Optimization Methods for Mutli-Agent Path Planning in Simulated Environments

James Wedum and Michael McCourt

*Abstract*— Teams of autonomous robotic agents have numerous applications for mapping spaces that are dangerous or inaccessible to humans. Due to the operating environment of the agents, the likelihood of damage or loss of equipment is high. As such, it is necessary for platforms to be inexpensive, fast, and durable. This paper presents several methods for increasing the efficiency of agents performing depth-first-search path planning in a simulated environment when presented with the problem of mapping a wildfire. Comparisons are drawn between existing implementations of the simulation environment and specific changes made to increase the speed using MATLAB. Approaches to optimization synthesize information available in an undergraduate computer science program and research on wildfire spread, biological agent path planning, belief consensus, and machine learning. The code base for the simulation can be accessed at https://github.com/EpiqPhale/FireMapping

# Introduction

Multi-agent autonomous robotic teams have a variety of potential applications in a wide range of fields. Advances in technology have both increased the availability of platforms capable of realizing these devices, as well as reduced their cost. However, even with technological advances, autonomous robotic agents are still constrained by their physical hardware. As such, often a balance must be struck between speed, efficiency, and accuracy in determining a course of action for an agent in its state space. This research is focused on both reducing the computational complexity of these systems as well as exploring methods to increase their accuracy in an uncertain environment. To these ends, it is necessary to take a multidisciplinary approach to fuse information on algorithms and computational efficiency [1][2], artificial intelligence [3], biological agent decision making processes [4], and machine learning [5].

Autonomous multi-agent robotic teams can be used to map spaces that are dangerous or physically prohibitive for direct human observation, such as inside nuclear reactors, wildfires, and outer space. However, path planning for these purposes can be extremely computationally intensive. The existing simulation, while acceptable for most intents and purposes, could take several hours, or even days to gather enough data to make statistically relevant inferences. By improving the computational efficiency and execution speed of the code base, it becomes possible to gather much more detailed or accurate information regarding the effectiveness of alterations to planning algorithms.

Furthermore, improvements in efficiency can be extended to physical hardware implementations of the agents. If these devices require complex or expensive hardware in order to perform path planning in real time, it may make their use infeasible or unattractive, due to the high risk of loss in the operational environment. As such, reducing the complexity of the path planning algorithms result in a cheaper, more disposable robot.

Finally, increases in efficiency can be turned into increases in accuracy. By reducing the computational complexity and time of the system, it becomes possible to use more sophisticated methods to plan better paths or gather more accurate data. For these reasons, it is integral that the system operates as efficiently as possible, thus underpinning why research into optimizations is important in this context.

Existing methods for belief consensus in multi-agent autonomous robotic systems using a reward-based path planning regime, measurement sharing, and belief consensus form the basis for the optimization problem. These methods are built upon through the addition of efficient long-range planning, optimization of data structures, and improvements to the computational implementation of the algorithms used in the system.

The existing simulation can be optimized using several methods. There are various textbooks and supplemental materials on basic algorithms, computational complexity, and data structures that are a part of the Computer Science undergraduate program at the University of Washington Tacoma that provide a survey of relevant information regarding programming optimizations. The Cormen Algorithms text is considered by many to be one of the preeminent books on the subject of computer algorithms and basic complexity analysis [1] and is relevant to the problem of optimization for the system. Additional references regarding algorithms and complexity provide further perspective, as well as pseudo-code implementation examples [2]. The undergraduate program also offers an elective artificial intelligence course with an associated text [3]. This text is relevant to mathematical predictive models, as well as computational complexity and intelligent behavior.

The existing simulation relies on belief consensus as the primary vehicle for the multi-agent nature of the system [10]. Belief consensus allows a method for multiple agents in an uncertain environment to reach an agreement on the current state of the environment [6]. Existing research on solving the consensus problem for multiple agents using belief consensus is leveraged to inform agent communication and path planning in the system [7]. The system can potentially be modified by taking inspiration from biological agents. There has been limited research done on how rats in groups search for an objective cooperatively in an unknown space that may inform more intelligent behavior for robotic agents under similar constraints [4]. Finally, in order to accurately assess the veracity of the function of the agents, it is necessary to have a simulation that approximates the spread of an actual fire. Work regarding modeling the stochastic systems using a grid-based fire spread technique with top-down and bottom-up fire controls has been previously published [8]. Further research has also been published regarding fire percolation thresholds [9] that may also be used to improve simulation models.

