# Search and Rescue Robot Path Planning: Thesis Proposal

Andrew Hogan
*Wentworth Institute of Technology,*
*550 Huntington Avenue, Boston, MA 02115*

**Abstract**

A major goal of agent-oriented path planning in Artificial Intelligence is to produce a solution that encompasses the trade-off between runtime and computational complexity while also maintaining a certain level of solution completeness based on the problem being solved. One such area where the solution completeness is especially significant is that of search and rescue robots. Often, it is more important for the agent in this setting to complete its task in a likely high-pressure situation. As a result, this domain allows for the use of methods that may produce an increase in the overall expense of computational resources.

The process of simulating a search and rescue robot on varying environments requires a concise form of environment representation. This simulation relies on a conversion of input environments from a 2D grid space to a visibility graph to identify obstacles (static and dynamic) in the environment. Visibility graph representation allows for the construction of edges between the corners of obstacles in the environment for the agent to follow to reach the desired (goal) location.

I will implement a mix of static and dynamic environments using building floor plans for the simulation of a search and rescue robot in a progression of environment complexity. Environments become more complex through the addition of obstacles in both quantity as well as dynamic emergence after a certain time of environment execution. Several variations of commonly used path planning algorithms, such as A* with varying heuristics, and weighted A*, will be used to construct short paths for the search and rescue robot in each environment type. The overall results will be a discussion of the completeness, runtime, and cost complexity of each variation of environment as well as the associated solver.

## I Introduction

An overlap between the fields of Machine Learning and Robotics, the domain of path planning continues to see an emergence in its problem solving ability. Traditional path planning, in both physical settings with an actual robot as well as in simulations, is concerned with constructing a set of step-by-step movement instructions for the agent to follow. Generally, these problems are designed to find the optimal path for the agent to traverse an environment and may or may not include additional tasks around the environment as well as the presence of obstacles. Given the need to produce step-by-step directional movements for the robot, environments in this problem domain are normally represented in a 2D grid space. While this breaks out the environment into equal spaced blocks to aid in the creation of sequential directions for the agent, it presents a number of issues for more complicated applications of path planning. In addition to the storage requirements of the exponentially growing state space for a 2D grid, the notion of obstacles that are seen in the uncontrolled environments of the real world cannot always be easily represented by a 2D grid. The dynamic nature of obstacles in real world settings present a major shortcoming of 2D grid-based path planning and a motivating factor for enhancing advanced applications of path planning.

One such advanced application that is the main focus of my work is search and rescue robots. A search and rescue robot is an agent tasked with assessing the scene of an ongoing or recently occurring natural disaster or tragedy. They are normally responsible for executing the scan of an environment for survivors or important equipment when needed in a search role, and can also be tasked with delivering resources to trapped survivors until safe extraction can occur in a rescue mission. While these agents can be presented with similar environments to traditional path planning agents, the manner in which they learn and traverse them are significantly different. As mentioned, obstacles faced by a search and rescue robot in a real world environment are not guaranteed to be static nor predictable in their emergence as with simple path planning. Certain dynamic obstacles, such as the wall of a building falling in a disaster scene, create an entirely new issue for agents in a search and rescue setting. As a result, the 2D grid space of traditional path planning does not easily translate into the search and rescue robot idea, and a new system for the representation of these environments is needed.

A potential form of representation for environments in the search and rescue domain and the chosen one for this implementation is the visibility graph. This approach builds a graph of the environment using corners of obstacles as vertices to later build a path comprised of visible edges between these corners. Not only does this method vastly reduce the state space needed in order to fully represent the environment, but the notion of dynamic obstacles can be more easily introduced with a visibility graph representation. The specifics on building an optimal path from a visibility graph will be discussed in greater detail in the Methods section, but the use of this data structure allows for enhanced possibilities in environment simulations for a search and rescue robot. Since the visibility graph focuses on obstacles and other blocking points in an environment, such as walls, less steps are required overall to reach the agent goal location while allowing for the use of more complicated environments. The following image provides a visual for the construction of a visibility graph:

