Universiteit Leiden
The Netherlands

Opleiding Informatica

Efficiency of Stigmergic Communication Protocols in Multi-Agent Maze Searches

Ebenezer Fosu

Supervisors:
Mike Preuss

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/1/2026

Abstract

Efficient exploration of unknown environments remains a point of interest for researchers. Because of the increased availability of inexpensive hardware, we can use multiple robots to perform this tedious work for us. In the case that network is limited, robots must use other modes of communication to share information. This thesis aims to evaluate the efficiency of the Multi-Robot Depth-First Search (MR-DFS) algorithm, which uses a stigmergic approach, in non-perfect maze searches using two different maze generation types. Specifically, we compare the performance of MR-DFS robots to other known maze solver robots in mazes generated using the Depth-First Search (DFS) and Prim algorithms. We find that in large mazes the MR-DFS significantly reduces the time to explore a maze compared to other known graph exploring methods.

Contents

1 Introduction

For a long time humans have been interested in how we can effectively explore unknown environments. This abstract problem can be applied to various useful situations, ranging from autonomous cleaning appliances mapping the layout of your house to robots exploring intricate cave systems. With the rising availability of cheap robots, this field has become increasingly more interesting. Research of this problem has been done under various conditions, changing variables such as obstacle density, communication constraints, and collaboration methods between robots. The latter two conditions will be focused on in this thesis, which focuses on a specific communication method between agents in maze searches. The challenge of navigating unknown spaces becomes more difficult when robots need to consider communicating effectively with each other. There are many different ways in which robots can do so: When there is a network available, robots can send new information over this network to other robots. When network is limited, this information could be broadcasted up to a certain range.

The communication method of interest in this thesis is a stigmergic approach: Robots leave information in the form of markings or traces inside of the maze that can be perceived by other robots. This information influences each robot's decision-making and can lead to more efficient exploration of the environment. The first known stigmergic maze solving algorithm, the Trémaux algorithm [1], is still popular today and utilizes edge markings to store information in the maze. It is however not particularly compatible with multiple robots. Brass et. al. [2] provide an algorithm that adapts the Trémaux algorithm into an algorithm that works for multiple robots. In this thesis we investigate the efficiency of their proposed Multi-Robot Depth-First Search (MR-DFS) algorithm in general graphs, which is a branch that was missing in their research. By doing so, we wish to provide more insights on the extent to which MR-DFS robots can be useful in search or exploration problems where decentralized communication methods are necessary.

1.1 Problem Formulation

The main research question of this thesis is:
How efficient are stigmergic communication protocols in multi-agent coordinated non-perfect maze searches?

To further investigate this problem, the research question has been divided into multiple subquestions, namely:

SQ1: How can we generate solvable non-perfect mazes?

SQ2: How can an autonomous agent optimally solve a maze?

SQ3: How can the selected agents effectively communicate by making use of unique stigmergic communication protocols?

1.2 Thesis overview

The structure of this thesis is as follows: Section 2 describes the state of the art research findings about robot exploration, eventually focusing more on cooperative exploration and the use of decentralized communication protocols. In section 3, the approach for this thesis will be discussed and elaborated on. Section 4 will contain useful information about the implementation of the software that has been written to aid in performing the experiments. Section 5 describes the experiment setup and the results. Section 6 summarizes the thesis.

2 Related Work

Previous research on the exploration of unknown environments using robots has happened under various conditions. The earliest approaches usually assumed a single robot that has to efficiently explore an a priori (partially) unknown area until it has been covered fully [3]. Well-known strategies such as Depth-First search (DFS), Breadth-First Search (BFS) and their variants have been widely analyzed because of their efficiency in tree and graph structures. With the high availability of low-cost hardware, the problem of exploration can be parallelized using multiple robots. However, this introduces the need for effective communication between robots to reduce overhead and redundant exploration. Such methods in which robots can communicate have been widely researched [4] [5].

When robots are able to communicate with each other with no constraints, we say that the communication is centralized. Centralized protocols have been widely studied in research concerning optimal routing [6], where efficient use of globally available information is key. This communication method can be highly effective, but it relies on the assumption that there no constraints such as a limited communication range or the complete absence of a network. This introduces the necessity to explore communication methods with limited capabilities. When there are such constraints on the communication between the robots, such as a distance range [7] or a line-of-sight protocol [8], we call it decentralized. An unmentioned decentralized approach is one where communication happens through the environment, which we refer to as stigmergic communication. In such systems, robots leave markings or other forms of information in the environment, which in turn can be observed by other robots, enabling communication without explicit inter-robot message exchange.

