On the Randomized Gathering of mobile agents in a polygon

A Project Report

submitted by

SHIVAM CHOLIN

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

May 2022

CERTIFICATE

This is to certify that the report entitled **On the Randomized Gathering of mobile agents in a polygon**, submitted by **Shivam Cholin [CS20M061]**, to the Indian Institute of Technology Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the project work carried out by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. John Augustine
Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

Firstly, I would like to express my special thanks of gratitude to my guide Prof. John Augustine for the continuous support, patience, motivation, enthusiasm, and immense knowledge. I would again thank him for introducing me to this vast and ever-growing field of distributed computing and providing excellent course in Distributed Algorithms in which I had the opportunity to be his teaching assistant and learn about this field. His guidance helped me throughout the project and while writing this report. I cannot ask for a better mentor.

I would like to thank my lab-mates Srikkanth (CS21S019), Nischith Shadagopan M N (CS18B101) and Shashank (CS20M058) for their help in looking for related research papers and analyzing and solving sub-problems. And thanks to Vaichalkar Aditya Girish (CS20B084) , Trambadia Mantra Niteshbhai (CS20B083) and Saptarshi Sen (CS20B068) for their help in simulation framework.

ABSTRACT

KEYWORDS: Distributed, Mobile Agents, Polygon, Randomness and Synchronous.

Distributed computing in mobile agents is a vast application field. It can represent many systems where several mobile agents cooperate and perform various tasks to achieve a common goal. Goals such as information spreading, surveillance, Search and rescue, gathering, etc.

Mobile agents can observe their surroundings, communicate with other nearby agents, and move around, and one of the most researched challenges involving mobile agents is gathering. The mobile agents are dispersed across a polygonal environment, with no prior knowledge of one another and no capacity to communicate with other agents unless they are within line of sight. Until all agents have been found and gathered, each agent slowly wanders around its surroundings, attempting to identify and meet with other agents and cooperate. They will be able to converse and plan future tasks after they have all been gathered.

This report discusses previous works regarding the feasibility and complexity of gathering and provides a few potential algorithms for gathering agents in a polygonal environment. Each algorithm has its level of complexity which leads to different ways of exploration and merging; exploration uses randomness to some extent for movement, and agents already together will use amoeba movement, which is coordinated movement through polygon preserving the agent's connection with the cluster as well as maximizing exploration factor. Merging is done when two different clusters discover each other.

A simulation framework is required to test these randomized algorithms and optimize them for efficient problem-solving. The framework and all the required packages and functionalities to simulate a distributed computing problem solved by mobile agents is provided further in the report, including six algorithms for randomized gathering with different level of complexities in each of its phases and different underlying methods to show how it affects efficiency and consistency in the rounds taken by the mobile agents to solve the problem of gathering.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

With an increase in the use of Robots and other automated mobile agents such as delivery drones and surveillance robots, we require algorithms for performing various tasks such as information spreading, surveillance, leader election, path-finding, coordinated movement, gathering, and many more tasks.

1.1 Distributed system

This report studies system composed of multiple autonomous computing entities. The same algorithm is executed locally by each entity of the system. Locality refers to each agent's knowledge, and it is limited to what it can see or what is nearby. Such systems are called distributed.

Distributed systems is the branch of computer science dedicated to their study. Cooperation between entities in constrained environments is the challenge in the latter. There are two reasons for studying such systems.

First, assigning some tasks to a group of entities rather than a single entity may result in several improvements in terms of complexity and fault tolerance, even if the replacements are weaker. Once the task is divided between the entities, the time required to complete it may be reduced. Furthermore, the larger the team, the less impact a single entity's failure will have on the task's success. A task assigned to only one entity, in particular, may fail as soon as it is subject to any fault.

1.2 Research and Algorithms

This report studies distributed computing in mobile agents in geometric spaces and graphs. While looking at a few related problems and research papers and discovering an interesting unsolved problem, which is the gathering of mobile agents in a polygon environment.

More research papers were studied on gathering agents in different spaces and different models. A good grasp of the problem statement was achieved with a good idea of where to start.

Studied the implications of such an algorithm in real-life scenarios and decided on a model to work with. Following this, many weeks were spent defining the environment and metrics that could help analyze the algorithms.

Brainstormed a few algorithm ideas and figured out which ones were feasible. Many candidate algorithms were either too complex and had too many phases or were just realized as not the solution. Some made it through and were used as potential algorithms for further analysis. This report discusses those algorithms.

1.3 Simulation and Implementation of Algorithms

In the study of randomized algorithms on mobile agents and their interaction, it is crucial to understand their behavior in the 2D environment. One easy way of doing it is through simulations.

To build a simulation framework for a distributed system with environmental aspects such as polygon and mobile agents interacting, we need two main components: the simulation program, which takes in the polygon and agents as objects and their initial positions within the polygon and outputs the movement of agents as the rounds progress, and the Visualizer, which shows the activity of agents in the polygon in each round based on the output of simulation program in a video format.

Each candidate algorithm is simulated numerous times on simulation software that provides the data to study and compare. This report studies six algorithms with slight variations among themselves and shows how the variations affect their performance. The simulation software can be used for more potential algorithms as it is flexible and easy to use.

1.4 Organization of Report

The rest of the report is organized as follows: chapter 2 describes the Model architecture and all the components and resources available to the mobile agents according to the model, Chapter 3 gives a brief introduction to the domain of distributed computing and studies done on distributed computing in mobile agents in both geometric spaces and graphs. Chapter 4 discusses a few potential algorithms for the Gathering of mobile agents in a polygonal space. The discussion of simulation begins with Chapter 5, which includes the simulation framework and all its requirements to simulate distributed computing in mobile agents and their movement. Chapter 6 discusses different gathering algorithms and their results, followed by a summary and conclusion.

CHAPTER 2

Distributed computing model and preliminaries

This section discusses the distributed computing model and the environment targeted for the proposed algorithms, as-well-as as a few preliminary concepts that will better help understand the problem and its requirements.

2.1 Details of the polygonal environment

The tasks performed by mobile agents need different algorithms based on the environment they are deployed in, and each algorithm is tailored specifically for a unique kind of environment. Here we discuss algorithms for Gathering mobile agents in a 2-dimensional polygon with finite straight edges and no holes. The medial axis of this polygon must form a tree.

The polygon is considered to have n vertices, and the polygon will also have n edges since it is closed.

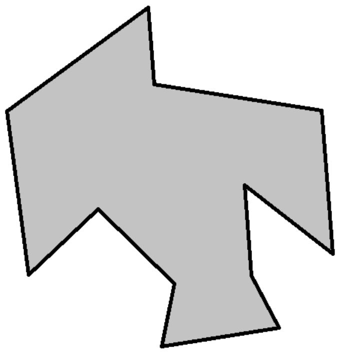


Figure 2.1: polygon

Fig. 2.1 is an example of a polygon with 12 vertices.

2.2 Model

Distributed systems follow many models, and each model has its own set of algorithms for each problem following the rules imposed by the model.

Congest model is a kind of synchronous distributed computing model, where the process is followed in rounds. Each round has phases, and the agents scan their surroundings, run the algorithm, send messages to agents in their line-of-sight, and move one step within the enclosed polygon. This report follows a solution for gathering mobile agents in a model inspired from and similar to congest model, it is synchronized and each round has same number of phases and in the same orientation

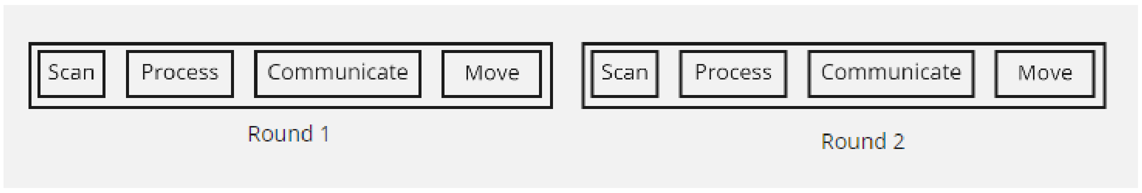


Figure 2.2: Rounds in a the model

The Fig. 2.2 shows how a typical round in synchronized round based model looks like, but if an algorithm requires more coordination than what the above phase layout can provide to the agents, it can use models with more added phases. Further in the report, we will discuss algorithms requiring multiple communication and process phases to accomplish cluster movement which aids in gathering.

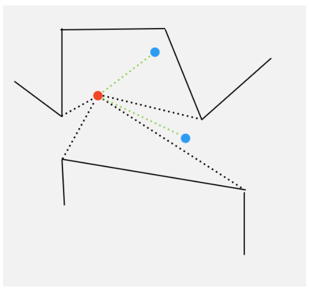


Figure 2.3: Scan phase

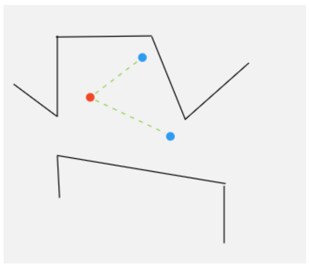


Figure 2.4: Communicate phase

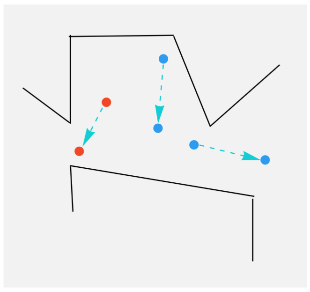


Figure 2.5: Move phase

2.2.1 Scan phase

In the scan phase, the agent scans its environment and notes down all the edges and walls of the polygon visible from its location and the location of any nearby agents that are visible.

Essentially it is generating a visibility polygon of its surroundings. This visibility polygon consists of visible walls of polygons and also cuts in polygon beyond which the agent cannot see. It also has other agents that are present within this visibility polygon. This visibility polygon is passed into the processing phase.

The following Fig. 2.7 shows a visibility polygon generated by the red agent, the grey area is the visibility polygon, and the yellow area is the area it cannot see and is not in the visibility polygon.

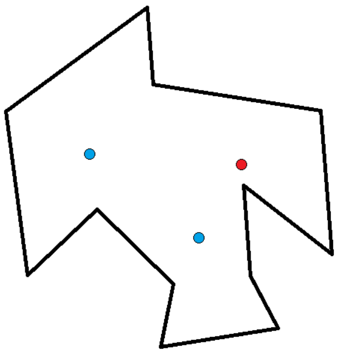


Figure 2.6: polygon with agents

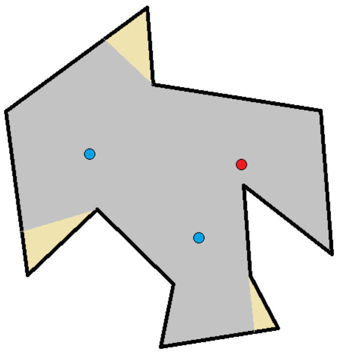


Figure 2.7: visibility polygon

2.2.2 Process phase

Followed by the process phase, here each agent uses the information gained in the scan phase as well as any information stored in memory from previous rounds to compute messages that need to be sent to other visible agents as well as determine where to move next; also any information needed for further rounds is stored in memory.

2.2.3 Communicate phase

Agents communicate necessary information among each other for the completion of the task, and this is done synchronously after processing. This phase is skipped in many algorithms as the location of other agents is enough information and no more communication is needed.

2.2.4 Move phase

The final step of the round is to move the agent to a new location in the polygon. The range of movement is limited to the region visible in the scan phase. The model does not limit how far an agent can move.

2.3 Sensors available to the agents

Each agent has the following sensors.

2.3.1 Omnidirectional radar

This radar provides information of direction and distance of nearby agents and boundaries of polygon visible from the agent's position, so this radar has depth perception. Radar helps the agent build a visibility polygon and plan out its next move. The agent can also use this radar to scan all the visible agents and ID them, this is possible by using a radar similar to Air Traffic Control Radar Beacon System (ATCRBS), which has the capability to ID the other agents if directly in line-of-sight. This radar can only be used in the Scan phase of the round, and no new information can be gathered using this radar in other phases.

2.3.2 Wireless Communication

They are used for passing messages among co-located agents in the communication phase.

2.4 Communication

As discussed earlier, the model imposes a phase of communication in each round which can be further characterized into the following types:

2.4.1 No communication, only spotting

In this kind of communication, agents do not pass messages. Instead, they use information about their environment and the location of nearby agents to decide on where to move next.