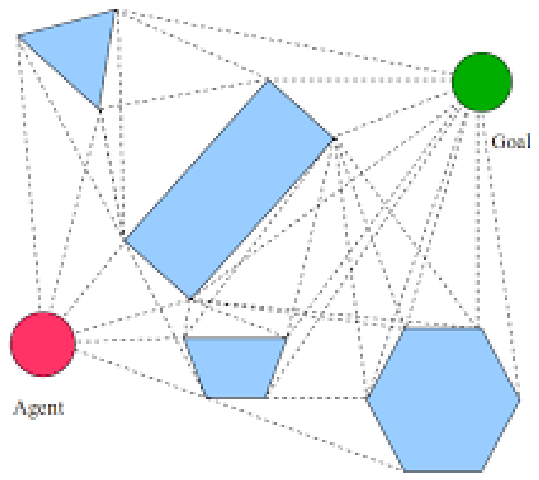

Figure 1: Visibility Graph Representation

## II Objectives

Throughout the implementation of path planning for a search and rescue robot in this project, there will be two main objectives. The first objective focuses on the establishment of varying environment and algorithmic complexity to test the validity of the path planning in multiple situations, while the second objective is concerned with the presentation of results.

For the environment complexity objective, I will establish 4-5 environment types containing varying degrees of obstacle complexity and dynamic nature. The specific parameters of each environment type will be explained in detail in the Methods section, but the main purpose is to create a basis for comparing agent performance on environments with varying degrees of complexity. For example, introducing dynamic obstacles into the domain (such as a wall falling) has the potential to alter the manner in which the search and rescue robot interprets the environment and constructs the optimal path compared to an environment with static obstacles. As a result, the implementation of environment parsing will account for the potential for dynamic obstacles to enhance the analysis between environment complexity. While the varying forms of environment will be the primary mode of performance comparison of the search and rescue simulation, algorithmic complexity will also be used for additional analysis. The algorithm used by the agent to solve for the optimal path in a given search and rescue environment has the potential to impact the admissibility, runtime, and cost complexity of the solution. As a result, varying the algorithmic complexity is an additional form of analysis that is relevant for the main objective of this project.

Once all of the environment types have been established and solved for the optimal path, an enhancement to the results of comparing agent performance across environment types and algorithms is a simulation of the robot on the environments. The purpose of the simulation is both to visualize the execution of the agent towards the goal state in a given environment while also representing the impact dynamic obstacles have on path solutions in the more complicated environments. As dynamic obstacles are introduced, the simulation will provide a visualization of how the agent path alters to adapt to the change in environment while remaining in an optimal direction towards the goal. Overall, the simulation will serve to validate the results themselves by providing a visual of the optimal path of each environment while also enhancing the discussion of the impact of dynamic obstacles through the adaptations made by the agent in the simulation. It is important to note that the main objective for the results is to conduct a discussion of the comparison between environment types along with algorithmic choices for the search and rescue robot domain, while the simulation is an enhancement of the results and a visual resource for interested readers.

## III Methods

The major components of the methods that will aid in the development of this project will be outlined in the following three sections: Environment Complexity Progression, Potential Solver Algorithms, and Environment Representation. Each section handles a specific part of how the search and rescue robot will be implemented as well as for the analysis of results.

### III.1 Environment Complexity Progression

As alluded to in the discussion of the main objectives for the project, a progression of search and rescue environments will be established by altering obstacles in both quantity and dynamic nature in order to produce an analysis between environment complexity. A description of each environment type can be outlined as follows:

1. **A "Clean" Environment**

   This environment will have no external obstacles aside from the constraints of the environment itself (walls, doors, etc) for the agent to avoid. As the starting point for increasing environment complexity, this first environment variant will not be extensively used in the advanced discussion of environment comparison in the results. Rather, it will serve as a "proof of concept" of sorts in order to validate that the solver algorithms are implemented correctly and to quickly observe the different paths created when the algorithm type is varied.