Although stigmergic approaches have been employed in more environments with little spatial constraints [9], our focus lies on stigmergic methods in more spatially constrained environments, namely mazes. Since mazes can be seen as trees or graphs (dependent on whether there are cycles present), there is a large overlap in research in optimal multi-robot maze-solving and tree or graph exploration. One of the earliest stigmergic methods was applied to single-agent maze-solving. The Trémaux algorithm [1], uses markings in passages to provide information to its future self, which allows it to explore mazes with loops, a task that is usually challenging for autonomous agents with no memory. Other research has provided valuable insights in multi-robot tree exploration [10] [2] and graph exploration alike [11] [2]. In particular, [2] provides a useful algorithm that combines DFS and the stigmergic property of the Trémaux algorithm to facilitate efficient communication between robots in both tree and graph-like environments. The study shows a promising analysis on the algorithm for trees, but it does not provide a similar analysis for general graphs.

This thesis builds upon these previously mentioned studies by evaluating the efficiency of the algorithm provided in [2] for general graphs using simulations.

3 Methodology

To address the various sub-questions that arose from Section 1, it is important that there is a clear understanding of all the components underlying the proposed approach. As the end goal is to simulate and evaluate stigmergic communication in mazes, we must understand the underlying choices made for this simulation. The definition of efficiency in the context of this thesis, the generation of solvable non-perfect mazes and the mechanisms of the agents are all addressed in this section.

3.1 Mazes

3.1.1 Definition

In the context of this thesis, we define a maze as a connected, undirected graph with two distinct nodes, namely *Start* and *End*, where the Start node is always found on the outer wall. The goal is to reach the end node while beginning from the start node. The degree of each vertex in the graph is at least 1 and at most 4. In other words, each node in the graph is connected to at least one other node and at most four other nodes in the graph. The graph can be depicted as a two-dimensional grid of cells with walls and open spaces: These open spaces are the connected nodes of the graph. The mazes used in this thesis are non-perfect, meaning that there can be more than one possible path from the start node to the end node. This can also be described as the fact that there are loops present in the maze that can be used as shortcuts. Figure 1 shows a visualization of the aforementioned description.

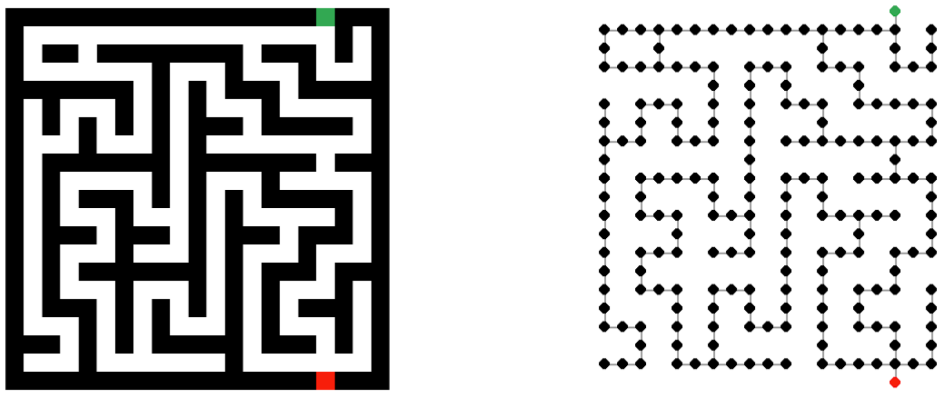


Figure 1: Example of a maze with a Start and End node and its corresponding graph

3.1.2 Generation

Both algorithms used to generate mazes in this thesis generate a minimal spanning tree that is later transformed into a non-perfect maze. In other words, they generate a perfect maze, one without loops, after which certain walls are systematically removed to create a non-perfect maze.

The **Depth-First Search** algorithm uses depth-first traversal to create a perfect maze in the following fashion: The algorithm begins by choosing a random cell as the starting point. Then it randomly chooses a wall of the current cell and opens a passage to an adjacent unvisited cell. This then becomes the current cell. It repeats this step until all adjacent cells of the current cell have been visited. When this happens, the algorithm backs up to the previous visited cell and repeats the second step. At some point, the algorithm backtracks all the way to the beginning, after which it finishes with a minimal spanning tree. Because of the depth-first traversal mechanism, the algorithm opens up paths as deep as possible before backtracking and opening up new paths, resulting in long corridors with relatively few branches. An example of a maze generated using the Depth-First Search algorithm can be found in Figure 2

