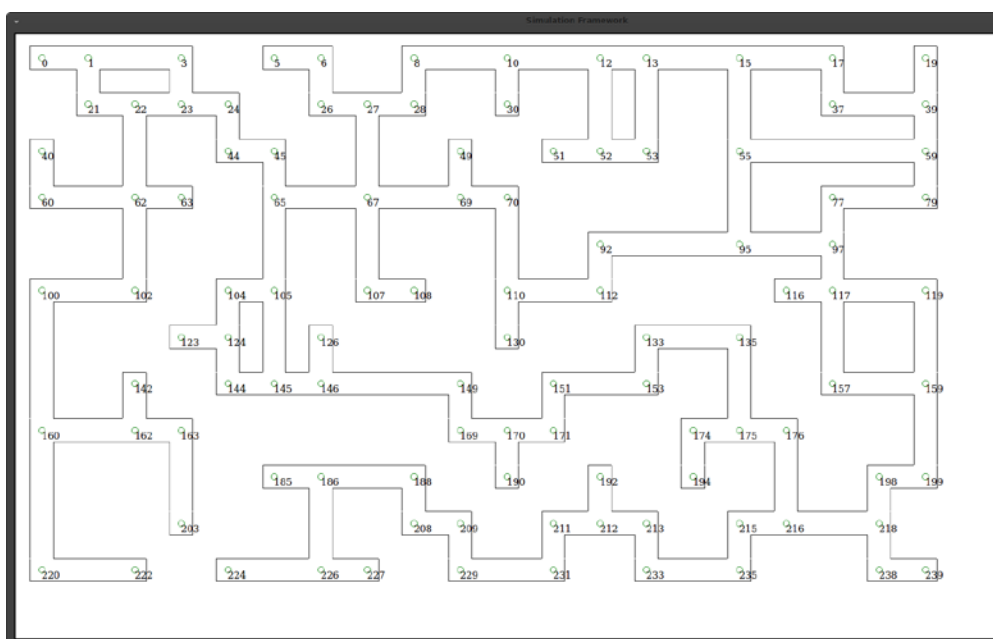
Scale (pixels/inch)	4.25
Area (sq-ft)	637
Width (ft)	32
Height (ft)	20
Average Nodes	102