2. **Static Obstacles**

   In this environment, there will be multiple static obstacles present at the start of program execution. This will require the agent to construct its optimal path with these obstacles in mind as locations in the environment that are potentially no longer reachable. Due to the static nature of the obstacles, the representation of this environment in the visibility graph will allow for a one time process of encoding the obstacles in the graph which will not change throughout the program execution. Despite this, the notion of which obstacles provide enough surrounding open space to be included in the visibility graph presents an extra layer of complexity for solving this environment compared to the clean environment.

3. **One Dynamic Obstacle: Time Known**

   The third environment type is the first to introduce the notion of dynamic obstacles. In this scenario, the agent will be faced with the emergence of one dynamic obstacle (wall falling, moving debris/other entity, etc) at a known location and time interval after the start of program execution. Described further in the Environment Representation subsection, the inclusion of dynamic obstacles changes the manner in which obstacles as well as the visibility graph itself need to be represented. Since there is now a notion of time for the emergence of a new obstacle, this must be accounted for while the agent conducts its path execution and potentially encounters the dynamic obstacle. It is important to note that elements from the second environment type can also be present in this type. In other words, the only condition guaranteed in this environment type is that there will only be one dynamic obstacle. There could, however, also be multiple static obstacles present at the start of environment execution as well.

4. **Multiple Dynamic Obstacles: Times Known**

   Expanding on the previous environment, the fourth environment type presents multiple dynamic obstacles to the agent throughout the program execution. As before, the times of emergence of each of these obstacles are known at the onset and static obstacles can still be present in the environment as well. The purpose of this environment type is to continue to increase the complexity in the overall path planning of the agent by increasing the likelihood that the agent will be forced to adapt the optimal path on the fly. In the previous environment with only one dynamic obstacle, it is easier for the agent to plan its path accordingly before being faced with the obstacle or even potentially avoid it altogether since one obstacle likely will not overly impact a large environment space. With multiple dynamic obstacles at varying locations, however, the likelihood for potential collisions is increased, putting an increased level of planning complexity on the agent.

5. **Multiple Dynamic Obstacles: Times Unknown**

   The final and most complicated environment type, the fifth environment tasks the agent with responding to the most unpredictability across all environment variants. In this scenario, the agent is faced with a mix of multiple static and dynamic obstacles as with the previous environment, but the times of dynamic obstacle emergence are no longer known at the onset of program execution. As a result, the visibility graph cannot easily encode obstacles during the parsing of the environment by simply denoting the times during which a location are/are not available based on obstacle emergence. Withholding the times of dynamic obstacle emergence requires the development of a predefined collision detection system that the agent can use to quickly respond to obstacles. As discussed, the simulation component will provide a clear visual aid for how the agent adapts its path to dynamic obstacles of this nature, as there will be limited time to avoid collisions with obstacles in this environment type. After the successful completion of this environment, the progression of complexity between the five environments outlined here will provide the tools for an interesting discussion on the performance of the agent across each environment variant.

## III.2 Potential Solver Algorithms

For each of the environment types, the algorithm used to find the optimal path will also be varied for an additional form of comparison analysis. The main path solving algorithm used for this implementation will be the A* algorithm. In addition to being an efficient choice for a variety of path planning problems, the heuristic function for sorting nodes in A* can be easily altered in order to vary the algorithm execution and achieve that form of comparison for this project. The following figure provides a visual of the general pseudocode for the A* algorithm:

```
Open ← priority queue containing initial state
while true do
    if Open is empty then
        return failure
    Node ← Open.Pop()
    if Node is goal then
        return Node (or path to Node)
    else
        Children ← Expand(Node)
        Add Children to Open, sorted on $f(n)$
```

Figure 2: A* Algorithm Pseudocode

The last line of the algorithm pseudocode denotes that children node after a call to expand are sorted based on the function f(n). For the purposes of A*, f(n) is calculated using the following equation:

$$f(n) = g(n) + h(n) \tag{1}$$

Where g(n) is the path cost from the start state to the current state, and h(n) is a *heuristic* estimate from current state to the goal. As discussed, the heuristic estimate is the component of the algorithm that can be varied in order to produce different results for how A* interprets and builds the optimal path for an environment. Since h(n) is simply an estimate of how far the agent is from the goal state based on its current state (node n), there are a variety of elements in the environment that can aid in providing an option for this estimate. Some example heuristic estimates for any of the environment types could include:

- Number of obstacles in environment
- Number of dynamic obstacles yet to emerge (0 in static environments)
- Number of currently visible and meaningful obstacle corners in the agent field of vision

Through the use of these example heuristics as well as others for the implementation of A* across each environment type, analysis between the heuristics can be conducted to determine which ones produce the most optimal solutions. In general, heuristic functions that never overestimate the cost to the goal produce the most admissible path solutions. In addition, a variation of the A* algorithm called weighted A* will also be experimented with to increase the set of algorithms for solution comparison. The weighted A* algorithm multiplies the heuristic function h(n) by a constant factor to increase the runtime of the original A* algorithm. As a result, the impact of the chosen heuristic is heightened in the implementation of weighted A*. Nodes in weighted A* are sorted with a different f(n) function, denoted below:

$$f(n) = g(n) + w * h(n) \tag{2}$$

Where g(n) is still the path cost from start state to the current state and h(n) is still the heuristic estimate, with the inclusion of multiplying h(n) by the constant factor for the weight w. As a result, using the three potential heuristic functions mentioned earlier in this section on both A* and weighted A* will produce six unique solutions for solving a given environment, providing additional algorithm choices for the discussion of results.

## III.3 Environment Representation

To further expand on the use of a visibility graph to represent the components of each environment type, this section will serve as a description of the graph itself, the notion of an obstacle list, the process of checking neighbors in the graph when expanding a node in A*, and the later inclusion of floor plans as test environments.

The nodes of the visibility graph will include existing features of a given environment, such as walls and doors, as well as any static obstacles that are already present in the environment at the time of agent execution. This representation will allow the agent to build potential edges between each node of the graph to reach the goal location. When dynamic obstacles are included, however, the visibility graph cannot be used on its own to fully represent the environment. To handle dynamic obstacles, an additional data structure is needed to contain the list of dynamic obstacle locations as well as their associated time of occurrence (excluding the final environment type). With this organization, the agent has the ability to check both the visibility graph and the additional obstacle list when expanding a node to determine valid new states. In general, when determining children nodes in a call to expand in A*, the agent will check all neighboring nodes of the visibility graph to determine if any paths will be blocked. This will include cross-checking the obstacle list to ensure that neighboring nodes are not included in the location of a dynamic obstacle, and if one is, that the current time is before the time of occurrence of that obstacle. The inclusion of an additional data structure for dynamic obstacles will allow for the implementation of all environment types, with the last environment also requiring a collision detection system since the dynamic obstacles are known but the time of occurrence is not.

The varying types of environment complexity have been described in detail throughout this proposal, but the source of how these environments will be created has not. For initial testing, simple environments will be user-engineered with predefined symbols for open spaces, blocked spaces, and goal locations in a grid layout held in text files. Ultimately, however, the environments tested on will represent real floors/rooms from building floor plans. A separate preprocessing step will be established to parse the floor plans into a readable format for the visibility graph, and static/dynamic obstacles will then be added onto these environments to build the full environment progression system. Not only will this enhance the manner in which environments become more complex, it will also produce more meaningful results by using locations that actually exist.

## IV Research Questions

Since this project is still in its early stages of implementation, there are a number of questions that have guided the research into the search and rescue robot domain so far and will continue to be relevant throughout the analysis of the results. For each question area, there may be an answer that has already been determined in the process of developing this proposal, but others cannot be effectively answered until implementation progresses further or even until the results are produced.

The first area of questioning is to determine how a visibility graph is better suited for the search and rescue robot domain compared to a 2D grid representation. Both approaches for environment representation have been discussed in detail throughout this proposal, with associated support for the use of a visibility graph in this implementation due to its ability to represent varying types of obstacles and complicated, real-world environments. The use of a visibility graph will be addressed again later in the full document during the discussion of comparing the runtime and admissibility of solutions from varying environment types.