Prim's algorithm is originally used to create minimal spanning trees for weighted connected trees. When considering which neighboring cell to visit, it chooses the neighbor with the lowest weight. Since in the context of this thesis the graphs are unweighted, which means that all edges are equal in length, the algorithm is slightly adapted to suit the needs of maze generation. Instead of assigning an explicit random weight to every possible edge, we keep track of all neighboring unvisited nodes and choose a corresponding edge randomly at each step. Randomly sampling the edges is probabilistically equivalent to assigning random weights to the edges before the algorithm starts. It is also computationally less expensive, as it avoids storing information that is only used once during the algorithm. The modified Prim's algorithm works as follows: The algorithm starts by choosing a random cell as the starting point. It also keeps track of all neighboring nodes that are still unexplored. It then randomly selects an unexplored neighbor and opens up the path to the nearest visited node. The updated unexplored neighbors are added to the set of old neighbors and the algorithm continues until all cells are visited. This frontier-based selection results in a minimal spanning tree with a high amount of branches, because the maze expands at different places both simultaneously and randomly. An example of a maze generated using Prim's algorithm can be found in Figure 2

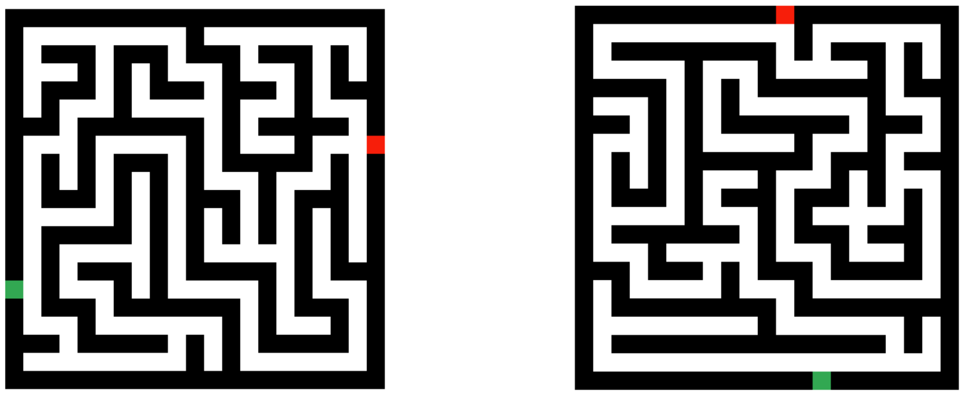


Figure 2: Example of mazes generated with DFS (left) and Prim's (right) algorithm

Since all nodes are reachable, the start and end nodes can be selected by opening up any segment of the outer walls that have a neighboring open cell. To increase the chances of a greater solution path length, the start and end node are always selected in opposite walls.

As mentioned earlier, the described algorithms create perfect mazes. To transform these into non-perfect mazes, we need to systematically remove walls to create loops, which can potentially create new solution paths. The mechanism for adding loops is explained further in Section 4.

3.2 Agents

There are various ways an autonomous agent can solve an unexplored maze. The *wall following* algorithm implies that the agent keeps its left or right 'hand' on the corresponding wall at all times and follows it until it reaches the end [12]. If there are loops present in the maze or if the exit is not connected to the component that contains the entrance, the wall follower will make a full return to the entrance. In the case that the maze is simply connected, which means that all walls are connected, (and thus the entrance and exit are contained by the same component,) this algorithm always finds the exit. For perfect mazes, this algorithm essentially performs a depth-first search, ensuring that all reachable nodes are eventually visited at least once.

Another useful, more efficient algorithm that allows an autonomous agent to solve a maze was designed by Charles Pierre Trémaux. The Trémaux algorithm [12] [1] uses markings at junctions to denote whether an agent has traversed a passage before: Whenever the agent enters or leaves a junction, the corresponding access point is marked. When the agent is in a junction, which is any node with two or more neighboring open spaces or a dead end, it uses the *first applicable rule* of a set of rules to decide where to go next.

1. If only the entrance you just came from is marked, pick an arbitrary unmarked entrance, if any.

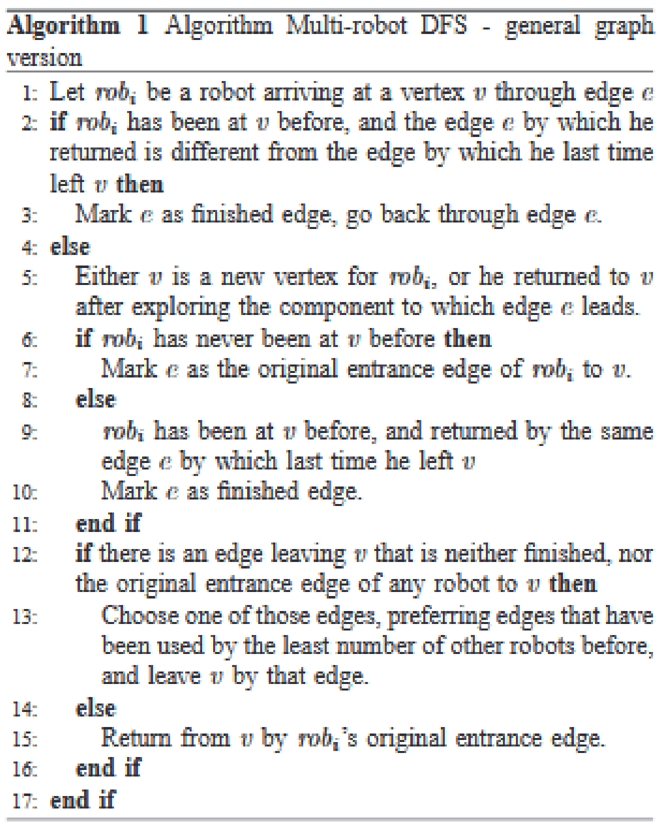
2. If all entrances are marked, go back through the entrance you just came from, unless it is marked twice. This rule will apply whenever you reach a dead end.