The remainder of this paper is organized as follows. In Section II, the problem, constraints of the state space, and capabilities of the agents are introduced. Section III provides a survey of the optimizations made to the simulation. Section IV examines the planning methods that were studied as part of this research project. Section V provides an overview of how path replanning takes place in the system, and the inferences that can be made from the information. Section VI proposes methods of further improving the system and includes conjectures of the efficacy of improvements that were not implemented in the simulation.

# Problem scenario

The simulation is directly based on a previous simulation that was used for the same purpose, to the extent of sharing some of the operative code [10]. The purpose of the system is to simulate a team of autonomous unmanned aerial vehicles (UAVs) mapping a wildfire. The simulation uses matrices to represent 25 square meter locations. These locations have either a value of 1 (on fire) or 0 (not on fire). The agents move around the space using “taxicab” movement [3], taking measurements from the state space when they occupy a specific location. The agents have a 10 percent chance of a false positive and a 10 percent chance of a false negative. The false measurements operate by checking the actual Boolean state of a location in the state space and inverting it ten percent of the time. The agent stores this measurement in its own measurement table as the uncertainty of the area.

The agents have limited communication between them, with their connections following a Laplacian matrix in the simulation. They operate using the principles of belief fusion [6]. After a specific amount of time has passed (the fusion interval), the agents use Bayesian fusion of their respective belief maps based on the connections outlined in the communication Laplacian. Thus, when the agents perform belief fusion, they only fuse with two of the other agents.

Other than belief fusion, the uncertainty of the agents updates at the beginning of every main loop. The uncertainty of the agents is plotted on a matrix using cartesian coordinates with uncertainty at each location. Each location is updated based on the uncertainty of surrounding locations and the amount of time since the location was last visited. That is, if several of the locations adjacent to a primary location are on fire, the uncertainty of the primary location will tend towards it being on fire. The converse is also true.

See reference [10] for further information on the existing system.

Chart

Description automatically generated

Fig 1. This figure shows an example of the estimated state of one of the UAVs following a 1000 step simulation.

# Simulation Optimizations

Several areas of the simulation were optimized. Since the system was primarily used for simulating the behavior of the robots, the main constraint was the amount of time needed to collect information in order to make accurate statistical inferences. Thus, optimizations and re-designs heavily favor execution speed over memory utilization. These redesigns and optimizations included altering the data structures of the system, improving code readability, altering functionality to reduce unnecessary accesses and writes, and capitalizing on existing optimizations native to MATLAB.

## Data Structure Redesign

First, the system was migrated from a primarily iteratively indexed matrix format to a graph format. Each simulation used a form of coordinate as an analog to the physical operating space of the agents. I.e., the matrix format accessed the “physical space’s” properties using a Cartesian coordinate system, where each location represents 25 square meters. The graph uses linearly indexed nodes to represent the same physical space (Figure 2). The graph format used was an unweighted digraph. This ideally would allow for greater efficiency in accessing and manipulating individual data points, as well as allowing for faster traversal of the state space using graph algorithms. Furthermore, it allowed for the use of built-in MATLAB graph functions, vastly simplifying the code and further improving speed. This change was most prevalent in the path-planning portion of the system. The graph acted as a “locational wrapper” for the underlying state matrices. 

Fig. 2. This figure shows a visualization of the 20x20 digraph informing the connections between individual nodes in the simulation.

## Comparison of Planning Methods

The agents plan a path to a fixed distance using a reward function that is comprised of an entropy-based reward and a proximity penalty weighted by an alpha parameter. See Section IV, subsection F for more information on the alpha parameter.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | (1) |

|  |  |  |
| --- | --- | --- |
|  | |  |
|  | |  |
|  | If location of another agent  Otherwise | (2) |
|  | | (3) |

While both methods rely on an uninformed depth-first search for traversing the state space, there are significant differences in their implementation. The previous method created strings of paths for the path-planning portion of the system. It then corrected these paths based on the bounds of the state space. Finally, the path was decoded (string to numerical coordinates) and fed to the system for use in the reward portion.

The new method uses graph successors recursively to plan the paths then accesses the locations in the state matrices. At each level the reward value is calculated and concatenated to the previous levels, along with the path; instead of calculating them in separate loops.

This method highlights the difference between Big-O order and actual computation complexity. While both methods should have the same order of computational complexity, in practice combining the operations into one loop instead of two reduces the coefficients of the order; as well as trailing factors that are not included in the Big-O complexity [1].