2.4.2 Line-of-sight communication

Agents communicate with other visible agents through radar(line-of-sight); an agent here can also choose to broadcast its message.

2.4.3 Bounded-range communication

Similar to Line-of-sight, we bound the range at which the agent can see or communicate with other agents. This type of communication is not studied or used in this report.

CHAPTER 3

Related work and background

The gathering of mobile agents is a well-studied problem in other models and environments but not in the closed polygonal environment with synchronized model like the one we are interested in. Each model and environment cannot be compared directly; design and complexity of algorithms dramatically change based on the environment and model being adopted. Reading research and work on similar problems will help us analyze the algorithms and give us an insight on how to approach the problem efficiently.

3.1 Similar algorithms in graphs

Gathering algorithms for graphs are usually less complicated and use memory and round number for execution. Furthermore, since their environments only consist of nodes and edges, the algorithms can also address byzantine agents.

3.1.1 Deterministic gathering in graphs

Algorithms for problems such as Strong Rendezvous in graphs with finite nodes and the Asynchronous approach in the plane and many algorithms of interest are well-documented by Sébastien Bouchard [3]. His paper studies and provides deterministic algorithms for gathering in graphs that also address byzantine failures. Hence if a polygon with no holes can be converted to a tree using the medial axis, the problem can be solved using slight tweaks to the algorithm such that the agent travels along the medial axis and complexity can be similarly analyzed. However, since agents use visual data and have no prior knowledge of the polygon, this approach falls apart as we need a medial axis. An algorithm might also better exploit two-dimensional polygon features to travel and explore quickly without considering the medial axis.

3.1.2 Deterministic gathering in trees

For comparison, we can study the deterministic Gathering algorithm for trees. Anonymous agents are capable of Gathering in asynchronous trees as shown in [2], which maps the tree by completing a DFS, following which the agent calculates the center of the tree; all agents do this task and move to the center of the tree. This algorithm gives us a base algorithm to compare the number of rounds required for agents to gather in polygonal space by using the polygon's properties instead of using the polygon's medial axis as a tree and gathering to the center of the tree. Since the medial axis is difficult to compute, A decent substitute would be by triangulating the polygon as shown in 3.1 and connecting in-centers of adjacent triangles, the in-centers form nodes of the tree.

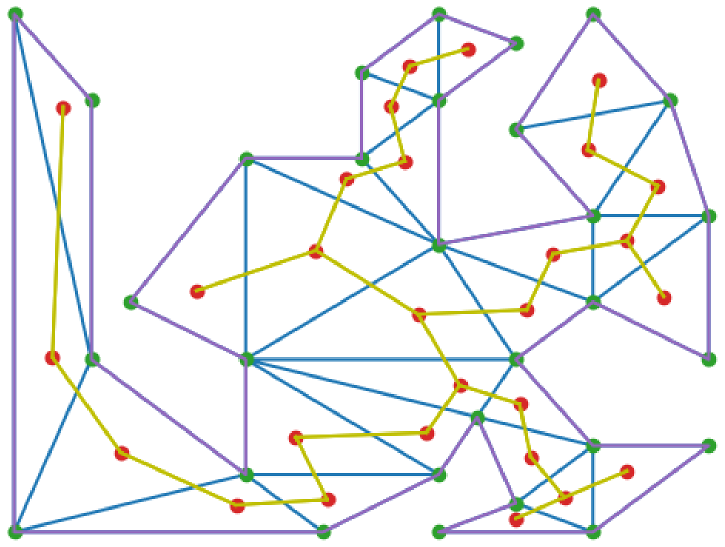


Figure 3.1: Triangulation of the polygon

Two connected in-centers are always visible through the polygon; this is the case as each triangle sharing a side with another will have the line connecting the in-centers of these triangles passing through the common side. The proof for it is the following.

Proof of line connecting in-centers of two triangles with a common side passes through the common side

Let there be two triangles ABC and CBD. Both triangles have a common side BC, both points A and D are on opposite sides of line passing through B and C. Let M be the in-center of triangle ABC and N be the in-center of triangle CBD. Both M and N lie on opposite sides of BC since the triangles do not have any common subsection and the in-centers must lie within the triangles as shown in diagram 3.2.

Since in-center is point perpendicularly equidistant from each side, and can form a circle that has the sides of the triangle as 3 tangents of that circle. Draw perpendicular line of size radius from in-center to each side of the triangle 3.3.

Now take the common side BC and the in-centers X and Y into consideration, and perpendicular lines from in-center to BC 3.4.

Now axis align the common side to Y axis such that C is $(0, 0)$ and B is some $(0, yB)$, as shown in 3.5, let the in-center be M be (xM, yM) and N be (xN, yN), and since we know N and M have perpendicular lines to the line BC connecting at some $(0, yN)$ and $(0, yM)$ and both N and M lie on opposite side of the BC, we can say $yN <= yB, yN >= 0, yM <= yB, yM >= 0, xM <= 0$ and $xN >= 0$.

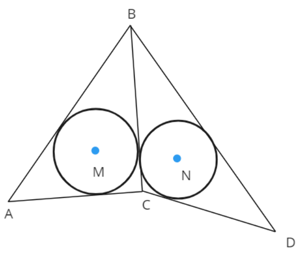


Figure 3.2: Two in-centers

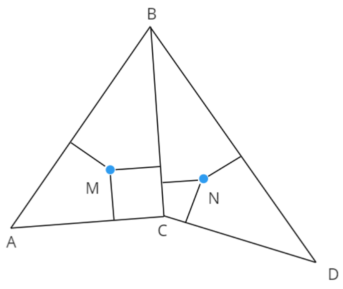


Figure 3.3: Perpendicular lines to sides

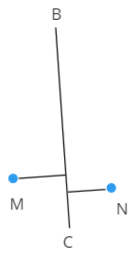


Figure 3.4: Isolate

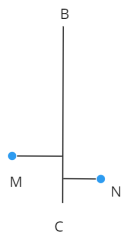


Figure 3.5: Axis align

The line connecting M (xM, yM) and N (xN, yN) is

$$(y - yN) = ((yM - yN)/(xM - xN))(x - xN)$$

This line passes through y axis at $(0, (yNxM - yMxN)/(xM - xN))$ shown in 3.6

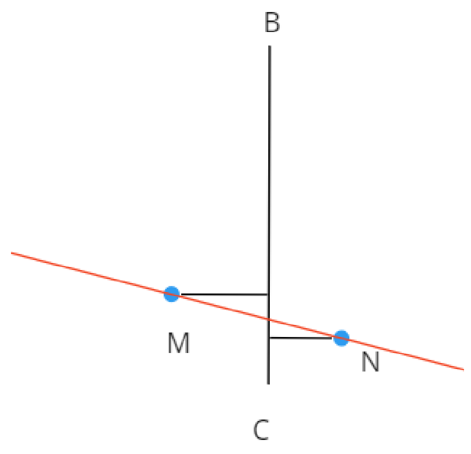


Figure 3.6: line connecting in-centers

To prove the line connecting in-centers passes through BC we need to prove

$$0 <= (yNxM - yMxN)/(xM - xN) <= yB$$

Find derivative $(yNxM - yMxN)/(xM - xN)$ with respect to yN

$df/dyN = xM/(xM - xN)$ constantly positive derivative, so higher the value of yN higher the y coordinate of intersection

Find derivative $(yNxM - yMxN)/(xM - xN)$ with respect to yM.

$df/dyN = -xN/(xM - xN)$ constantly positive derivative, so higher the value of of yM higher the y coordinate of intersection.

Replacing yM and yN with yB which is their max value gives us Max value of y coordinate which is yB and replacing yM and yN with 0 which is their Min value gives us Min value of y coordinate which is 0.

Hence Proved $0 <= (yNxM - yMxN)/(xM - xN) <= yB$ And therefore proved the line connecting the in-centers passes through the common side of the two triangles.

3.1.3 Efficient algorithms for mobile agents

Some of the fundamental features to have in an algorithm is completeness and efficiency, and there are well studied efficient algorithms for sub-problems of gathering mobile agents in a graph such as Exploration of a graph, map construction, and rendezvous problem provided by Adrian Kosowski [6].

Since polygons can have intricate shapes, Map construction might require memory up to $O(|V|)$, where V is the number of edges in the polygon, and algorithms discussed in this report use randomness to move and use no prior information about the environment and neither record it during movement so there is no need for map construction.

Rendezvous of mobile agents in a graph is of interest since there are many different kinds of walks an agent can take that can significantly affect the round complexity of the algorithm.

As seen from previous works [6] Table 2.1, we know random walks in graphs takes $O(V^2 \log V)$ or $O(ED \log V)$ time complexity to cover an entire graph, D is the diameter of graph and metropolis walk takes $O(V^3)$ or $O(ED \log V)$ time complexity to cover an entire graph,V is number of vertices is graph, E is number of edges and D is the diameter of the graph.

Studying more about these kinds of random walks will better help us decide the most optimal movement for a cluster, which will result in a time-efficient gathering.

Few algorithms proposed in this report lean towards the idea of biased random walk for leaders (Agent moves with high probability towards front and lower towards the back).

3.2 Similar algorithms in polygons

Studying similar algorithms for mobile agents in polygon provides the necessary introduction to the domain, the necessary pathway to build an efficient algorithm, and to analyze its performance.

3.2.1 Guarding a polygon

Guarding a polygon, an algorithm for a classic computational geometry problem called *Art gallery problem* from a mobile agent's perspective, seeks to minimize the number of guards required to guard an art gallery problem, demonstrated in [1].

It shows the minimum number of agents required to guard the polygon while staying connected. The paper [1] also talks about the navigation of mobile agents from a single starting point to achieve this complete guarding. This shows that having enough agents in a cluster to guard polygon completely will result in complete Gathering in $O(n)$ time. However, since it requires $O(n)$ agents, which might not be feasible, this approach is not used, but navigation of agents from a single point to maximize the visibility of the cluster in a polygon is valuable and can be considered for exploration of clusters.

3.2.2 Gathering of disoriented mobile agents on a plane

The Gathering is an easy problem to solve if all agents are present on the Cartesian plane and can see each other, but there are algorithms with added restrictions such as restriction on the range of vision or memory of an agent [4]. These algorithms help our Gathering algorithm during phases when multiple agents belonging to different clusters spot each other in the same round and provide us with points on the plane that are best suited for these agents to gather such that the agent needs to travel the least distance.

3.2.3 Visibility coverage for a polygon with holes

This is a similar $\mathit{Artgalleryproblem}$ seen earlier but has added complexity of holes in the polygon, but can only operate on non-convex polygons [7].

The agents begin deployment from a fixed point and operate without prior knowledge of the environment and only under line-of-sight sensing and communication. The agents, once finished deployment achieve full visibility coverage of the polygon while maintaining line-of-sight connectivity with each other.

Since in this report we only consider polygons without holes, the algorithms are tailored only to such polygons, but if we need an improved algorithm for handling holes in the polygonal environment, visiting these algorithms would be a good place to start.

3.2.4 Flocking control

Mobile agents such as drones are robots that have physical hardware with moving parts that are sensitive and should not come in contact with any object while being operated, not even other mobile agents. There are well-studied flock control algorithms to achieve moving in a cluster while maintaining a minimum distance between each agent.

These algorithms ensure the agents are a minimum of a certain distance away from each other and move in a formation representing a hexagonal close-packed structure. The agents here only move on a two-dimensional plane and communicate via line-of-sight communication.

Algorithms of flock control are added complexity that can be used if necessary; the algorithms discussed in this report assume mobile agents are point-sized objects.

3.3 Similar algorithms in different models

Similar algorithms but in different models or set in a different environment with a different set of rules can be closely related to the gathering algorithm or sub-problems of it.

3.3.1 Single agent exploration

There are algorithms for single-agent exploration of a polygon in the least amount of time or least amount of steps [5]. These algorithms can help a single agent have maximum visibility coverage in the least amount of time.

However, algorithms covered in this topic are deterministic and use sensors while moving, and our model has separate moving and scanning phases. These algorithms also have a very high time complexity directly proportional to the number of vertices in the polygon. If there is to be a deterministic algorithm designed for gathering problems, these path planning algorithms could be a good place to start, as they provide an algorithm and analysis that will also closely relate to the analysis of gathering.