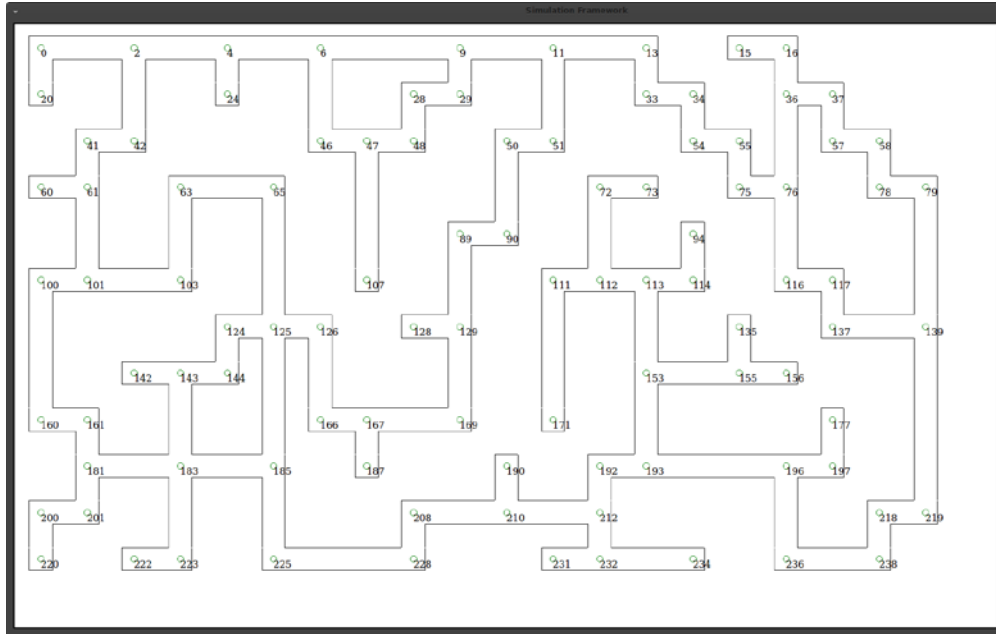
Class 3, Map 1 – 109 Total Nodes



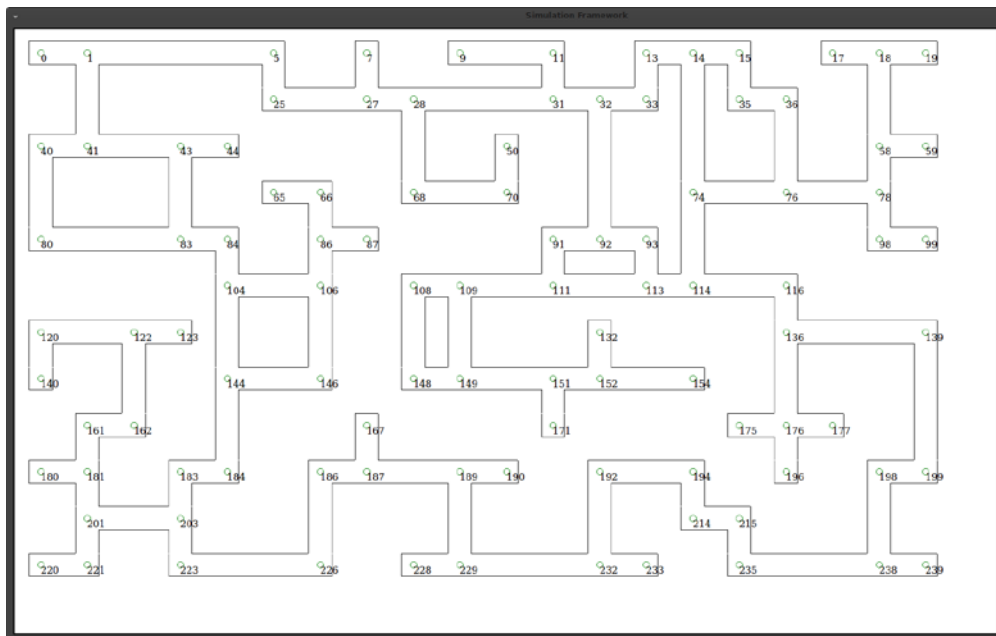
Class 3, Map 2 – 97 Total Nodes



Class 3, Map 3 – 106 Total Nodes



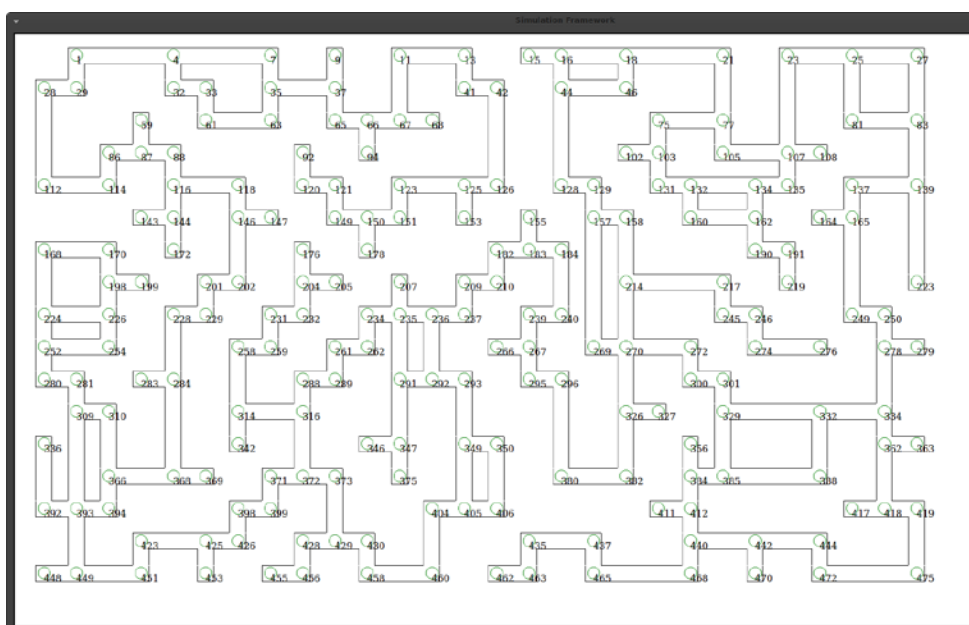