## Planning Optimization

The recursive method for path planning and reward calculation does not use pre-allocation for the arrays. As such, it is likely that they are grown with some form of amortization method (or represented as linked-list data structures in memory), reducing computational efficiency significantly due to excess writes. However, even with amortized array growth due to concatenation methods used in the path planning, there was a speed-up of nearly a factor of 10 using the graph-based path planning method vs the pre-planning method. With proper memory pre-allocation, this speedup can likely be further improved.

Computational geometry was also used to speed up searches and indexing in the system. Specifically, certain sections rely on linearized coordinates to access data. While only a small optimization, linear coordinates better match how data is stored in memory, resulting in a reduced number of computations to access data. They are also easily transformed from linear to multi-dimensional (4, 5), and vice versa (6).

|  |  |
| --- | --- |
|  | (4) |
|  | (5) |
|  | (6) |

## Adjustments to Fire Spread

The fire spread algorithm was also adjusted in the system to run faster. The previous version iteratively checked each location in the state space and spread the fire based on a probability if a location was not on fire while a number of adjacent locations were on fire. The rework of this section instead gathered all locations that were on fire and stochastically percolated the fire to adjacent locations that were not currently on fire. Since in the original simulation roughly half of the map would be typically on fire at the end of the simulation, this change was designed to reduce the number of checks performed.

This resulted in the beginning of the simulation having fewer checks, with the number of checks increasing over time. Since this method never checks the entire state space, it still is faster at all points in the simulation, however with more variability. Both simulations performed similarly on number of locations on fire over time (Table 1).

|  |  |  |
| --- | --- | --- |
| Duration | Method | Average Locations on Fire |
| 1000 | Previous | 274.4200 |
| 1000 | New | 272.0300 |
| 500 | Previous | 87.23 |
| 500 | New | 91.86 |
| 250 | Previous | 25.5 |
| 250 | New | 24.16 |

Table 1. This table shows the average number of locations that were on fire at the end of 100 simulations for both simulation method with durations of 250, 500, and 1000 steps respectively.



Fig. 3. This figure shows a typical example of fire spread over the course of the simulation if the previous method.



Fig. 4. This figure shows a typical example of fire spread using the new method.

## Parameter Passing and Scoping

Finally, the remaining major point of optimization is in the passing of arguments and scoping in the functions. The system utilizes several matrices for various tasks. These matrices are relatively small in the system, typically 400 locations or fewer. However, they are frequently read from and written to. The access and writing time of the matrices was improved by carefully scoping data wherever possible and by passing a single data point for writing. This takes advantage of MATLAB’s built-in optimizations for writing to matrices. Specifically, if a single data point is changed at the appropriate level, the matrix won’t be re-written. That is, if an entire matrix is passed as an argument and then written back, each index will be written to one or more times, or a new location will be made in memory and the matrix copied. Instead, only single points are overwritten in the top level to reduce the number of unnecessary reads and writes in the system.

# Path Planning Methods

Several methods were explored with the intent of improving the path selection quality. These methods ranged from purely mathematical theory to logical conclusions based on the analogous physical geometry of the system.

## Game Theory and Artificial Intelligence

First, information regarding basic game theory from an artificial intelligence standpoint was surveyed from available course texts [3]. Specifically, could the robots make an informed prediction regarding the position and path of nearby robots in an uncertain space. Methods for prediction were examined including Pareto optimality and Nash equilibrium. Ultimately, it was deemed to be too computationally expensive and lower priority to explore than some of the other available methods.

## The “Punch-Out” Method

The second method was based on the intelligent removal of nodes in the path planning graph. It was observed that the frontier of the path planning space for each individual agent aligned with polynomial expansion, or Pascal’s Triangle. This suggested that by removing some nodes from the path planning graph, it would be possible to dramatically reduce the number of explored paths without sacrificing accuracy. I.e., if all nodes adjacent to a low-value node have more than one path to reach it, the low-value node can be removed without affecting the frontier of the path search. Furthermore, if the node has a high centrality and a low value, removing it can dramatically reduce the number of paths that need to be processed in the tree. This method was explored but not implemented in code due to time constraints.