3.3.2 Amoeba movement

Amoeba movement refers to the randomized movement of mobile agents in a cluster without losing line-of-sight connectivity. Problems like these have not received much attention in robotics communities, and for us, it seems like it is the core problem that can solve gathering. What we do have is the cluster movement algorithm which was discussed earlier [8] of agents on a plane without touching each other. Algorithms like these on polygons are not studied. We need a time-efficient amoeba movement algorithm that explores the polygon in the least amount of time while also merging with any other cluster found on its way.

Since the movement is randomized, the upper bound of the gathering algorithm depends on how fast a single cluster can explore the entire polygon.

We will discuss a few amoeba movement algorithms at a high-level idea in the next section that gives us a better understanding of how they work and how they affect the exploration factor.

CHAPTER 4

Candidate Algorithms

To define the execution of an algorithm by mobile agents, we need two kinds of elements. On the one hand, there is the theatre stage, the actors, and the play, which is to say, the environment, the mobile agents, and the algorithm they run. On the other hand, deterministic rules govern the effects of an algorithm's instructions on the environment and mobile agents. It is possible to determine the system's state at any time given the environment, the algorithm, and these rules.

However, we use randomness for all the algorithms discussed below. Each algorithm attempts to form clusters of agents that have discovered each other until one cluster remains with all agents.

This section contains four algorithms, each with a different level of complexity in terms of movement, exploration, and merging, all of which are explained in terms of their high-level idea.

4.1 Leader-followers algorithm

A simple algorithm where each cluster has a leader, and the followers follow the leader. The leader uses biased random walk to explore the polygonal environment. All mobile agents are initially leaders and then merge into clusters as the rounds follow, the cluster grows, and eventually, all the agents are found to be in the same cluster.

Each round in this algorithm consists of three phases: scan, process, and Move. The phases occur in that order. During the scan phase, all agents broadcast their IDs to other agents while noting down the IDs of agents they can see and their positions. No message is passed among them. Processing is done, and they move.

4.1.1 Movement

The leader of a cluster is free to move without concerning where other agents of its cluster are. It uses biased random walk to explore. The colored red in the Fig. 4.1 is the leader; the remaining blue agents are follower agents.

The follower agents move to the agent's position that they are following from the previous round. Every follower agent follows some other agent, and it can either be the leader or another follower agent

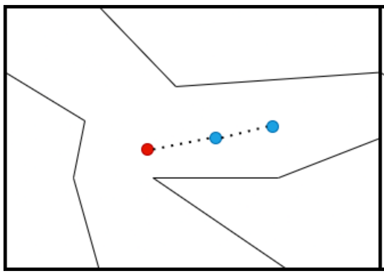


Figure 4.1: Round x

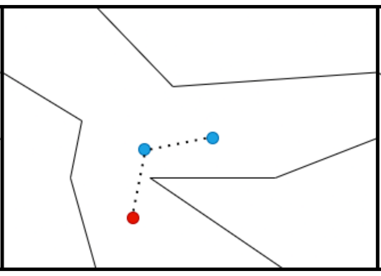


Figure 4.2: Round y > x

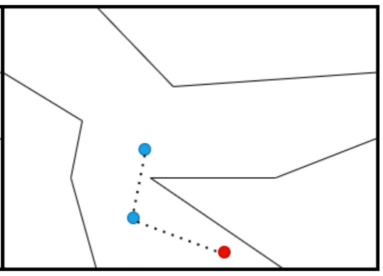


Figure 4.3: Round z > y

4.1.2 Exploration

The leader solely does the task of exploration, the remaining agents in the cluster are followers and do not partake in exploration of the polygon themselves, but since they trail behind the leader, they can be spotted by leaders of other clusters and can be followed.

4.1.3 Merging

Each agent is initially a leader and a cluster itself, and they merge by simply one cluster following another cluster. Two possible collisions could occur, one is if two leaders spot each other, then a leader is chosen among them, the chosen leader continues, and the other leader becomes a follower and follows the chosen leader, the second case is when a leader spots a follower from another cluster, here the leader becomes a follower and follows the spotted follower from other cluster.

In the Fig. 4.5, we can see the leader of the bottom cluster spots a follower in the top cluster and decides to follow it, and changes itself to a follower.

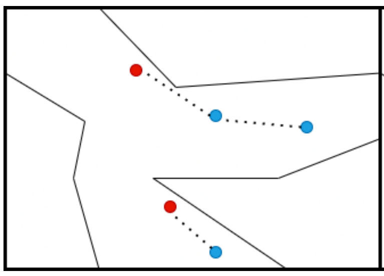


Figure 4.4: Round x

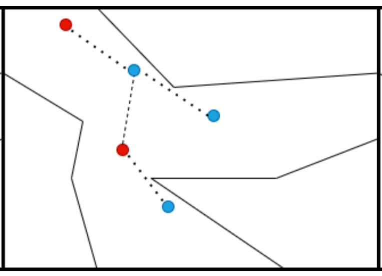


Figure 4.5: Round y > x

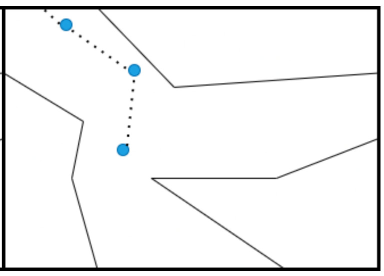


Figure 4.6: Round z > y

4.2 Leader, Guards and explorers algorithm

This algorithm uses what is called the amoeba movement to explore the polygon. It has a better exploration factor compared to the Leader-follower algorithm since all agents participate in exploration as opposed to one, but it has much more complex movement and merging phases, which also requires coordination and leads to more phases added to each round.

Each cluster has a leader, a few guards, and explorers. All the agents participate in exploration, and only the leader dictates the movement of the cluster, and guards provide a better reach for exploring agents.

The rounds in this algorithm are much more complex and require one scan phase at the beginning followed by one process phase for a leader to decide its next step, followed by communicating phase so that leader can communicate its next step with the guards followed by another process phase for guards to decide their step. Then the last communicate phase for guards to send their next step to the explorers, explorers decide their next step, and finally, all agents have their next step decided and move together in the final phase, which is the move phase.

In the Fig. 4.7, red-colored agents are leaders, yellow-colored agents are guards, and the remaining blue-colored ones are explorers.

4.2.1 Movement

The leader takes short steps and updates the guards about its next step, the guards then use this information to decide their next step, both the current position of leader and next position of the leader must be visible from the next position of the guard, the guards are always less than a certain distance away from the leader, each explorer is assigned a guard and receives information about guard's next step and takes its step accordingly such that it can see the guard in the next step. The guard and the explorer choose their successive positions randomly from the visible polygon.

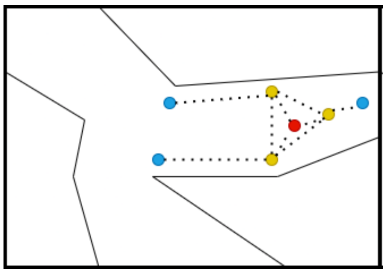


Figure 4.7: Round x

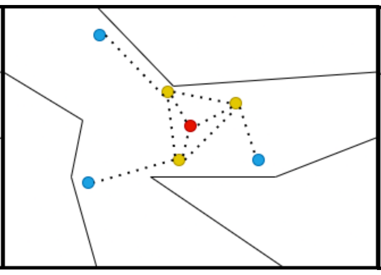


Figure 4.8: Round y > x

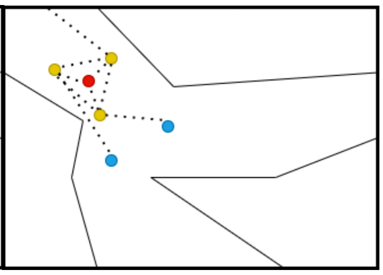


Figure 4.9: Round z > y

4.2.2 Exploration

All agents participate in exploration. The exploration factor gets higher as we add more agents to the cluster.

As soon as an agent from a different cluster is observed, the leader is passed the information.

4.2.3 Merging

An agent from a cluster spots an agent from another cluster; it passes this information in the same round to the leader. Following this, the leaders of each cluster freeze for one round when they receive such information. This is enough to notify guards and their explorers that a new cluster has been found.

Following this, all cluster agents move to the position of the agent of their respective clusters that found another cluster, a new leader is chosen. The newly chosen leader assigns roles, and the leader starts moving again.

Each agent initially starts as an explorer and moves freely without being bound to any guard; when another cluster is found, it acts as a leader of its cluster.

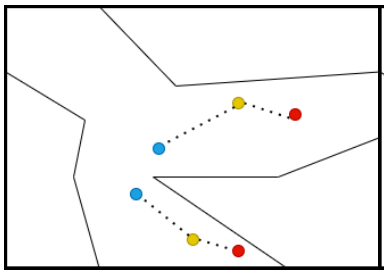


Figure 4.10: Round x

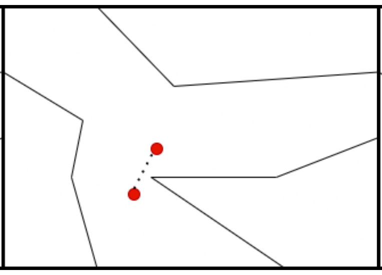


Figure 4.11: Round y > x

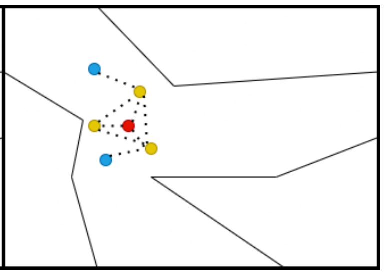


Figure 4.12: Round z > y

4.3 Leader with explicit tentacles algorithm

The leader with explicit tentacles uses guards as nodes in a tentacle and explorers as its tip. The maximum length of the tentacles is a hyperparameter and can be changed. The leader and explorers behave the same way in this algorithm as in the leader, guards, and explorer algorithm. Guards are tasked to keep the explorers connected to the leader.

This is an improvement on the leader, guards, and explorers algorithm; here, we increase the number of guards to improve the reach of explorers, but this comes at the cost of the time taken for each round. The time taken for each round is directly proportional to the length of the tentacles.

As discussed in the previous algorithm (Leader, guards and explorers), the position of the cluster gets updated from the leader and moves out towards the explorers, the agents arrange themselves as tentacles around the leader, and the number of phases required for each round is directly proportional to the length of the tentacle.

Red-colored agents are leaders in the Fig. 4.13, yellow-colored agents are guards, and the remaining blue ones are explorers.

4.3.1 Movement

The leader agent takes small steps and moves without concerning itself about the position of agents in the cluster.

The information about the leader's next step is relayed down the tentacle. Each node (guard) in the tentacle uses this information to decide its position in the next round, except for the first guard, the first guard from the leader ensures the current, as well as next step of leader, are visible from the next step it takes, the relayed information of guards ultimately reaches the explorer before the end of the round, and the entire cluster moves together.

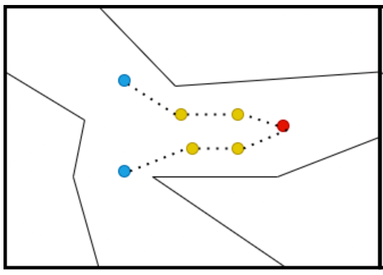


Figure 4.13: Round x

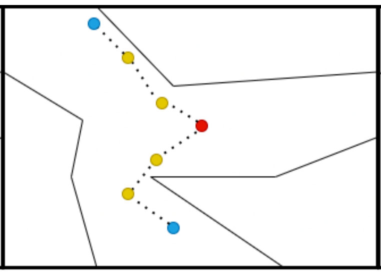


Figure 4.14: Round y > x

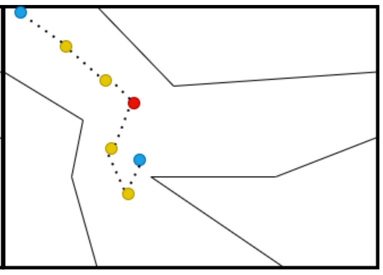


Figure 4.15: Round z > y

4.3.2 Exploration

All the agents in the cluster participate in exploration, and any agent spots an agent from another cluster, relays the information up and down the tentacle. When the information reaches the leader, it freezes for one round, notifying all the tentacles that a new cluster is found.

4.3.3 Merging

Merging is similar to the leader, guards, and explorers algorithm. The cluster moves towards the agent in their respective cluster that found the other cluster. A new leader is elected, and the leader assigns new roles.