Class 3, Map 4 – 98 Total Nodes



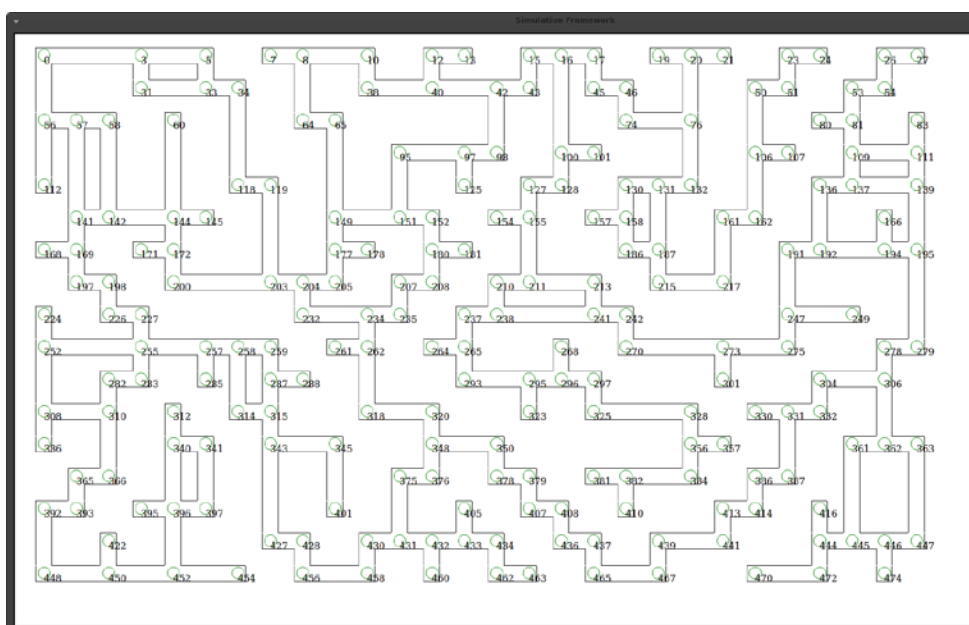
Class 3, Map 5 – 101 Total Nodes

Size Class 5 – 1278 sq-ft

Scale (pixels/inch)	3
Area (sq-ft)	1278
Width (ft)	46
Height (ft)	28
Average Nodes	215