Directly targeted towards the main objective for the results of this project, another important question to pose at this time is how can approaches to path planning problems simultaneously balance the trade off between runtime and cost complexity while also maintaining solution completeness? As previously mentioned, it is critical for a search and rescue robot to produce a solution that is complete, particularly given the nature of its use. In other words, a search and rescue robot is useless and potentially even harmful if it cannot correctly navigate to the location it needs to go. On the other hand, a major component of the results for this project is to compare the runtime and cost complexity of the solutions from each environment type.

As a result, there needs to be a discussion of how path planning problems balance the trade off between solution completeness and cost, and how this relates to the implementation of this project.

Introduced in the final subsection of the Methods section, the eventual source of environment testing will be the use of real building floor plans. A major challenge, therefore, becomes the process of determining how floor plans can be converted into visibility graph format for the solver algorithms to use. There are a number of potential ideas for approaching this problem, with some likely more computationally expensive than others. One idea is to first convert the floor plan into a 2D gid, with equidistant "blocks" of floor representing each space of the grid. Then, the 2D grid can be converted into the visibility graph using the same approach as the one implemented for the initial user-engineered environments. While this is a valid solution, it is computationally expensive due to the two step process involved. Converting the floor plan to a 2D grid in itself would be an expensive task, with the conversion to a visibility graph still required after the initial conversion. Another idea is to convert the floor plan directly into the visibility graph in one preprocessing step. This approach may even be easier to implement overall, particularly for static obstacles that are already clearly visible on floor plans such as walls and doors. In addition, this method would also force the establishment of a format for representing static obstacles from the floor plan which could in turn yield a similar solution suitable for representing dynamic obstacles. As mentioned, the creation of an obstacle list for dynamic obstacles in addition to the visibility graph will provide the most efficient procedure for expanding nodes in A*, and the direct conversion from floor plan to visibility graph could aid in this part of the implementation.

## V Tools

There are a few tools that can be used to aid in the development of the path solving itself as well as for the simulation. Beginning with the simulation, specific software will be needed to host the simulation of each environment external to the path planning itself. One potential option for this is the SimPy library in Python.

Since the software enables the simulation of real-life events with actively moving components, it could provide an efficient way to build the simulation of each environment type.

Another Python library that has proved useful in the early stages of implementation is Pyvisgraph. This library builds a visibility graph given an input environment by detecting the coordinates of the corners of each obstacle in the environment. It can also be used to find the shortest path between two points using the stored obstacle information in the graph. While Pyvisgraph could be useful to validate path solutions from A*, it has proven to produce non-feasible solutions on 2D grid environments in early testing. As a result, it will likely be used primarily for the beginning stages of environment representation since it accurately finds the static obstacles in an environment and creates a vertex for each obstacle corner.

## VI Timeline of Deliverables

The following section is a tentative timeline for the completion of project components:

### November 2022

- Complete implementation of solving static environments on user-engineered environments

### December 2022

- Complete implementation of solving dynamic environments on user-engineered environments
- Begin the process of pulling real floor plans for environments

### January 2023

- Finalize implementation of solving both static and dynamic environments with floor plan environments
- Start writing results/discussion sections of Thesis

### February 2023

- Begin simulation of solving each environment type
- Continue to conduct tests on varying environment complexities and algorithms to further develop results section

### March 2023

- Finalize and test simulation output
- Wrap up Thesis document and prepare Thesis Defense

### April 2023

- Defend Thesis

## VII Conclusions

As the implementation for this project continues, major emphasis will be placed on the manner in which dynamic obstacles are represented along with pulling building floor plans for later environments. It will be significant to determine which floor plans to use and the accessibility of them while attempting the conversion into a visibility graph for the path planning of this search and rescue robot simulation. In addition, each stage of project implementation will have a notion of indicating how paths produced by each environment type can be validated as admissible. If this is discovered to not be true for one of the five environment types, it will cause the discussion of results comparing each environment to lose merit. As a result, this will be an important validation step each time a new environment is completed to ensure that the discussion of results is supported throughout the implementation.