4.4 Leader with implicit tentacles

Up until now, the focus was on keeping the cluster connected to avoid losing an agent during exploration, and all agents kept no memory of previous rounds. However, if memory is available, each agent in a cluster can participate in exploration without staying connected to the leader. This increases the range of exploration since an agent does not need to worry about being scattered from an already gathered cluster.

The implicit tentacles in the algorithm's name imply that each node acts as a tentacle and moves away from a previously decided location.

In the Fig. 4.16, we see two clusters, each with different starting points; two agents colored blue from different clusters spot each other and return to their original position, marked in red. In this algorithm, the leader is only responsible for deciding which cluster to move towards and is not a needed entity for this algorithm to work. But in case that multiple clusters are found, having a leader makes it easier to choose which cluster to merge with.

Each round in this algorithm requires as many phases as we saw in the leader-follower algorithm.

4.4.1 Movement

All cluster agents start from a location (marked in red on the figure); they all leave this position but remember their steps. This requires memory.

Each agent takes k steps for exploration, and k steps back to the original position. The memory is required to store these k steps as the agent needs to backtrack. If an agent finds an agent from another cluster, it notifies the leader when it has backtracked.

If no new cluster is found, the cluster moves to the new location and repeats again.

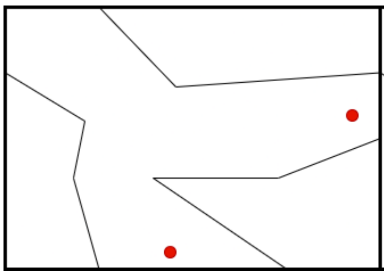


Figure 4.16: Round x

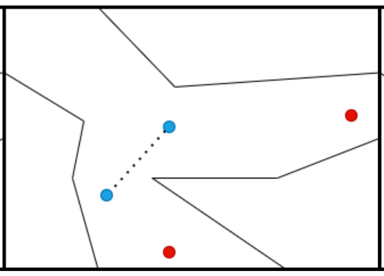


Figure 4.17: Round y > x

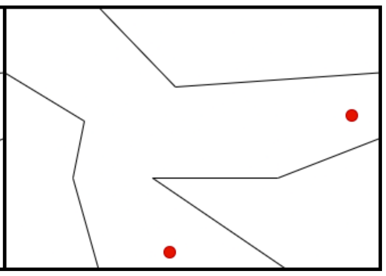


Figure 4.18: Round z > y

4.4.2 Exploration

Each agent participates in exploration; since the time taken for each round is shorter compared to other previously discussed algorithms, it can take many steps in a short amount of time. After each step, the agent broadcasts its cluster's leader's ID and scans if it can see any agent broadcasting a different ID. Suppose a different ID is observed; the agent backtracks to the original position and reports this information to the leader.

4.4.3 Merging

If an agent discovers a new cluster during the exploration phase, it provides the location where it found the other cluster to the leader. The cluster then moves to that location, merges with the new cluster, and chooses a new leader. Then they move ahead as a new cluster.

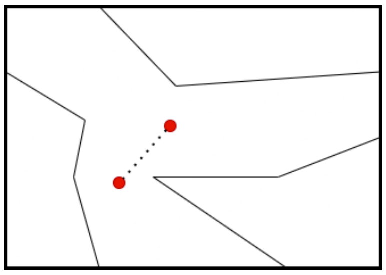


Figure 4.19: Round x

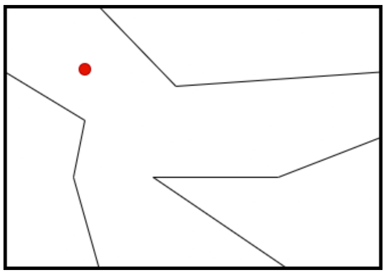
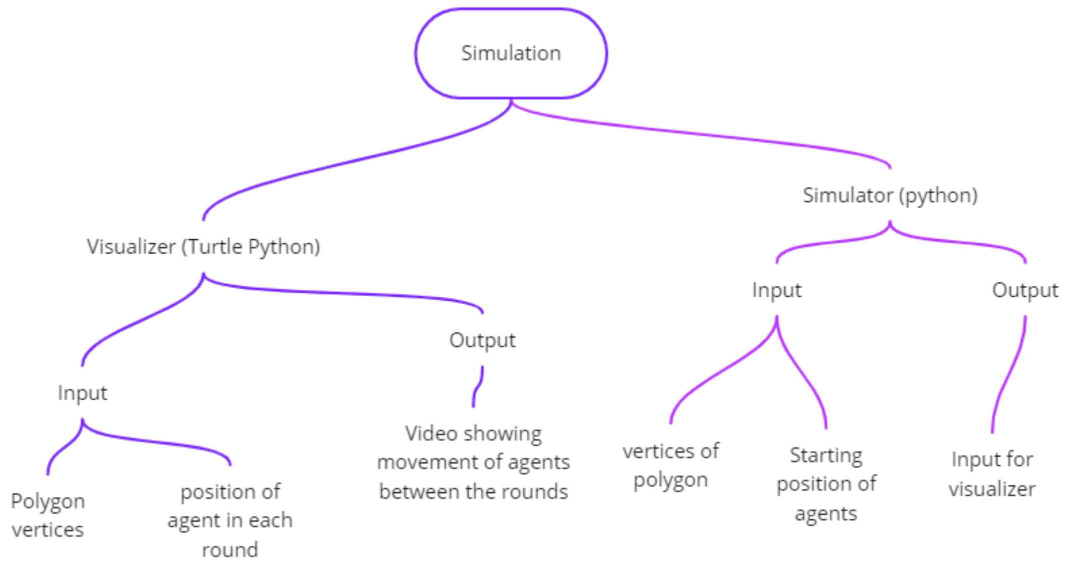


Figure 4.20: Round y > x

CHAPTER 5

Simulation Framework

The framework requires two main components the simulation program and a visualizer. The simulation program takes the polygon and the agents' placement and returns the agents' position each round. The visualizer takes these positions and shape of the polygon as an input and generates a video showing the movement of all agents in the polygon as the rounds progress.



Simulation
Visualizer (Turtle Python)
Simulator (python)
Input
Output
Input
Output
Polygon vertices
position of agent in each round
Video showing movement of agents between the rounds
vertices of polygon
Starting position of agents
Input for visualizer

Figure 5.1: Simulation framework

5.1 Simulation program

Python language and many of its libraries are used to write the simulation program. The agents are identical and individual entities with their memory and processors, and each has a unique ID. The agents are treated as objects of the same class with different IDs, to simulate this distributed aspect. Each phase of these rounds has its member function within this Agent class. The memory is initialized using member variables before the rounds begin. Algorithms discussed in this report use four different phases: look, compute, communicate, and move. The phases are called one at a time and cannot overlap. Each phase is called in a linear sequence for all agents before the next phase is called to simulate the synchronous calls of each phase. Simulating these phases is discussed in further sections.

5.1.1 Look

During the look phase/Scan phase, the agents are expected to use a 360 Degree radar similar to Air Traffic Control Radar Beacon System (ATCRBS) to scan their environments, generate a visibility polygon and detect all the agents that can be seen from this location and ID them. To simulate this phase, the visibility polygon of an agent is generated using a visibility polygon function which takes in the shape of polygon and position of the agent and uses a circular sweep method and provides the coordinates of the sub-polygon that an agent sees. To simulate spotting all the agents an agent can directly see, we check every agent's location and check if it lies within the visibility polygon; if it does, it can be seen, or else it is obstructed. The following diagram shows (5.2) 3 agents in a polygon, and the following diagram shows (5.3) the visibility polygon generated by the blue agent.

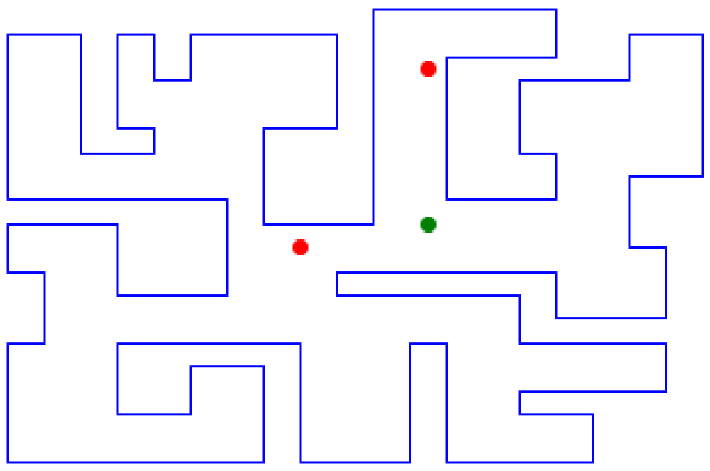


Figure 5.2: Polygon with agents

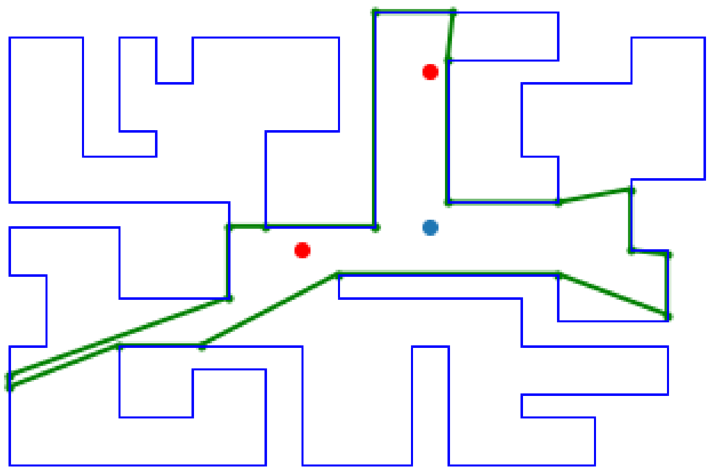


Figure 5.3: Visibility Polygon

When this function is called, the output of the look function gets stored in the memory of the agent, which is the visibility polygon and the agents directly visible to the respective agent. This function usually remains the same for all algorithms, requiring no changes.

5.1.2 Compute

Compute phase is the same as process phase mentioned earlier in Chapter 2, uses the already present data in the memory and makes decisions such as what to communicate, whom to communicate with and where to move in the upcoming Move phase. The Compute phase executes the bulk of the algorithm and is mainly responsible for the agent's behavior. Few algorithms with different compute phases and different positioning of this phase in the rounds will show how it affects the agents' performance in Chapter 6.

5.1.3 Communicate

The agents are expected to use wireless signals to communicate with each other and store the exchanged messages in a buffer, which then will be used in compute phase to make decisions. To simulate this, each agent object is given full access to the member variables of all other agent's objects; this lets each agent change information in other agents' memory or pull information from it, which says if an agent needs some information, it will directly take it from the sender's memory and store it in its memory. Furthermore, vice versa is also true; if an agent wants to give some information to another agent, it will change values in the receiver's memory itself.

Simulating/Executing the Communicate phase sequentially poses a risk of chaining memory where 1^{st} agent changes the memory of 2^{nd} agent, following which 2^{nd} agent uses the same information which was changed by 1^{st} agent and passes it on to 3^{rd} agent, this leads to information traveling from 1^{st} to 3^{rd} within 1 communicate phase which is an incorrect simulation, to avoid this lists and variables need to be used and changed in a specific manner. They should be monitored for the above-mentioned anomalies.

5.1.4 Move

In the Move phase, the agents physically travel from their current position to the position decided during the compute phase; if there is no new position, the agent stays in its current position. For simulating this, each agent maintains a member variable that stores the agent's current position. The move phase can be executed by changing the agent's current position to the next step and passing the previous current position into the list of positions traveled by the agent. If the next step is not set during the compute phase, the current position is passed into the list of locations traveled, but the current position remains.

5.2 Visualizer

The visualizer uses a turtle package provided by python to draw each frame and visualize it in a graphic format. The position of each agent in each round is fetched from the simulation program and the move phase of each round is shown by filling the intermediate steps taken by the agent to travel from one point to another. The number of intermediate steps in each round dictates how slow or fast the simulation will be.

CHAPTER 6

Algorithms and Simulation results

6.1 Two Agents Random Walk

In a simple algorithm to test the simulation program, two agents are placed in the polygon with no previous knowledge of the polygon and no information on the other agent. They are randomly walking in the polygon as the rounds progress until they see each other, following which they meet in the middle in the following round.