Class 4, Map 1 – 212 Total Nodes



Class 4, Map 2 – 222 Total Nodes

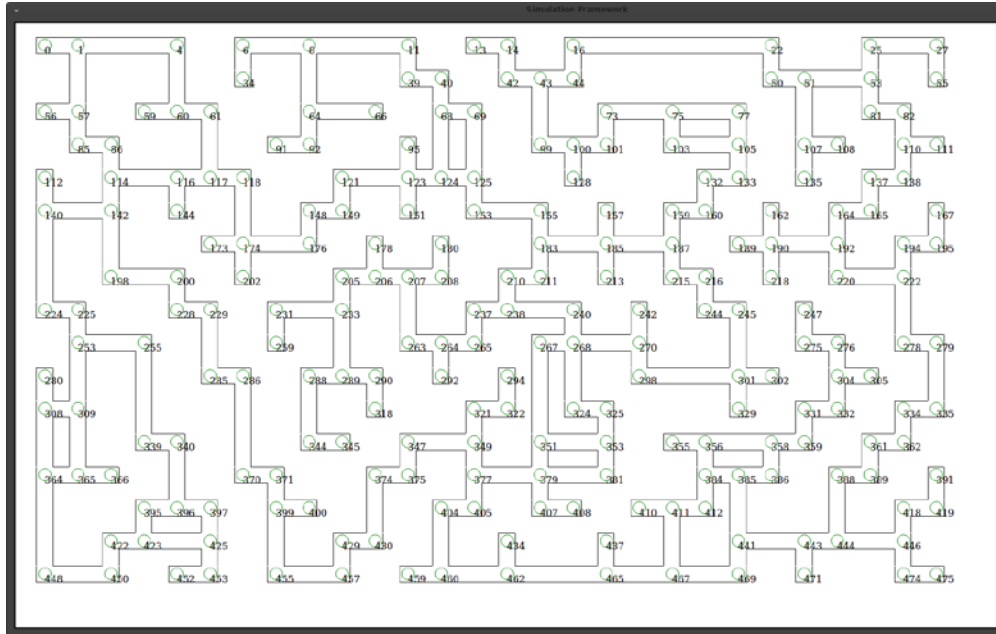
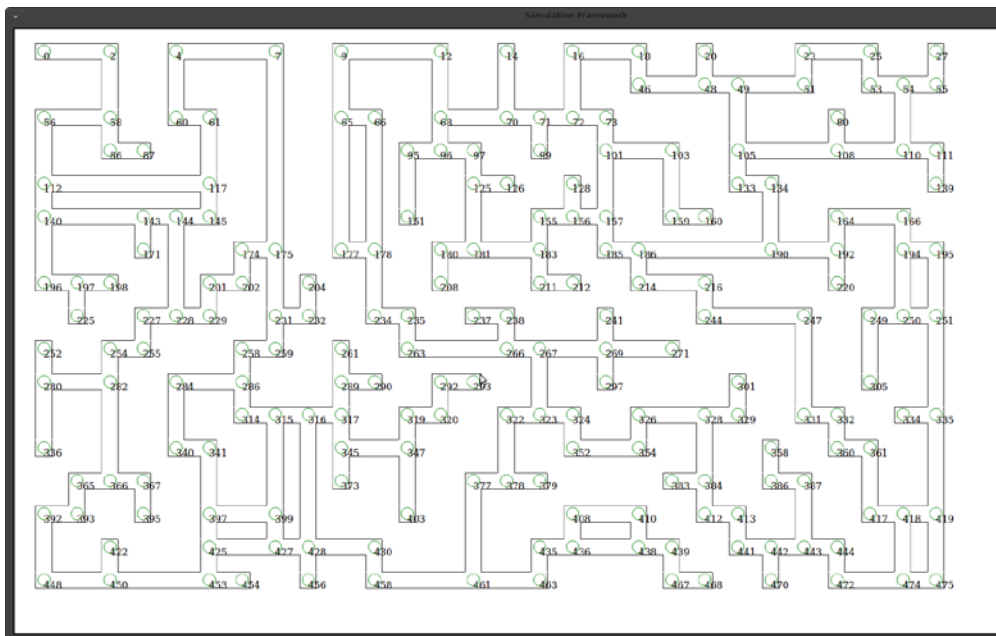
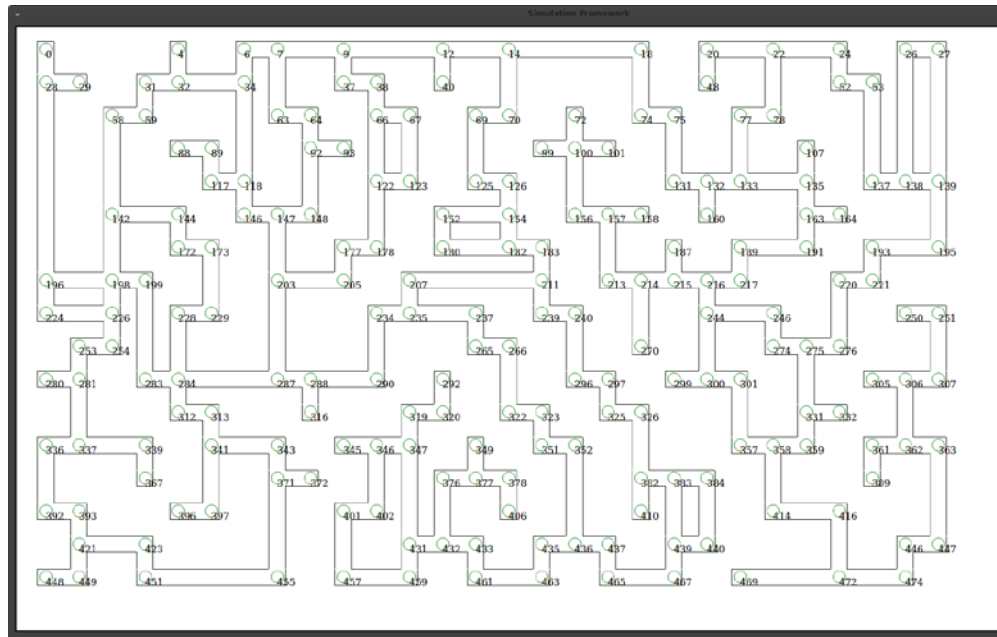