## Dijkstra’s and Alpha-Beta Inspired Method

The third method uses the principle of relaxation, inspired by combining parts of Dijkstra’s shortest path algorithm and alpha-beta pruning. Relaxation in Dijkstra’s algorithm allows for efficient searching in that paths are followed as long as they fall below a threshold value that is only increased when all available paths at that threshold are exhausted. As applied to this project, the highest value paths (or the lowest values of the complemented uncertainty) would be explored first. The relaxation threshold could be altered for each depth level in the planning tree to prevent the algorithm from being excessively greedy.

Second, trimming of the tree was inspired by alpha-beta pruning. The impetus of the pruning method is to not overly explore a suboptimal option. See [3] for more information on minimax with alpha-beta pruning. In this implementation, an upper relaxation threshold and a minimum pruning threshold is held in memory. If the path falls below the pruning threshold at any specific step, further exploration is pruned. E.g., two steps in a three-step path have been explored, and their total uncertainty is 0.1, then the maximum uncertainty of every resulting path that branches from the current sub-path must be less than or equal to 1.1. If there are paths at two steps with greater than or equal to 1.1 (such as 0.5 and 0.6 at each respective step), then exploring beyond the two-step frontier that resulted in 0.1 would be inefficient. If all paths fall below the relaxation threshold, the threshold is decreased by a fixed amount. This is the inverse of Dijkstra’s algorithm, in that the relaxation threshold moves from a maximum toward zero [1].

This method was also not included in the final product due to the complex nature of the algorithm and the difficulty of introducing it into the existing system.

## Long-Range Aggregation

The mid-range algorithm was designed to make a frontier “band” beyond the distance of the initial planning frontier. The band was then broken into eight sections (visually in the cardinal directions). Each section would be averaged, and added to the path of the agent, weighted based on the component vectors of the path relative to each section. This method was partially implemented in the system but was not completed due to time constraints.

## Mid-Range Aggregation

The short-range aggregation captures the average value of the next possible path that can be explored from the end of the current path. This is accomplished by selecting an area of the current state space matching the size of the explorable area from the end of the current path. Based on the current “taxi-cab” style of movement capable of the agents in the current simulation, it is a diamond-shaped area with a radius equal to the explorable depth. These spaces are summed, then averaged. The average was chosen for this method rather than the sum because of the potential for an emergent edge-avoidant behavior of the agent. I.e., if the agent was close to the edge of the map there would be fewer locations to sum.

|  |  |  |
| --- | --- | --- |
| Method | Average Error | Avg UAV Error |
| Average | 13.21 | 17.61 |
| Sum | 16.49 | 20.66 |

Table 2. This table shows a comparison of calculated average error for the unified map and individual agents over 10 simulations (due to time constraints).

## Alpha Parameter

The alpha parameter was added to the simulation to provide a weighting between the repulsive force and the entropy-based reward function. The simulation was tested with alpha values between zero and one with the average error at each step from 100 runs calculated [Figure 5]. Based on averaging the error for 100 runs of the simulation for each each alpha value between zero and one with a step width of 0.01, it appears that the current repulsive force method does not have a positive impact on error. This contrasts with the previous simulation, where the alpha parameter outperformed the entropy function in isolation [10]. This may be due to the way the reworked simulation calculates and weights the repulsive force; at every step vs. once after the path is calculated.

Fig

Fig. 5. This figure shows the average error of the simulation vs. the alpha parameter. An alpha of 1 uses only the entropy function, while an alpha of zero uses only the repulsive force.

# Path Replanning

To further improve the efficiency of the system, the frequency of replanning for individual agents was assessed. This was examined for two reasons: First, the frequency of replanning is important to the frequency of planning. E.g., if the agent follows its entire path 90% of the time, it would be inefficient to replan before it has traversed the entire path. Second, it informs the depth to plan to. That is, if the agent frequently replans after the second step of a three-step path, it is inefficient to plan to a three-step path or larger.

Both points can be used to optimize the simulation and potential devices based on the system. Indeed, if it is inefficient to used longer paths or frequent planning, one can use cheaper, slower hardware, reduced memory, or even physically smaller devices to the same effect. This is important because these devices would be expected to operate in dangerous or remote environments. Thus, having less expensive or less complex hardware makes losses significantly more acceptable in an operational environment.