Neither of the agents can use memory to store their previous moves, so their movement is random, and they might revisit parts of the polygon they have already visited. Also, no communication is needed since they gather the following round once they see each other. For movement, both agents scan their surroundings and generate a visibility polygon and pick a random point within the visibility polygon to move to each round until they see each other.

6.1.1 Initialization

All agents are instances of a Class and have member variables that store small amounts of data to describe the round and its progress; firstly, we initialize these variables. Both agents are provided with the shape of the polygon; this is only used to generate visibility polygon and holds no other purpose. They are also given unique Identification numbers and a random starting point within the polygon.

They also have a list of coordinate points they have traveled to stored within the object, but this list is only extracted once the agents are gathered and is used to for visualizing their movement and does not aid them in the process of gathering. Initialization is done once before the Rounds begin and need not be done again.

Algorithm 1 Initialization of member variables (Two Agent Random Walk)

```
Input: polygon, ID
Output: None
  Agent polygon = polygon
  Agent position = Random point in polygon
  Agent ID = ID
  Agent Movement = Empty List
  Agent Movement = Agent Movement + {Agent Position}
```

6.1.2 Phases of each round

For a simple algorithm such as **Two agents random walk**, The Agent needs just 3 phases: Look, Compute and Move. Following are the algorithms for each of those phases.

Look

Agents are expected to use their sensors to scan their surroundings, form a visibility polygon, and spot the agents in the visible region. The visibility polygon function, which takes in the polygon and position of an agent as input and outputs the visibility polygon, is used to simulate this phase. Once the visibility polygon is calculated, we can check which agents are within this polygon and output that as a list along with the visibility polygon.

When the Look function is executed in a round, it sets the Agent's current visibility polygon and the list of visible agents.

Algorithm 2 Look Phase (Two Agent Random Walk)

```
Input: polygon, list of agents, Agent position
Output: None
  Agent visibility_polygon = visibility polygon generated from Agent's position
  Agent visible_agents = All the other agents within Agent visibility_ polygon
```

Compute

The next move and the information to be communicated in the upcoming phases are computed in Compute phase; for this algorithm, only coordinates of the subsequent position are calculated. If an agent can see an agent, it sets the center of their respective locations as the next step, and declares itself as gathered; otherwise, it picks a random location in its visibility polygon and sets that as the next step.

Algorithm 3 Compute Phase (Two Agent Random Walk)

```
Input: None
Output: None
  if Agent visible_agents is empty then
      Agent next_move = Pick a random point in visibility polygon
  else
      Agent next_move = Middle of current Agent and visible agent
  end if
```

Move

The Agent checks whether its next step is set or not; if it is, it sets that as its current position and adds it to its movement list. The Agent can only take one step each round and move within the polygon and move in a straight line.

Algorithm 4 Move Phase (Two Agent Random Walk)

```
Input: None
Output: None
  if Agent next move is set then
      Agent position = Agent next move
      Agent next_move = NULL
  end if
  Agent Movement = Agent Movement + {Agent Position}
```

6.1.3 Results

Following is the statistics of number of rounds required to gather in a 1000 simulations of the **Two Agent Random walk Algorithm**. In each simulation 2 agents where randomly placed in the polygon and the algorithm was executed, the number of rounds required to gather were noted.

Table 6.1: Two Agent Random Walk simulation results

Agents	Average no. of rounds	Standard deviation	90th percentile	50th percentile	MIN	MAX
2	13.89	14.233	32.0	9.0	2	105

6.1.4 Learning

The data shows high variance, and the average is higher than the 50th percentile shows the agents take a long time to gather in cases where they are placed far away from each other in the polygon as they use randomized walk and revisit sections of polygon, but most agents gather in fewer rounds if placed closer as expected.

6.2 Gathering Using IDs only (Type 0)

6.2.1 Changes from previous algorithm

The previous algorithm tests the agents' 3 phases and random walk capability. But it can only be used for the Gathering of two agents, and Gathering multiple agents just by looking at other agents and their position is difficult.

Instead of just using the position of other agents, if the agents can ID the other agents during the scan phase using radars similar to Air Traffic Control Radar Beacon System (ATCRBS) which can ID the agents using the returning radar signal, it opens up the realm for several possible Gathering algorithms. **Gathering Using IDs only** algorithm uses the IDs of other agents to form clusters of agents following each other until there is just one cluster remaining.

6.2.2 Initialization

Initialization is similar to the previous algorithm, the new member variable needed to be initialized the **Agent following**, this mentions the Agent that is being followed by the current Agent and is initialized as Null and is set when the Agent spots an agent with ID smaller than its own.

Algorithm 5 Initialization of member variables (Type 0)

```
Input: polygon, ID
Output: None
  Agent polygon = polygon
  Agent position = Random point in polygon
  Agent ID = ID
  Agent Movement = Empty List
  Agent Movement = Agent Movement + {Agent Position}
  Agent Following = Null
```

6.2.3 Phases of each round

This algorithm uses IDs of other Agents, but since IDs are scanned using sensors themselves, the agents need the same 3 phases, Look, Compute and Move, each round. No communication is required.

Look

Agents are again expected to use their sensors to scan their surroundings, form a visibility polygon, and spot the agents in the visible region. Once the visibility polygon is calculated, we can check which agents are within this polygon and output that as a list of IDs in ascending order along with the visibility polygon.

When the Look function is executed in a round, it sets the Agent's current visibility polygon and the list of visible agents.

Algorithm 6 Look Phase (Type 0)

```
Input: polygon, list of agents, Agent position
Output: None
  Agent visibility_polygon = visibility polygon generated from Agent's position
  Agent visible_agents = sorted list of other agents within visibility polygon
```

Compute

The next move and the information to be communicated in the upcoming phases are decided in Compute phase. In this algorithm, only coordinates of the subsequent position are calculated as, again, there is no communication phase involved if an agent can see one or more agents with IDs smaller than its own, it sets the **Agent following** as the Agent with most minor ID. It sets the current position of that Agent as its next step. If an agent cannot see any other agent with an ID smaller than itself, then it assumes it is a leader, and it picks a random point in its visibility polygon and sets that as the next step; it moves on every alternate round so it can let the cluster catch up to it. Since there is no communication, the agents need to check every round if all agents are present at the same spot to classify themselves as gathered; if they are gathered, they stop moving.

Algorithm 7 Compute Phase (Type 0)

```
Input: None
Output: None
  if All agents are at same location then
      Classify agent as gathered and stop moving
  else
      if Agent visible_agents is empty then
          if round number is even then
              Agent next_move = Pick a random point in visibility polygon
          else
              Agent next_move = Agent position
          end if
      else
          if Agent visible_agents[0]'s ID $<$ Agent ID then
              Agent following = Agent visible_agents[0]
              Agent next_move = Agent following's position
          end if
      end if
  end if
```

Move

The Move phase is the same as a previous algorithm; the Agent checks whether its next step is set or not; if it is, it sets that as its current position and adds the current position to its movement list. The Agent can only take one step each round and move within the polygon and move in a straight line.

Algorithm 8 Move Phase (Type 0)

```
Input: None
Output: None
  if Agent next_move is set then
      Agent position = Agent next move
      Agent next_move = NULL
  end if
  Agent Movement = Agent Movement + {Agent Position}
```

6.2.4 Results

Following are the statistics on the number of rounds required to gather for a different number of agents using **Gathering using IDs only (Type 0)**. Each configuration was simulated 100 times. In each simulation number of agents, depending upon the configuration, were randomly placed in the polygon, and the algorithm was executed; the number of rounds required to gather was noted.

Table 6.2: Gathering using IDs only (Type 0) simulation results

Agents	**Average no. of rounds**	**Standard deviation**	**90th percentile**	**MIN**	**MAX**
5	38.08	27.41	70	8	176
10	38.58	20.53	64	8	128
15	41.78	25.41	76	10	148
20	42.32	23.79	80	14	108
25	48.68	31.94	92.40	12	140
30	44.76	22.17	78	12	110
35	50.32	28.39	86	10	176
40	48.9	29.47	86.20	12	184
45	49.44	25.68	86	12	150
50	45.02	29.07	78	12	170
55	46.42	26.90	84	8	142
60	48.32	32.37	108.20	10	166
65	48.66	30.77	76.20	14	240
70	46.16	32.84	84.60	10	208
75	46.26	27.55	80.40	12	150
80	44.94	25.86	84.20	12	136
85	45.82	29.51	88	8	182
90	48.1	29.74	86	10	224
95	50.44	31.51	90.60	12	198
100	46.94	25.40	74	14	150

6.2.5 Learning

This algorithm completes Gathering when all the agents in the polygon have seen the Agent with the lowest ID; hence the number of rounds required to gather using this algorithm can be treated as the number of rounds required for the farthest Agent from lowest ID (Leader) agent to spot the leader as an upper bound. As the number of agents in the polygon increases, the Agent furthest away from the Agent with the lowest ID increases until it can no longer rise due to constraints of the polygon. Since there is no collaboration/communication among agents and the leading Agent stops every alternate round in this algorithm, the number of rounds required to gather is high and has very high variance. The average number of rounds required to gather rises gradually as the number of agents randomly placed in the polygon rises, but after a certain threshold, it stops rising as the Agent furthest away from the lowest ID agent gets constrained by the polygon.

This algorithm clearly shows there is room for improvement in terms of performance. It can be done by adding some level of collaboration among the agents. Such an algorithm is discussed next.

6.3 Gathering Using Train of Followers (Type 1)

6.3.1 Changes from the previous algorithm

The previous algorithm used no communication phase or communicated only their IDs, limiting their collaboration ability. By adding communication among agents that can see each other, there is a significant boost in performance, and the agents need not closely be clustered anymore and can branch out and look for other clusters.

In the previous algorithm, the leader needed to stop every alternate round so the cluster could catch up to it, but in this algorithm, agents use the communication phase to pass IDs of follower agents to the leader so the leader at all times knows if all the agents within the polygon are following it or not and it need not stop. Once the leader realizes all agents are following it, it stops moving, which lets the cluster merge at a single spot.

6.3.2 Initialization

Initialization is similar to the previous algorithm; the new member variable needed to be initialized the **Agent followers**; this mentions the list of agents following this particular Agent, and the contents of this list are passed on to the **Agent follower** (Agent that the current Agent is following) if it is set.

Algorithm 9 Initialization of member variables (Type 1)

```
Input: polygon, ID
Output: None
  Agent polygon = polygon
  Agent position = Random point in polygon
  Agent ID = ID
  Agent Movement = Empty List
  Agent Movement = Agent Movement + {Agent Position}
  Agent Following = Null
  Agent Followers = Empty list
  Agent Followers = Agent Followers + {Agent ID}
```

6.3.3 Phases of each round

This algorithm uses IDs of other Agents as well as communicates its followers to **Agent following**, it requires four phases which are in order: Look, Compute, Communicate and Move.

Look

Agents are again expected to use their sensors to scan their surroundings, form a visibility polygon, and spot the agents in the visible region. Once the visibility polygon is calculated, we can check which agents are within this polygon and output that as a list of IDs in ascending order along with the visibility polygon.

When the Look function is executed in a round, it sets the Agent's current visibility polygon and the list of visible agents. Nothing changes for the Look phase compared to the previous algorithm.

Algorithm 10 Look Phase (Type 1)

```
Input: polygon, list of agents, Agent position
Output: None
  Agent visibility_polygon = visibility polygon generated from Agent's position
  Agent visible_agents = sorted list of other agents within Agent visibility_polygon
```

Compute

The agents need to check every round if all agents are present at the same spot to classify themselves as gathered; if they are gathered, they stop moving. If they have not gathered, they continue the similar algorithm as in the previous algorithm's Compute phase, and instead of non-followers stopping every alternate round, they move randomly every round. The non-followers start following others only when they spot another agent with a smaller ID than their own; once they do, they pick the one with the smallest ID as **Agent following**, and once they start following an agent, they do not swap this following Agent. The other difference being a non-follower agent needs to check **Agent followers** to see if the size has reached the total agent count in the polygon; if it has, that means it is leading the cluster with all the agents, and it stops moving, which lets all the agents in the cluster merge at the non-followers (leader) location.