Figure 27: Class 4, Map 3 – 229 Total Nodes



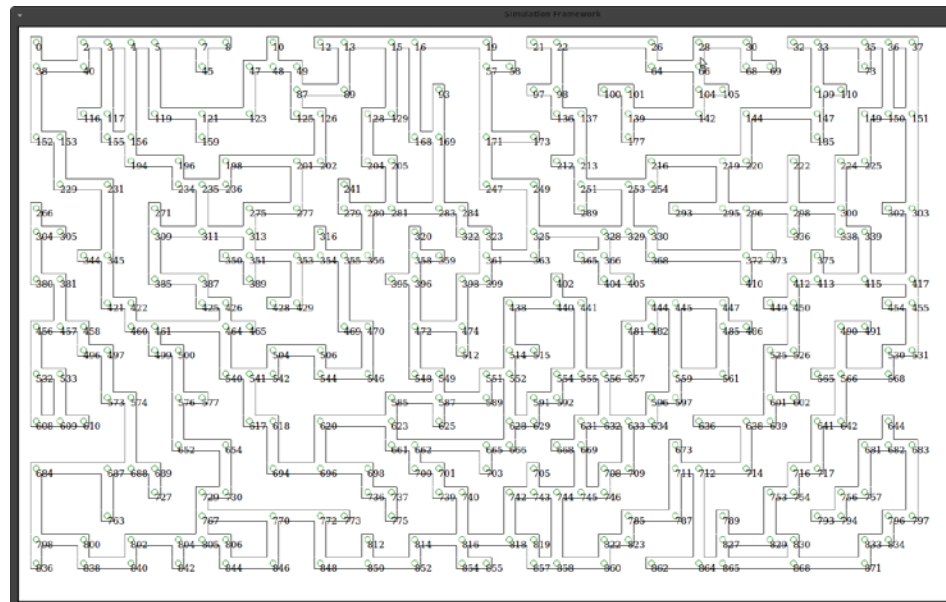
Class 4, Map 4 – 205 Total Nodes



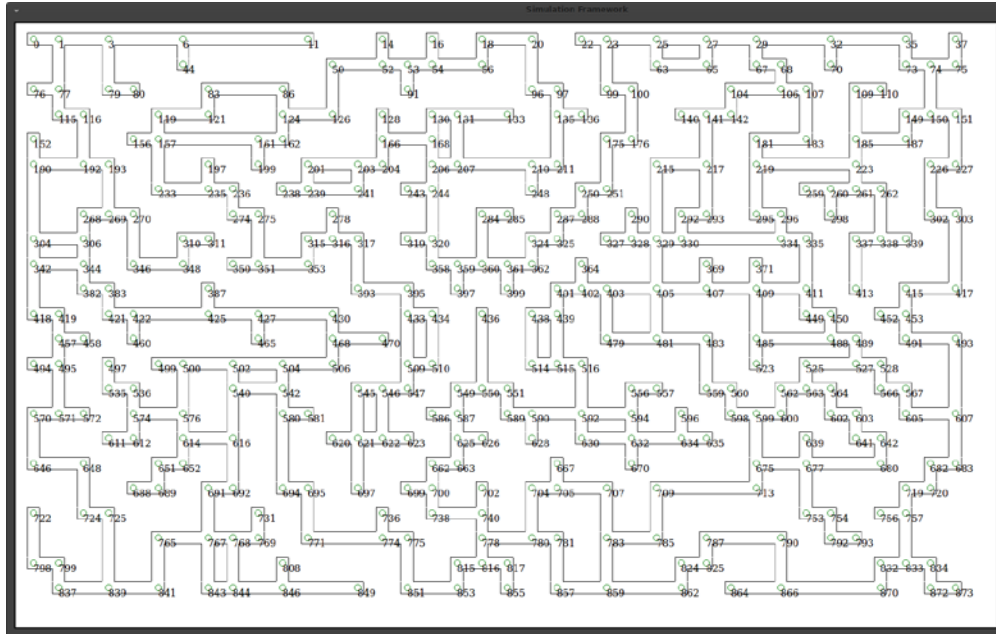