3. Pick any entrance with the fewest marks (zero if possible, else one)

This set of rules ensures that a double marked passage is never taken, allowing the agent to treat a loop in the maze as a dead end. The rules ensure that the Trémaux algorithm can solve both perfect and non-perfect mazes. When the agent reaches the exit, a path back to the entrance can be constructed by following all the singly marked passages. However, this algorithm does not guarantee the shortest possible path.

In the context of this thesis, we are interested in multi-agent maze solving algorithms. Wall follower agents can find an exit individually but cannot really benefit from the advantages that collaboration can offer. The Trémaux algorithm gives more possibility for collaboration because of the markings in the environment. However, the algorithm was designed with a single agent in mind. A naive approach to solve this design issue would be to simply allow multiple agents to read each other's markings. Without modifications to the algorithm however, there are some issues that arise. The agents may traverse the same edges before they are properly updated, leading to redundant exploration. There is also no rule set for situations where multiple agents simultaneously traverse an edge, which can lead to unexpected behavior. In short, even with multiple ”collaborative” Trémaux agents, collaboration is not optimized.

Brass et al. [2] propose a maze algorithm that adapts the concept of the Trémaux algorithm to optimize the collaboration potential between the agents. The pseudo-code for this Multi-robot DFS (MR-DFS) algorithm can be found in Figure 3. Similarly to the Trémaux algorithm, the MR-DFS prioritizes choosing edges that have not been fully explored yet. Additionally, it makes a distinction between original entrances to the vertices and completed edges, similar to how the Trémaux algorithm has single and double markings respectively. The performance of the MR-DFS algorithm is of interest in this thesis. Brass et al. performed experiments on a different version of the algorithm that works for trees, delivering promising results: For a small amount of robots (N=2 or N=3), the exploration time is very close to the upper bound. Additionally, when the number of robots increases, the time steps needed to solve the maze decreases. The paper does not provide analysis results for the algorithm that works for general graphs. In the rest of this thesis, experiments will be conducted using simulations to get a better understanding of the efficiency of the MR-DFS algorithm for general graphs. This will be done by first interpreting the results of the aforementioned algorithms and the MR-DFS algorithm and then by comparing them.



Algorithm 1 Algorithm Multi-robot DFS - general graph version

```
Let $rob_i$ be a robot arriving at a vertex $v$ through edge $e$
if $rob_i$ has been at $v$ before, and the edge $e$ by which he
returned is different from the edge by which he last time
left $v$ then
    Mark $e$ as finished edge, go back through edge $e$.
else
    Either $v$ is a new vertex for $rob_i$, or he returned to $v$
    after exploring the component to which edge $e$ leads.
    if $rob_i$ has never been at $v$ before then
        Mark $e$ as the original entrance edge of $rob_i$ to $v$.
    else
        $rob_i$ has been at $v$ before, and returned by the same
        edge $e$ by which last time he left $v$
        Mark $e$ as finished edge.
    end if
    if there is an edge leaving $v$ that is neither finished, nor
    the original entrance edge of any robot to $v$ then
        Choose one of those edges, preferring edges that have
        been used by the least number of other robots before,
        and leave $v$ by that edge.
    else
        Return from $v$ by $rob_i$'s original entrance edge.
    end if
end if
```

Figure 3: Pseudocode for the MR-DFS algorithm for general graphs

3.3 Metrics

To compare the different algorithms to each other, we need to select fair and quantifiable metrics to measure performance. The first metric is the **exploration time** T_e: All the robots synchronously move at each time step. For this reason, T_e also denotes the distance traveled by each robot. This is useful for the second metric, which is the **solution ratio** r_s. This metric effectively compares the total distance traveled to the shortest path from the entrance to the exit, which is calculated beforehand. Important to note is that we are *not* reconstructing the final path taken for this metric. We are simply calculating the ratio of distance traveled to the optimal distance. These two metrics will provide us valuable insights in how efficient the robots solve the maze.

4 Implementation

The methods described in Section 3 have been implemented in a Python project. This project connects all the earlier mentioned components and facilitates the simulations. To generate perfect mazes, the `mazelib` [13] library was used. This library contains a generator method for both the DFS and Prim algorithm, namely the `BacktrackingGenerator` and the `Prims` generators respectively.