Algorithm 11 Compute Phase (Type 1)

```
Input: None
Output: None
  if All agents are at same location then
      Classify agent as gathered and stop moving
  else
      if Agent following is set then
          Agent next_move = Agent following's position
      else
          if Agent followers list has all agents then
              stop moving the agent
          else
              if Agent visible agents[0]'s ID $<$ Agent ID then
                  Agent following = Agent visible agents[0]
                  Agent next_move = Agent following's position
              else
                  Agent next_move = Pick a random point in visibility polygon
              end if
          end if
      end if
  end if
```

Communicate

A simple way of simulating communication between two agents is by simply giving one Agent the authority to alter the memory of another agent. For this algorithm, the trailing Agent takes the union of **Agent followers** from its memory and its leading Agent, unions it, and sets the new list as **Agent followers** in the leader agent's memory.

Algorithm 12 Communicate Phase (Type 1)

```
if Agent following is set then
    Agent following's followers= Agent following's followers $\bigcup$ Agent followers
end if
```

Move

The Move phase is the same as the previous algorithm; the Agent checks whether its next step is set or not; if it is, it sets that as its current position and adds the current position to its movement list. The Agent can only take one step each round and move within the polygon and move in a straight line.

Algorithm 13 Move Phase (Type 1)

```
Input: None
Output: None
if Agent next_move is set then
    Agent position = Agent next move
    Agent next_move = NULL
end if
Agent Movement = Agent Movement + {Agent Position}
```

6.3.4 Results

Similar to results in the previous algorithm, the following are the statistics of the number of rounds required to gather for the different number of agents using **Gathering using Train of Followers (Type 1)**; each configuration was simulated a 100 times. In each simulation number of agents, depending upon the configuration, were randomly placed in the polygon, and the algorithm was executed; the number of rounds required to gather was noted.

Table 6.3: Gathering Using Train of Followers (Type 1) simulation result

Agents	**AVERAGE**	**STDEV**	**MAX**	**MIN**	**90th Percentile**
5	20.44	13.91	87	5	36.40
10	24.62	14.07	69	6	45.10
15	25.73	14.32	85	8	44.10
20	26.66	14.38	96	7	44
25	26.34	12.12	68	8	44
30	26.98	12.66	77	9	42.10
35	27.18	13.18	78	10	46.10
40	26.84	12.86	75	10	46.10
45	26.11	10.98	65	11	41.20
50	26.58	11.82	65	11	43.10
55	27.11	13.16	77	11	45
60	28.33	13.06	73	9	46
65	26.62	11.13	64	12	44.20
70	27.11	11.74	67	12	46
75	28.57	13.19	86	10	45.30
80	26.97	13.04	76	10	47.10
85	29.35	13.95	71	11	51
90	25.92	14.19	91	10	38.70
95	26.71	13.10	93	10	40
100	28.17	13.47	73	10	46

6.3.5 Learning

This algorithm completes Gathering when all agents have seen at least one other Agent with an ID smaller than itself other than the Agent with the smallest ID. The first detectable flaw of this algorithm is that the Agent with 2nd smallest ID has to spot the Agent with the smallest ID to complete Gathering. This limitation can be passed to other agents as well; 3rd smallest ID agent has to spot one of the two agents with smaller IDs to become a follower and so on. This limitation causes a Maximum number of rounds required to gather half of the previous algorithm. For the previous algorithm, the leading Agent stopped every alternate round; hence the actual number of moving rounds is still the same.

The following comparison of Type 0 and Type 1 shows an average number of rounds required significantly improved due to the cluster of agents moving every round instead of alternate rounds to preserve the cluster.

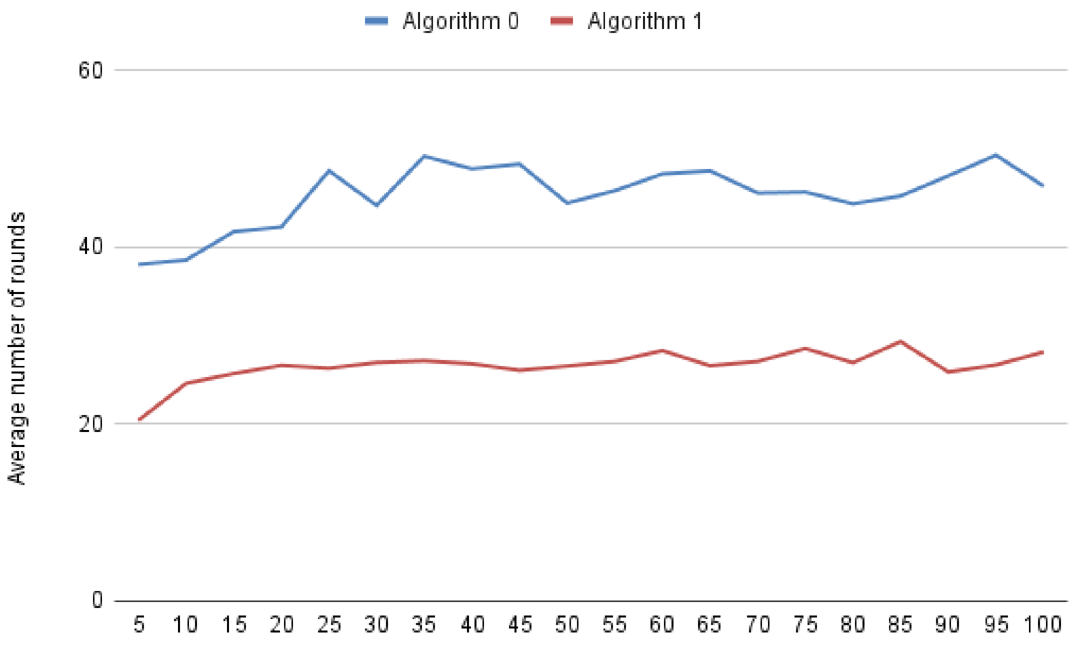


Figure 6.1: Comparing Average number of rounds required to gather for Type 0 and Type 1

The fall in Standard deviation is due to fall in average.

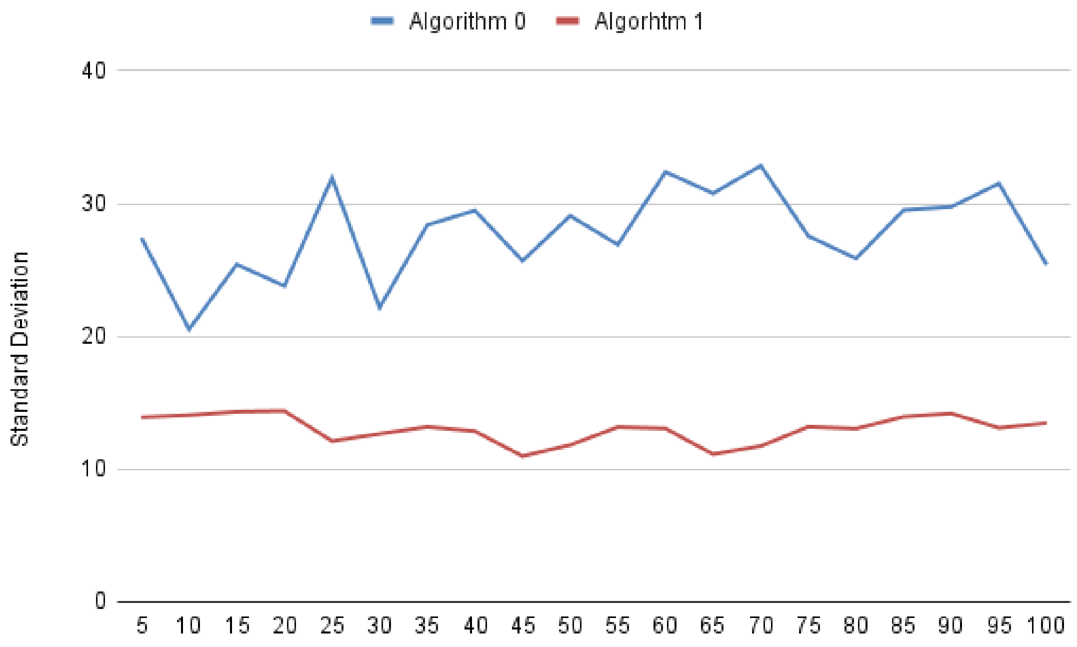


Figure 6.2: Comparing Standard deviation for Type 0 and Type 1

Since clusters formed are moving around in a train and the longer the train, the easier it is for the cluster to be spotted, we can try to improve performance by increasing the train size; the following algorithm will be attempting that.

6.4 Gathering Using Train of Random Followers (Type 2)

6.4.1 Changes from the previous algorithm

Collaboration plays a significant role in ensuring the problem is solved efficiently, and there are different ways in which agents can collaborate in gathering. In the previous algorithm, the agents in the same cluster formed a chain which, when seen by other Agents with smaller IDs, can be followed to merge the chains or clusters.

So theoretically, if the chain sizes are increased, clusters will find it easier to discover each other. In the previous algorithm, the agents followed the smallest ID agent they saw first while they were non-followers, and once they followed that Agent, they became permanent followers and never switched their leading Agent, this limits the chain size if many agents are in a section of polygon and all of them can see each other to just two agent chain as there will be just one leading Agent. However, if the agents are made to randomly choose among all the agents with IDs smaller than theirs, the chain formed would be longer, and the Agent with the smallest ID will not be forced to become the leading Agent for all the agents, thus increase the visible reach of the cluster.

6.4.2 Initialization

Initialization is similar to previous algorithm.

Algorithm 14 Initialization of member variables (Type 2)

Input: polygon, ID

Output: None

```
Agent polygon = polygon
Agent position = Random point in polygon
Agent ID = ID
Agent Movement = Empty List
Agent Movement = Agent Movement + {Agent Position}
Agent Following = Null
Agent Followers = Empty list
Agent Followers = Agent Followers + {Agent ID}
```

6.4.3 Phases of each round

Only a slight change in Compute phase is required from the previous algorithm. The required four phases are in order: Look, Compute, Communicate and Move.

Look

Same as the previous algorithm.

Algorithm 15 Look Phase (Type 2)

Input: polygon, list of agents, Agent position

Output: None

```
Agent visibility_polygon = visibility polygon generated from Agent's position
Agent visible_agents = sorted list of other agents within Agent visibility_polygon
```

Compute

Similar to the previous algorithm, the difference is that instead of picking the Agent with the smallest ID within the visibility polygon among all agents with a smaller ID than its ID, an agent chooses any random agent with a smaller ID than itself.

Algorithm 16 Compute Phase (Type 2)

```
Input: None
Output: None
  if All agents are at same location then
      Classify agent as gathered and stop moving
  else
      if Agent following is set then
          Agent next_move = Agent following's position
      else
          if Agent followers list has all agents then
              stop moving the agent
          else
              if Agent visible_agents[0]'s ID < Agent ID then
                  Candidate = Agent visible_agents with ID < Agent ID
                  Agent following = Randomly chosen from Candidate
                  Agent next_move = Agent following's position
              else
                  Agent next_move = Pick a random point in visibility polygon
              end if
          end if
      end if
  end if
```

Communicate

This algorithm uses the same communication phase as the previous algorithm. The trailing Agent takes the union of **Agent followers** from its memory and its leading Agent, unions it, and sets the new list as **Agent followers** in the leader agent's memory.

Algorithm 17 Communicate Phase (Type 2)

```
if Agent following is set then
    Agent following's followers= Agent following's followers $\bigcup$ Agent followers
end if
```

Move

The move phase is the same as the previous algorithm.

Algorithm 18 Move Phase (Type 2)

```
Input: None
Output: None
  if Agent next_move is set then
      Agent position = Agent next move
      Agent next_move = NULL
  end if
  Agent Movement = Agent Movement + {Agent Position}
```

6.4.4 Results

Similar to the results in the previous algorithm, the following are the statistics of the number of rounds required to gather for the different number of agents using **Gathering using Train of Random Followers (Type 2)**, Each configuration is simulated 100 times. In each simulation number of agents, depending upon the configuration, were randomly placed in the polygon, and the algorithm was executed; the number of rounds required to gather was noted.