Class 4, Map 5 – 207 Total Nodes

Size Class 5 – 2272 sq-ft

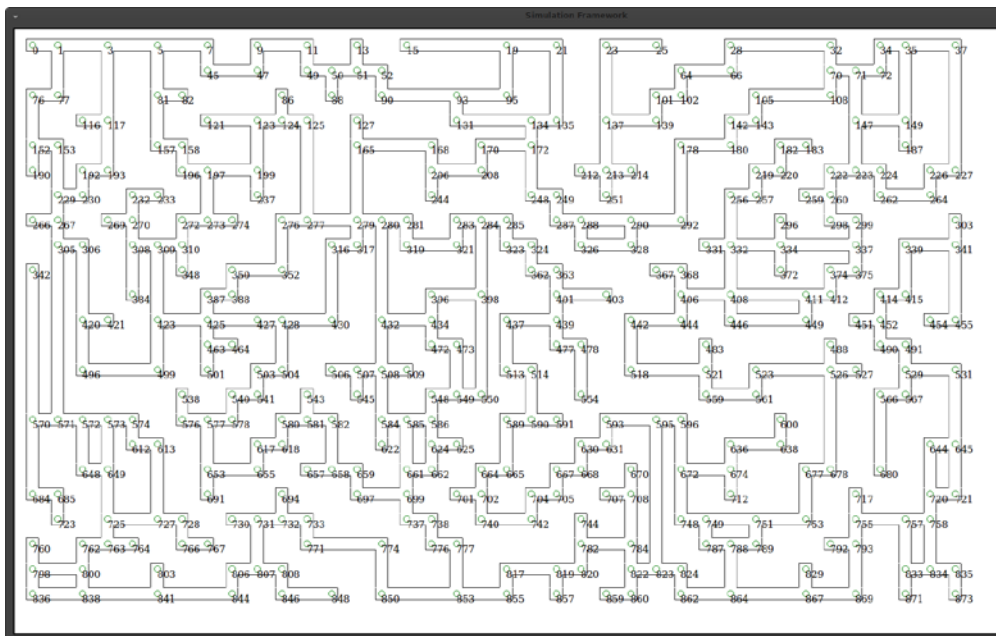
Scale (pixels/inch)	2.25
Area (sq-ft)	2272
Width (ft)	61
Height (ft)	37
Average Nodes	372