After the maze has been generated, a custom function systematically removes walls from the maze to create loops and potentially introduce shortcuts. This function checks for every wall cell in the maze whether it is in a section of consecutive walls. In this case, one of the walls may be removed to provide access to the maze component behind the wall section. There are parameters that determine the minimum length of the consecutive wall for the wall cells to be considered candidates for removal. The result is a set of candidate cells that may be removed. The function then sequentially removes candidates based on a provided percentage parameter. When a candidate is removed, nearing candidates are removed from the set to avoid an invalid maze configuration. The result is a non-perfect solvable maze that was generated with either the DFS or Prim algorithm. All the aforementioned parameters have been tuned through trial and error during preliminary testing. An example of how the artificial loops have the potential to create shortcuts can be viewed in Figure 4. The light gray square is the one that is removed to potentially create new loops. We can see that the removal of the square in Variation A causes the maze to have a shorter optimal path, while the removal of the square in Variation B does not change the optimal path of the maze.

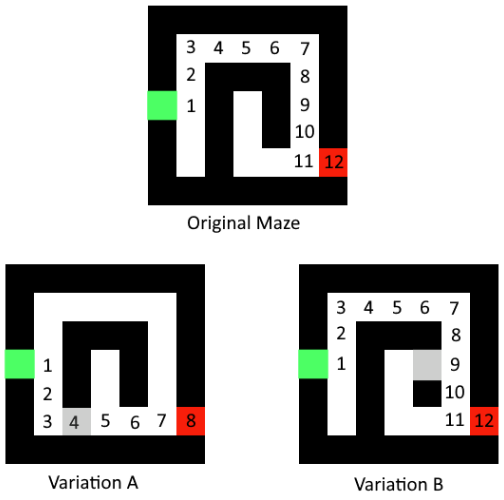


Figure 4: Illustration that shows shortcut potential

The `pygame` [14] library was used to visualize the simulations during both the development and experiment phase. The library facilitates easy visualization of the maze by allowing us to create visible cells for each cell in the maze. Furthermore, the simulation loop can be elegantly displayed when combined with the agent movement logic. Not only does this show a clear representation of agent behavior and decision-making during the simulation, but it also of great while troubleshooting issues or unexpected agent behavior.

The `Agent` class is a custom class that houses all the logic behind agent movement and communication protocols. The agent is designed in such a manner that it can only look in front of them using the `probe_front()` function. This means that if they want to see what is around them, they must first turn and then use the probing function to obtain the data about its surroundings. The class holds methods for all the maze solving algorithms used in this thesis: `hug_left`/`hug_right`, `dfs` and `mrdfs`. These methods all make use of some logic to determine where to move to during the current time step. At the end of the time step, the `move_forward()` method moves the agent in the desired direction.

Combining these libraries, methods and the Agent class gives us a suitable environment to run simulations in: The maze can be set up using the respective `setup_maze()` method, which generates the non-perfect maze, its entrance and exit and the shortest solution. After this, the agents are initialized using the respective `setup_agents()` method, which creates the Agent objects, configures the necessary fields and sets their position to the maze entrance. Hereafter the simulation loop is ran until the agents find the exit is found, after which the program terminates.

5 Experiments

Now that there is an understanding of the simulation environment, the focus shifts to the experiments. To be able to draw conclusions about the efficiency of MR-DFS in non-perfect mazes, experiments have been done to compare its metrics against other known maze solving algorithms. This section goes over the setup and results of each of the experiments. In each of the experiments, we are interested in the performance of the concerned agents in different environments. For this reason, we run the comparisons in **six** different environments, varying in both size and generation type, unless mentioned otherwise. The sizes used in the experiments are `10x10`, `10x30` and `35x35` to measure performance in small, narrow and large mazes respectively. As mentioned earlier in the thesis, the entrance of the maze is always on the outer wall. The location of the exit differs for each experiment and is therefore specified in the respective subsection. When the exit is on the outer wall, it is always found on the wall opposite of the wall that houses the entrance. When the entrance is inside of the maze, we use a custom function that finds the most central dead end in the maze using Euclidean distances. It is important to note that all the results of the experiments are median values to take outliers into account.

5.1 Experiment 1

In the first experiment, we compare 2 Wall Follower agents to 2 MR-DFS agents. Important to note is that in this experiment, one of the wall followers hugs the left wall while the other hugs the right wall. The exit of the maze is found in the outer wall. The results of the first experiment can be found in Figure 5.