Table 6.4: Gathering Using Train of Random Followers (Type 2) simulation result

AGENTS	AVERAGE	STDEV	MAX	MIN	90th Percentile
5	19.12	11.76	80	5	32.10
10	25.94	11.94	68	8	43.10
15	26.55	17.19	153	9	41.10
20	28.21	15.78	110	9	46.10
25	28.37	13.60	89	11	44.30
30	28.29	11.65	65	12	46.10
35	27.8	11.57	83	11	43
40	29.12	13.69	82	11	49
45	30.62	12.44	74	12	48
50	28.69	12.03	79	12	47.10
55	30.49	13.02	65	13	51
60	29.11	12.25	71	13	45
65	28.72	12.90	74	13	45.20
70	30.06	14.12	88	14	44.10
75	30.97	14.15	98	15	43
80	28.94	11.70	68	15	45.10
85	31.47	13.65	76	15	52.30
90	29.45	11.76	77	15	44
95	28.68	11.25	92	15	42.20
100	30.18	11.38	66	15	47.20

6.4.5 Learning

This algorithm completes Gathering when all agents have seen at least one other Agent with an ID smaller than itself other than the Agent with the smallest ID. The flaw of this algorithm is the same as we discussed earlier the Agent with 2nd smallest ID has to spot the Agent with the smallest ID to complete Gathering. The number of rounds required to form the cluster remain the same. The average number of rounds required to gather increases in this algorithm as the train size has increased, and the few extra rounds required are to collapse the cluster of followers at the leader's position.

The following comparison of Type 1 and Type 2 shows that the number of rounds required is slightly higher than the previous algorithm.

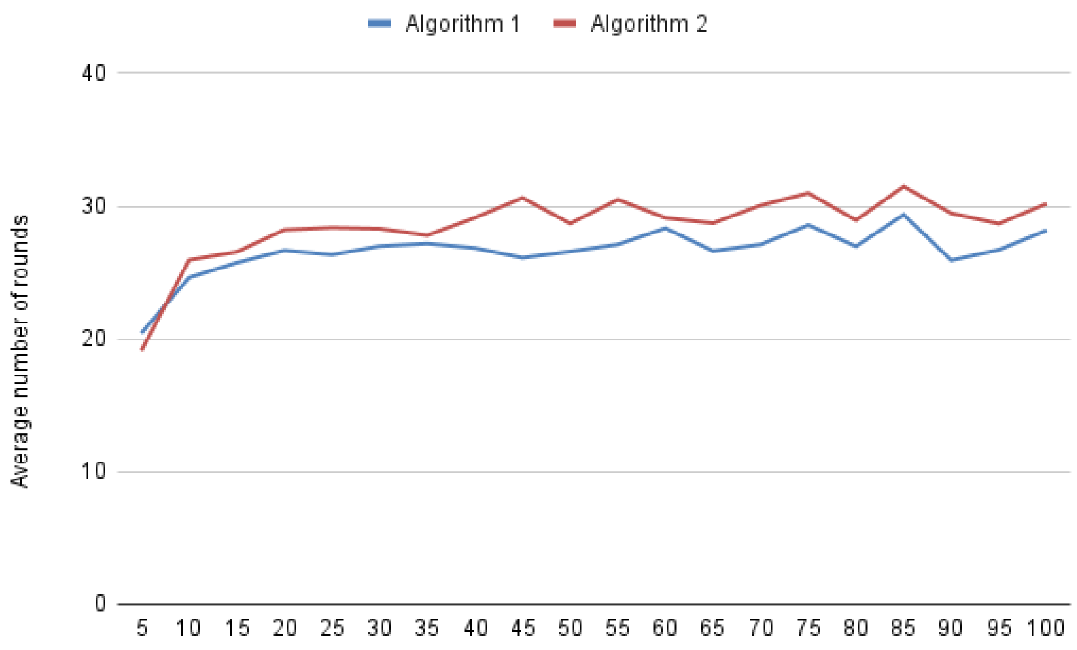


Figure 6.3: Comparing Average number of rounds required to gather for Type 1 and Type 2

The standard deviation remains the similar to previous algorithm.

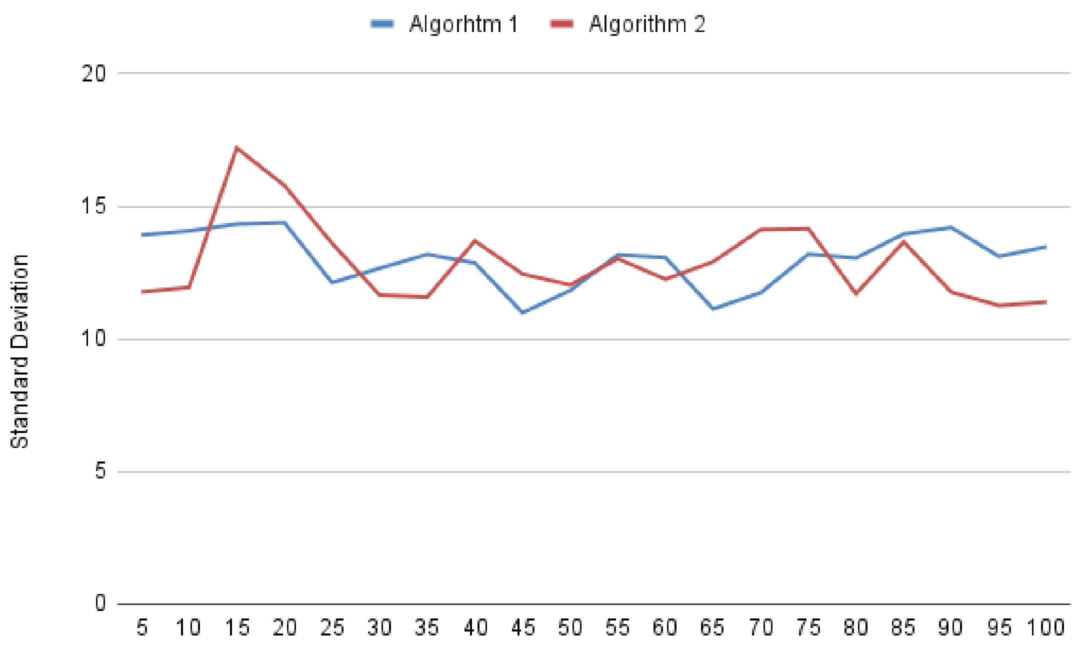


Figure 6.4: Comparing Standard deviation for Type 1 and Type 2

The underlying problem persists of gathering in lower ID agents, and the following algorithm proposes a solution for this problem.

6.5 Gathering Using Train of Random Followers with cluster ID (Type 3)

6.5.1 Changes from the previous algorithm

In previous algorithms, the Agent broadcasts its ID during the scan phase and only becomes a leading agent if a non-follower agent with smaller ID spots it, but in this algorithm, a cluster ID is maintained and is broadcast by the agents in the cluster; the cluster-ID is the same as Agent ID of leading Agent; this improves merging capability of trailing agents as their ID will be much higher than the leading Agent.

This change in the algorithm should be able to improve consistency in the number of rounds required to gather. Unlike the **Agent followers** which is passed from followers to the leaders, the cluster-ID is passed from the leading Agent to the followers during the communication phase of each round, and it is not a list, just an ID.

6.5.2 Initialization

Initialization is similar to the previous algorithm. Another member variable added is cluster-ID, named as **Agent following_ID**, initially set as its ID but changes when it follows other agents with smaller IDs.

Algorithm 19 Initialization of member variables (Type 3)

```
Input: polygon, ID
Output: None
  Agent polygon = polygon
  Agent position = Random point in polygon
  Agent ID = ID
  Agent Movement = Empty List
  Agent Movement = Agent Movement + {Agent Position}
  Agent Following = Null
  Agent Followers = Empty list
  Agent Followers = Agent Followers + {Agent ID}
  Agent Following_ID = Agent ID
```

6.5.3 Phases of each round

Only a slight change in Communicate and Compute phase, the algorithm requires four phases which are in order: Look, Compute, Communicate and Move.

Look

Same as the previous algorithm.

Algorithm 20 Look Phase (Type 3)

```
Input: polygon, list of agents, Agent position
Output: None
  Agent visibility_polygon = visibility polygon generated from Agent's position
  Agent visible_agents = sorted list of other agents within Agent visibility_polygon
```

Compute

Similar to the previous algorithm, the difference is that instead of picking the Agent with the smallest ID within the visibility polygon among all agents with a smaller ID than its own, the Agent chooses any random agent with smaller **Agent following_ID** than itself. To facilitate this the agents listed in **Agent visible_agent** need to be sorted by **Agent following_ID** instead of **Agent ID**, and treated similarly as did before.

Algorithm 21 Compute Phase (Type 3)

```
Input: None
Output: None
  if All agents are at same location then
      Classify agent as gathered and stop moving
  else
      if Agent following is set then
          Agent next_move = Agent following's position
      else
          if Agent followers list has all agents then
              stop moving the agent
          else
              V_A=Sort Agent visible agents with respect to Agent following_ID
              if V_A[0]'s following_ID $<$ Agent following_ID then
                  C_F = all agents in V_A with following_ID $<$ Agent following_ID
                  Agent following = Randomly chosen from C_F
                  Agent next_move = Agent following's position
              else
                  Agent next_move = Pick a random point in visibility polygon
              end if
          end if
      end if
  end if
```

Communicate

This algorithm uses the same communication phase as the previous algorithm. The trailing Agent takes the union of **Agent followers** from its memory and its leading Agent, unions it, and sets the new list as **Agent followers** in the leader agent's memory; following that, one bit of information is pulled from the leader agent's memory which is its **Agent following_ID**. The trailing Agent copies this value into its **Agent following_ID**

Algorithm 22 Communicate Phase (Type 3)

```
if Agent following is set then
    Agent following's followers= Agent following's followers $\bigcup$ Agent followers
    Agent following_ID= Agent following's following_ID
end if
```

Move

The move phase is the same as the previous algorithm.

Algorithm 23 Move Phase (Type 3)

```
Input: None
Output: None
  if Agent next move is set then
      Agent position = Agent next move
      Agent next move = NULL
  end if
  Agent Movement = Agent Movement + {Agent Position}
```

6.5.4 Results

Similar to the results in the previous algorithm, the following is the statistics of the number of rounds required to gather for a various number of agents using **Gathering Using Train of Random Followers with cluster ID (Type 3)**, Each configuration was simulated a 100 times. In each simulation number of agents, depending upon the configuration, were randomly placed in the polygon, and the algorithm was executed; the number of rounds required to gather was noted.

Table 6.5: Gathering Using Train of Random Followers with cluster ID (Type 3) simulation result

Agents	AVERAGE	STDEV	MAX	MIN	90th Percentile
5	19.37	10.02	49	4	33
10	24.76	11.93	63	8	44
15	22.9	10.35	71	11	35.10
20	26.1	8.46	53	12	37.20
25	27.38	11.34	85	12	43
30	25.81	8.77	49	13	39
35	27.94	9.07	62	12	40.10
40	27.56	9.33	64	14	37
45	27.49	10.57	79	13	41.20
50	30.09	10.64	66	14	43.20
55	30.55	10.68	89	15	44
60	29.17	8.33	54	15	41
65	30.69	9.88	62	16	44.30
70	29.56	8.36	70	17	39.10
75	30.09	8.82	56	17	43
80	30.62	8.01	73	19	40
85	33.11	11.45	85	19	46.40
90	31.16	9.06	73	16	41.10
95	33.63	11.68	87	18	47.10
100	33.27	9.27	66	18	44

6.5.5 Learning

This algorithm completes Gathering when all agents have seen at least one other Agent with cluster-ID smaller than its ID other than the Agent with the smallest ID. This algorithm is an improvement over the previous one as the Agent with 2nd lowest ID only needs to find the cluster with the lowest ID instead of the specific Agent. This algorithm performs better than Type 2 for less number of agents as the cluster-ID gets quickly spread among the followers, and the clusters can merge much more quickly. However, the algorithm is less efficient comparatively when the number of agents increases as the cluster-ID needs to pass from leader to follower, and if the clusters are long, it might take longer and also force the Agent with 2nd lowest ID to follow an agent far away from leader which leads a long time for the cluster to collapse.

The following comparison of Type 2 and Type 3 shows that the number of rounds required is slightly lower than the previous algorithm while there are fewer agents in the polygon but goes higher as the number of agents increases.