## Replanning Frequency

It was found that the agents typically do not follow their plan for more than two steps, regardless of the given path depth parameter (Table 3). While the agent deviates from step 3 of the path more than half of the time, it still is much better than random, as there are 16 possible locations and 64 possible paths when searching a depth of three. The path planning method does not appear to create any major bias in visited positions beyond the number of available paths to that location (Figure 6). However, it is notable that the agents frequently revisit the location they started at. It was observed that agents will often create paths with a length of one or two by backtracking over areas that have already been added to their path. When correcting for this, it introduced a directional bias to the agents during exploration.

|  |  |
| --- | --- |
| Path Step | Probability of Following Path |
| 1 | 1.000 |
| 2 | 0.830 |
| 3 | 0.428 |
| 4 | 0.269 |

Table 3. This table shows the probability of an agent following its planned path up to or beyond each step for 100 simulations of 1000 steps with a path length of 4 with an alpha of 0.5. Step 4 is nearly the same as random, since the agent typically has 4 directions that it can move in at each step.

Fig. 6. This figure shows the path planning at all steps of the simulation over 100 steps.

# Future Work

There are still several potential optimization points in the base code. These optimizations include pre-allocation in the path planning portion of the system and some minor adjustments to access and parameter passing. Furthermore, the system may be improved by moving towards a more object-oriented approach, where each agent holds its own set of member variables and functions. Finally, the simulation can be made more realistic by using multi-threading or multi-core processing. Not only would this method more accurately reflect the agents and individual entities, additionally it may result in a reduced runtime through better use of available system resources.

Several methods remained unimplemented at the end of the project. These methods may be implemented in the future with a positive effect on the speed and/or accuracy of the simulation.

* Implementing game theory or additional artificial intelligence principle into the reward processes of the agents may significantly improve accuracy. This may have a negative impact on runtime, but with the current round of optimizations, the simulation is much faster and thus may be able to accommodate these changes while remaining faster than baseline.
* The “Punch-Out” Method may be used to significantly reduce the size of the search tree using any of the common tree-search methods. By removing nodes strategically, it stands to reason that runtime could be significantly improved without a major impact on available paths.
* The Pruning and Relaxation method is another potential way to reduce the number of checks being performed in the path search, which has the potential to significantly reduce the size of the state space. It should likely be explored separately to the Punch-Out method in order to prevent the tree from being over-pruned.
* The Long-Range aggregation method is partially implemented and may offer an improvement in error over the Mid-Range method. This method should be explored to determine the value of planning at a distance.
* Either aggregation method may benefit from a beta weighting factor. This would potentially allow several methods to be run simultaneously and may lead to significant emergent behavior in the agents.

# Conclusion

The project followed many potential optimizations and improvements for the existing simulation. While several methods were not implemented due to time constraints, there were still many improvements made to the underlying system. These optimizations resulted in a nearly 10 times faster simulation, with similar capabilities to its predecessor. The work done over the course of this project forms the basis for potential future improvements and more advanced planning methods.

##### References

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, 2009*.
2. S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, *Algorithms*. New Delhi: McGraw-Hill Education (India), 2016.
3. S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Upper Saddle River: Prentice-Hall, 2010.
4. Máté Nagy, Attila Horicsányi, Enikő Kubinyi, Iain D. Couzin, Gábor Vásárhelyi, Andrea Flack, Tamás Vicsek, *Synergistic Benefits of Group Search in Rats,* Current Biology,Volume 30, Issue 23, 2020,Pages 4733-4738
5. Akshat Agarwal, Sumit Kumar, Katia Sycara, *Learning Transferable* Cooperative Behavior in Multi-Agent Teams, Cornell University, 2019
6. Olfati-Saber R., Franco E., Frazzoli E., Shamma J.S. Belief *Consensus and Distributed Hypothesis Testing in Sensor Networks*. In: Antsaklis P.J., Tabuada P. (eds) Networked Embedded Sensing and Control. Lecture Notes in Control and Information Science, vol 331.
7. Saber, Reza Olfati and Murray, Richard M. (2003) *Consensus protocols for networks of dynamic agents.* In: Proceedings of the 2003 American Control Conference, 2003. Vol.2. IEEE.
8. Kennedy, M.C., McKenzie, D. *Using a stochastic model and cross-scale analysis to evaluate controls on historical low-severity fire regimes*. Landscape Ecol 25, 1561–1573 (2010).
9. McKenzie, D., Kennedy, M. *Power laws reveal phase transitions in landscape controls of fire regimes*. Nat Commun 3, 726 (2012).
10. R. M. Seam and M. J. McCourt, "*Distributed Estimation of an Uncertain Environment using Belief Consensus and Measurement Sharing*," 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 1362-1367