Time to explore

	DFS			**PRIM**		
	10x10	10x30	35x35	10x10	10x30	35x35
2 WALL FOLLOWERS	99	299	631	147	370	1280
2 MR-DFS	99	411	1163	134	384	1092

Ratio of solution

	DFS			**PRIM**		
	10x10	10x30	35x35	10x10	10x30	35x35
2 WALL FOLLOWERS	1.39	1.80	2.71	3.78	3.94	10.51
2 MR-DFS	1.73	2.35	5.03	3.53	4.14	9.69

Figure 5: Experiment 1 results

From the results we can see that the wall followers and MR-DFS robots perform quite similarly, especially on smaller mazes. On the larger DFS mazes, the wall followers outperform the DFS by a large margin. Visualization shows that the outer exit works in the favor of the wall followers, because they are more inclined to trace the outer walls compared to the MR-DFS robot, which often end up exploring towards the center of the maze. In small mazes, this slight advantage is negligible, but it is apparent that the wall followers can make great use of this design advantage in larger mazes. This experiment can not be performed with the exit inside of the maze. Wall followers can not reach all cells in a non-perfect maze, so there can be cases where the wall followers loop back to the entrance without finding the exit. Regardless, it is interesting to see how a naive approach to solving mazes can still outperform a more complex algorithm when the playing field is leveled. Another, somewhat trivial, thing we can take away from this experiment is that the ratio of solution r_s grows as the maze size grows, which can be accredited to the fact that it is probabilistically more likely for the agents to take wrong turns in larger mazes.

5.2 Experiment 2

In the second experiment, we compare one regular DFS robot to two MR-DFS robots. We want to examine whether the introduction of an additional collaborative robot provides significant difference in performance compared to a single robot. The exit of the maze is found on the outer wall. The results of the experiment can be found in Figure 6.

Time to explore

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
1 DFS	205	586	2088	197	507	2257
2 MR-DFS	142	359	1329	129	403	1293

Ratio of solution

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
1 DFS	3.71	3.13	9.35	5.21	5.79	17.69
2 MR-DFS	1.96	2.19	6.45	3.34	4.65	9.51

Figure 6: Experiment 2 results

We can see from the results that the introduction of another robot provides a substantial difference in performance, even in smaller mazes. The collaboration that is introduced by the MR-DFS protocol allows for parallelization of the workload, which increases the efficiency of the maze walk.

5.3 Experiment 3

This experiment has a nearly identical setup to Experiment 2. The only difference is that in this experiment, the exit is found inside of the maze. This decreases the average optimal solution length. We are interested in whether the ratio of solution r_s changes accordingly. To reiterate the setup, we are again comparing 1 DFS robot to 2 MR-DFS robots. The results of the experiment are found in

Figure 7.

Time to explore

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
1 DFS	270	501	3229	206	443	2807
2 MR-DFS	123	424	1689	125	407	1596

Ratio of solution

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
1 DFS	3.93	6.36	18.07	8.64	8.02	36.04
2 MR-DFS	1.99	4.28	10.71	6.22	6.65	20.27

Figure 7: Experiment 3 results

When we compare the result of experiments 2 and 3, in particular the ratio of solution, it is interesting to note that the ratio does not decrease when the exit is moved to the center of the maze. Conversely, the ratio increases when the exit is located at the center, which disproves our earlier hypothesis. This could be due to multiple reasons. A potential reason could be that when the exit is inside of the maze, it is found in a dead end, usually after a large amount of branches. If the robot skips the passage on the final junction to the exit, it must reach another dead end until it backtracks to the previous dead end. The slightly higher branching factor inside of the maze may mean that the robot has to explore many different areas before backtracking to the junction where the exit is located, which increases the exploration time and ratio of solution. In the case that the exit is on the outer wall, the robot may have an easier time finding the exit. This is because the passages next to the outer wall are usually straighter compared to passages inside of the maze. This is due to the generation algorithm being bounded by the dimension constraints, which does not allow it to make a turn outside of the provided space. Whether this is the actual reason why the results differ from the expectations is unknown.

5.4 Experiment 4

In the fourth experiment, we compare the performance of two DFS robots to that of two MR-DFS robots. We are interested in seeing whether the collaborative aspect of the MR-DFS algorithm truly influences the performance of the robots when we compare it to two similar, non-collaborative robots. The exit can again be found in a dead end located near the center of the maze. The results of this experiment can be found in Figure 8.

Time to explore

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
2 DFS	105	364	1904	118	308	1522
2 MR-DFS	116	262	1335	131	327	1160

Ratio of solution

	DFS			PRIM		
	10x10	10x30	35x35	10x10	10x30	35x35
2 DFS	1.78	3.99	13.50	4.58	6.41	18.39
2 MR-DFS	2.21	2.57	7.98	4.67	6.74	12.11

Figure 8: Experiment 4 results

By analyzing the results, we can see that in small mazes, the difference in performance between non-collaborative DFS and MR-DFS robots is not significant. When the maze gets larger, we observe that both the time to explore and the ratio of the solution goes down significantly. We can deduce that the advantages of MR-DFS are utilized best when the maze gets larger, which is where non collaborative DFS robots tend to perform worse.