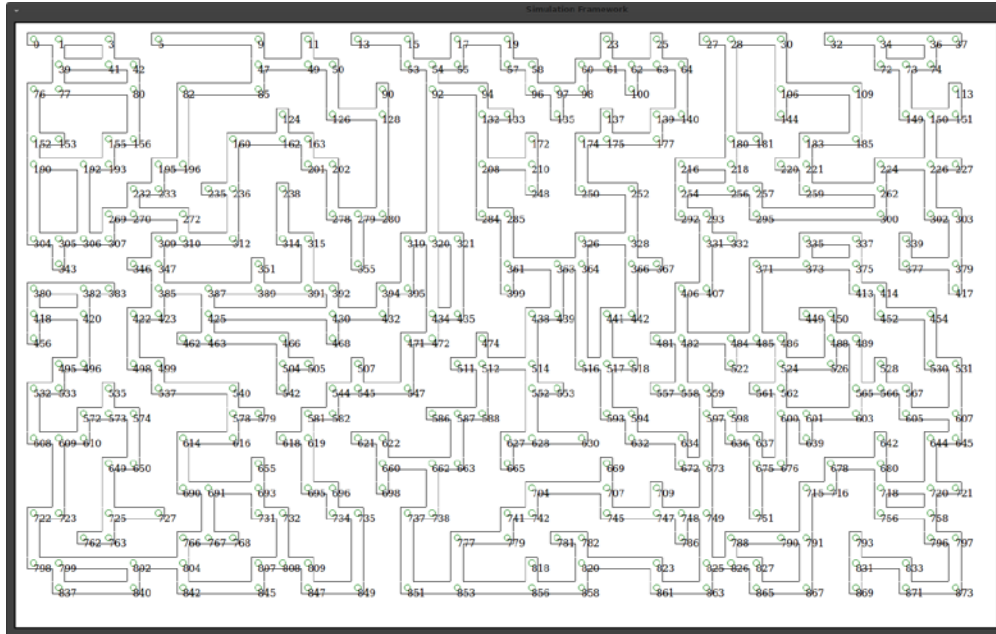
Class 5, Map 1 – 365 Total Nodes



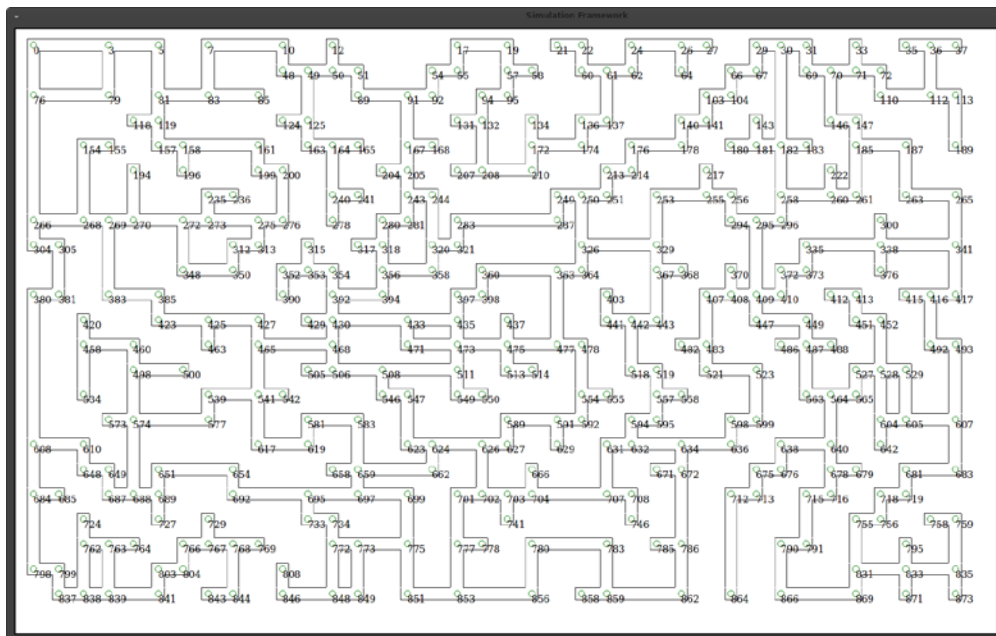
Class 5, Map 2 – 372 Total Nodes



Class 5, Map 3 – 370 Total Nodes



Class 5, Map 4 – 378 Total Nodes



Class 5, Map 5 – 373 Total Nodes