5.5 Experiment 5

In this experiment, we compare four DFS robots to two MR-DFS robots. We are interested in investigating whether the benefits of MR-DFS can outperform the numerical advantage of the non-collaborative DFS robots. Because we have seen that the advantages of MR-DFS are more apparent in large mazes, we run this experiment using only the `35x35` maze size. This experiment utilizes mazes that have exits on the outer opposite wall. The results of this experiment can be found in Figure 9.

Time to explore

	DFS	**PRIM**
4 DFS	1551	1232
2 MR-DFS	1430	1444

Ratio of solution

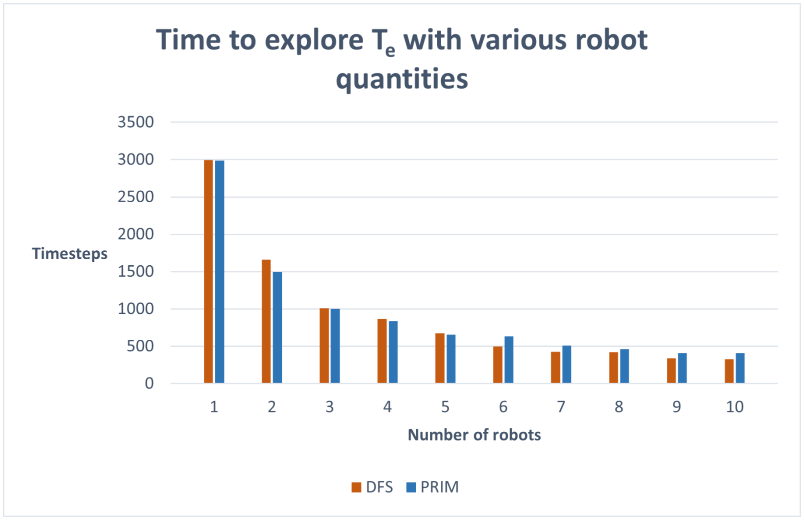
	DFS	**PRIM**
4 DFS	7.21	9.39
2 MR-DFS	6.45	10.31

Figure 9: Experiment 5 results

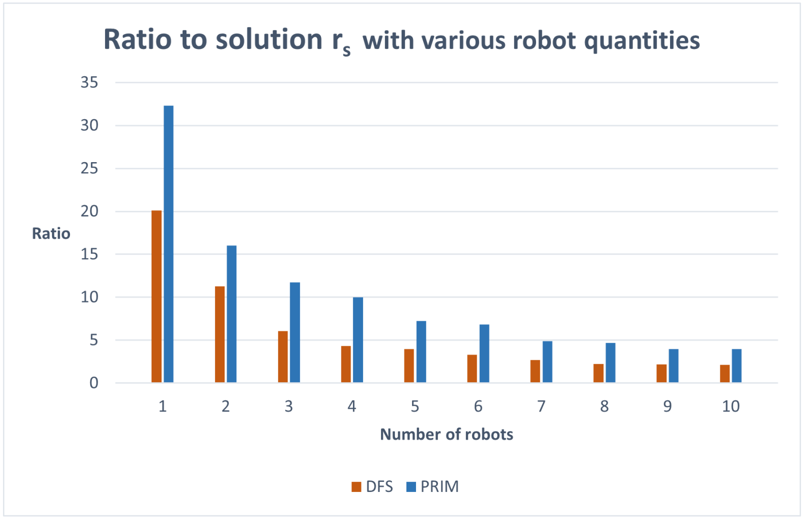
Observing the results show us that there is a slight difference in performance for the different sets of robots. The set of four DFS robots outperform the two MR-DFS robots in mazes generated with the DFS algorithm, and the opposite is true for mazes generated with Prim's algorithm. We can argue that the performance is partly based on the structure of the maze: Mazes generated with Prim's algorithm generally have more junctions, which means that the chance to have to backtrack is higher compared to mazes generated with DFS. The number advantage of the first set of agents can explore these types of mazes better, because the agents have a higher exploration throughput. Conversely, in DFS mazes there is a smaller amount of junctions, so there is generally a lower chance to have to backtrack when taking the wrong passage. MR-DFS robots can make better use of this maze property, because their exploration of unvisited areas of the maze is more coordinated and less redundant compared to regular group of DFS robots. It is unclear whether this explanation is responsible for the observed results. However, it is a likely contributing cause. In any case, two MR-DFS robots perform similar to 4 regular DFS robots, which is a great feat.

5.6 Experiment 6

We have seen in the previous experiments that the gain from the collaborative MR-DFS robots can be utilized best in larger mazes. The last point of interest is whether there is an optimal amount of robots for maximizing performance. In this experiment, we will deploy different numbers of robots in mazes of size `35x35` with the exit at the center of the maze and evaluate their performance. The results of this experiment are presented in Figure 10.



(a) Graph showing T_e for various robot quantities



(b) Graph showing r_s for various robot quantities

Figure 10: Graphs showing the results of Experiment 6

The results show that adding more robots certainly affects performance: An increase in the amount of robots lowers both T_e and r_s for each increment. However, the gains in performance start to noticeably become less after $n = 4$ robots. Beyond that, the improvement becomes marginal. This can imply that adding more than 4 robots increases the risk of redundant exploration.

6 Conclusions

In this thesis, we analyzed the efficiency of the MR-DFS algorithm in multi-agent non-perfect maze searches. We explored methods for both generating solvable non-perfect mazes and solving mazes optimally as an autonomous agent. Decisions were made for quantifiable, fair metrics to evaluate the performance of the robots during the simulation. Experiments have been conducted to compare the performance of various non-stigmergic and stigmergic robots. Using the methodology developed in Section 3, we were able to answer the sub-questions posed in Section 1.1, which where:

SQ1: How can we generate solvable non-perfect mazes?

SQ2: How can an autonomous agent optimally solve a maze?

SQ3: How can the selected agents effectively communicate by making use of unique stigmergic communication protocols?

The conducted experiments have shown that the MR-DFS robots outperform regular DFS robots in larger mazes. In small mazes, the difference between two different pairs of agents is not significant. Sometimes the MR-DFS robots are even outperformed by naive maze solving solutions such as wall followers. Experiment 5 has also shown that increasing the robot count beyond $n = 2$ provides additional performance gain, although marginal when n exceeds 4. We can conclude that utilizing MR-DFS robots in large non-perfect mazes is a more efficient approach compared to more common non-complex approaches such as wall followers and DFS robots.

7 Further Research

There are some improvements that can be made in further research. Brass [2] concludes the paper with the idea that the most important step towards practical applications of MR-DFS robots is to reconsider the robot movement mechanics. In his paper and in this thesis, we have assumed that robots can only move once per timestep and that all robots do so synchronously. In real life however, movement is asynchronous and the speed which the robots travel with may also change during a maze search. Examples would be when a robot can accelerate if it finds itself in a long, straight passage and when a robot needs to slow down if there are a lot of corners in its passage. Furthermore, there were some parameters that have been decided on during preliminary testing. Experiments could be ran to investigate the influence of loop quantity on robot performance. Another approach would be comparing stigmergic communication protocols to gossip protocols, which can provide even more insight for decision making around decentralized communication methods.

References

[1] Jean Pelletier-Thibert. Public conference. 2010.

[2] Peter Brass, Flavio Cabrera-Mora, and Jizhong Xiao. Multi-Robot Tree and Graph Exploration. *IEEE Transactions on Robotics*, 2011.

[3] Yongguo Mei, Yung-Hsiang Lu, C.S.G. Lee, and Y.C. Hu. Energy-efficient mobile robot exploration. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 505–511, 2006.

[4] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot Systems: A Classification Focused on Coordination. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, October 2004.

[5] Francesco Amigoni, Jacopo Banfi, and Nicola Basilico. Multirobot Exploration of Communication-Restricted Environments: A Survey. *IEEE Intelligent Systems*, 32(6):48–57, November 2017.

[6] Bojan Crnkóvic, Stefan Ivíc, and Mila Zovko. Fast algorithm for centralized multi-agent maze exploration. *arXiv eprints, https://arxiv.org/abs/2310.02121*, 2025.

[7] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli. The sensor-based random graph method for cooperative robot exploration. *IEEE-ASME T Mech*, 14(2):163, 2009.

[8] Ethan Stump, Nathan Michael, Vijay Kumar, and Volkan Isler. Visibility-based deployment of robot formations for communication maintenance. In *2011 IEEE International Conference on Robotics and Automation*, pages 4498–4505, 2011.

[9] Tüze Kuyucu, Ivan Tanev, and Katsunori Shimohara. Superadditive effect of multi-robot coordination in the exploration of unknown environments via stigmergy. *Neurocomputing*, 148:83–90, 2015.

[10] Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. In Martín Farach-Colton, editor, *LATIN 2004: Theoretical Informatics*, pages 141–151, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[11] Romain Cosson. Ariadne and Theseus: Exploration and Rendezvous with Two Mobile Agents in an Unknown Graph, July 2024. arXiv:2403.07748 [cs].

[12] HandWiki. Maze Solving Algorithm. `https://encyclopedia.pub/entry/33079`, 2022.

[13] John Science. Mazelib Github. `https://github.com/john-science/mazelib`, 2024.

[14] Pygame Website. `https://www.pygame.org/news`, 2024.