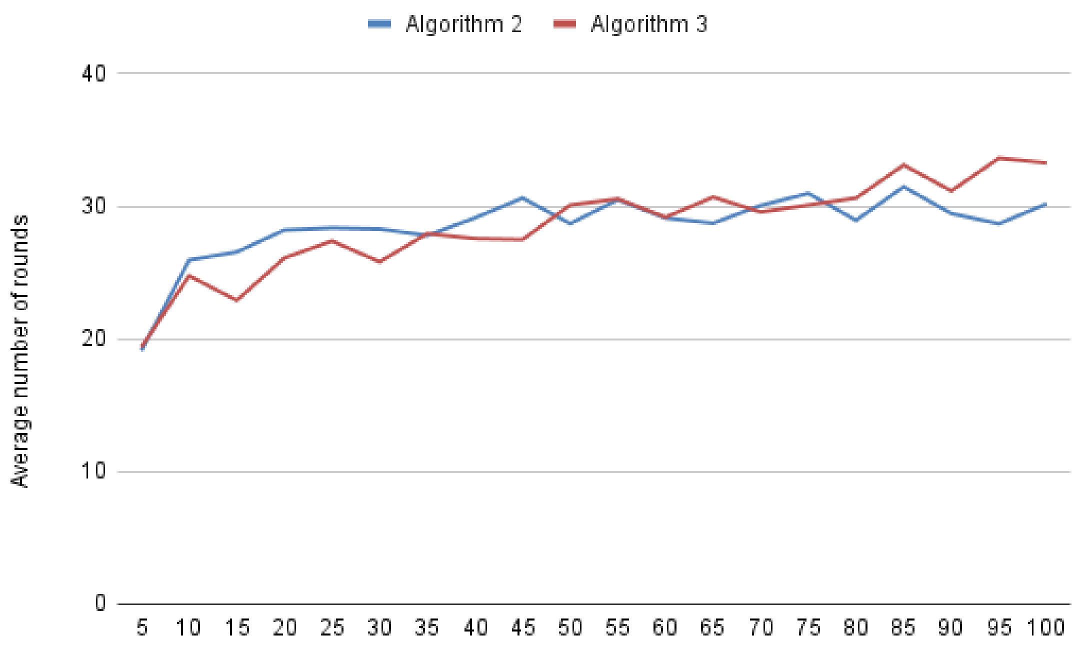


Figure 6.5: Comparing Average number of rounds required to gather for Type 2 and Type 3

The standard deviation is lower compared to the previous algorithm; as discussed, the Agent with 2nd lowest ID will discover the cluster with a lower ID with much more ease than the Agent with the lowest ID as it had to in Type 2.

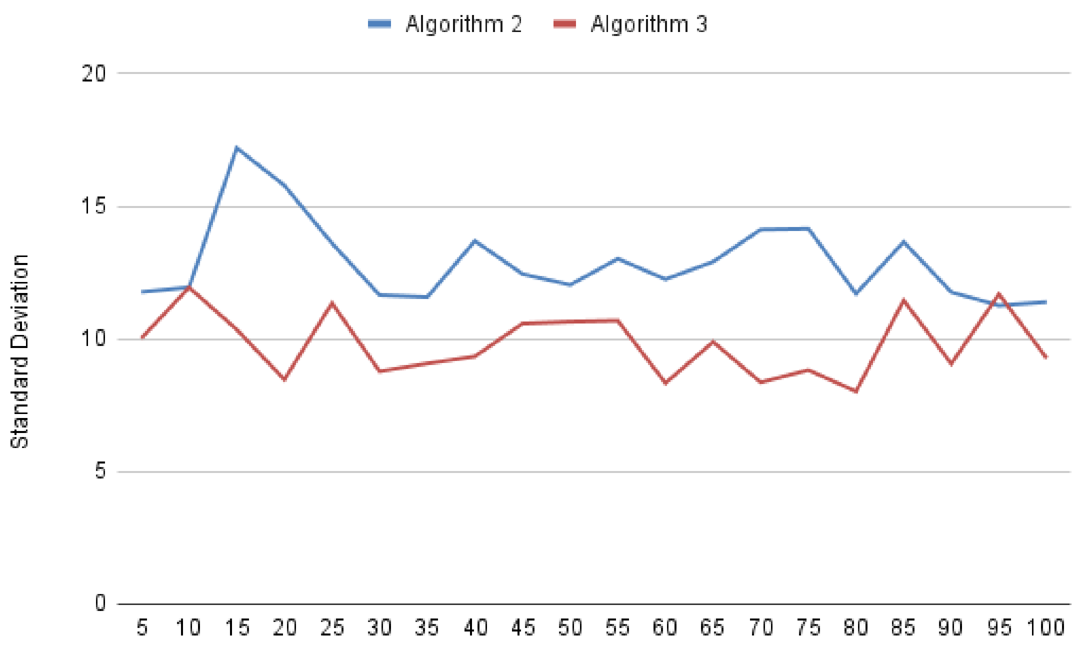


Figure 6.6: Comparing Standard deviation for Type 2 and Type 3

In this algorithm, the time to collapse a train/cluster of agents affects the number of rounds required to gather. The following algorithm proposes a way to quickly collapse the cluster once all agents are within one cluster.

6.6 Gathering Using Train of Random Followers with cluster-ID and quick collapse (Type 4)

6.6.1 Changes from the previous algorithm

After observing the data and simulation of previous algorithms, it can be seen that the agents travel the entire train of following agents to reach the leading Agent once the leader has stopped moving, even if the trailing agents can see the leader and know that the leader has stopped. To reduce the time required in collapsing the clusters, the agents check each round if they can see the Agent with the same ID as **Agent following_ID** and check if it has stopped moving; if it has, then it changes its **Agent following** to the stopped Agent and moves directly to it, this reduces the time required for the cluster to reach the leading Agent.

6.6.2 Initialization

No new member variables are added, and the initialization remains the same as a previous algorithm.

Algorithm 24 Initialization of member variables (Type 4)

```
Input: polygon, ID
Output: None
  Agent polygon = polygon
  Agent position = Random point in polygon
  Agent ID = ID
  Agent Movement = Empty List
  Agent Movement = Agent Movement + {Agent Position}
  Agent Following = Null
  Agent Followers = Empty list
  Agent Followers = Agent Followers + {Agent ID}
  Agent Following_ID = Agent ID
```

6.6.3 Phases of each round

Only a slight change in Communicate phase and communicate phase is called before the Compute phase for this algorithm. The four phases needed are: Look, Communicate, Compute and Move.

Look

Same as the previous algorithm.

Algorithm 25 Look Phase (Type 4)

Input: polygon, list of agents, Agent position

Output: None

```
  Agent visibility polygon = visibility polygon generated from Agent's position
  Agent visible agents = sorted list of other agents within visibility polygon
```

Communicate

This algorithm uses the same communication phase as the previous algorithm. The trailing Agent takes the union of **Agent followers** from its memory and its leading Agent, unions it, and sets the new list as **Agent followers** in the leader agent's memory; following that, one bit of information is pulled from the leader agent's memory which is its **Agent following_ID**. The trailing Agent copies this value into its **Agent following_ID**

Algorithm 26 Communicate Phase (Type 4)

```
if Agent following is set then
    if Agent visible_agents is non-empty then
        if Agent visible_agents[0]'s ID== Agent following_ID then
            if Agent visible_agents[0] has stopped moving then
                if Agent visible_agents[0] != Agent following's ID then
                    Agent following = Agent visible_agents[0]
                end if
            end if
        end if
    end if
end if
if Agent follower is set then
    Agent following's followers= Agent following's followers $\bigcup$ Agent followers
    Agent following_ID= Agent following's following_ID
end if
```

Compute

The Compute phase is the same as the previous algorithm compute phase.

Algorithm 27 Compute Phase (Type 4)

```
Input: None
Output: None
  if All agents are at same location then
      Classify agent as gathered and stop moving
  else
      if Agent following is set then
          Agent next move = Agent following's position
      else
          if Agent followers list has all agents then
              stop moving the agent
          else
              V_A=Sort Agent visible_agents with respect to Agent following_ID
              if V_A[0]'s following_ID < Agent following_ID then
                  C_F = all agents in V_A with following_ID < Agent following_ID
                  Agent following = Randomly chosen from C_F
                  Agent next_move = Agent following's position
              else
                  Agent next_move = Pick a random point in visibility polygon
              end if
          end if
      end if
  end if
```

Move

The move phase is the same as the previous algorithm.

Algorithm 28 Move Phase (Type 4)

```
Input: None
Output: None
  if Agent next move is set then
      Agent position = Agent next move
      Agent next move = NULL
  end if
  Agent Movement = Agent Movement + {Agent Position}
```

6.6.4 Results

Similar to the results in the previous algorithm, the following are the statistics of the number of rounds required to gather for the different number of agents using **Gathering Using Train of Random Followers with cluster-ID and quick collapse (Type 4)**, Each configuration was simulated a 100 times. In each simulation number of agents, depending upon the configuration, were randomly placed in the polygon, and the algorithm was executed; the number of rounds required to gather was noted.

Table 6.6: Gathering Using Train of Random Followers with cluster ID and quick collapse (Type 4) simulation result

Agents	AVERAGE	STDEV	MAX	MIN	90th Percentile
5	19.62	12.24	71	6	35.10
10	21.32	12.07	77	7	37.10
15	21.9	11.91	66	9	39.20
20	22.08	8.93	55	9	35
25	24.3	11.88	77	10	38
30	23.24	10.98	78	9	35
35	22.34	8.49	48	10	33.10
40	24.33	10.94	63	11	39.10
45	23.27	8.07	54	10	32
50	25.07	9.33	59	11	37
55	24.92	10.99	83	12	38.10
60	25.1	9.28	58	11	36
65	26.6	9.59	75	13	38.20
70	24.48	7.89	48	11	37
75	23.95	8.05	49	12	35
80	25.72	8.76	56	11	38.10
85	25.49	10.16	64	12	40.10
90	26.21	8.96	54	13	38.10
95	25.08	9.66	78	11	36.20
100	24.33	8.89	62	13	35.10

6.6.5 Learning

The underlying cluster formation format remains the same as Type 3, but the collapsing of the cluster is changed and massively affects the average number of rounds required to gather, especially when the cluster/train is enormous. It is much quicker.

The following comparison of Type 3 and Type 4 shows that the number of rounds required is lower than the previous algorithm while there are fewer agents and goes lower as the number of agents increases in the polygon.

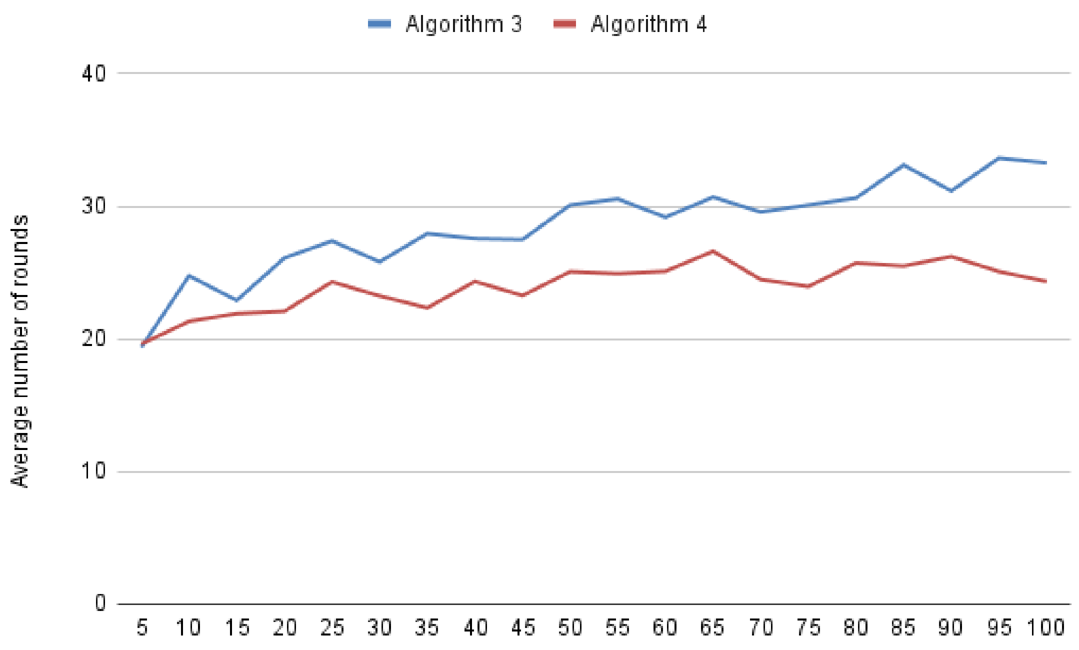


Figure 6.7: Comparing Average number of rounds required to gather for Type 3 and Type 4

The standard deviation remains the same as the previous algorithm.

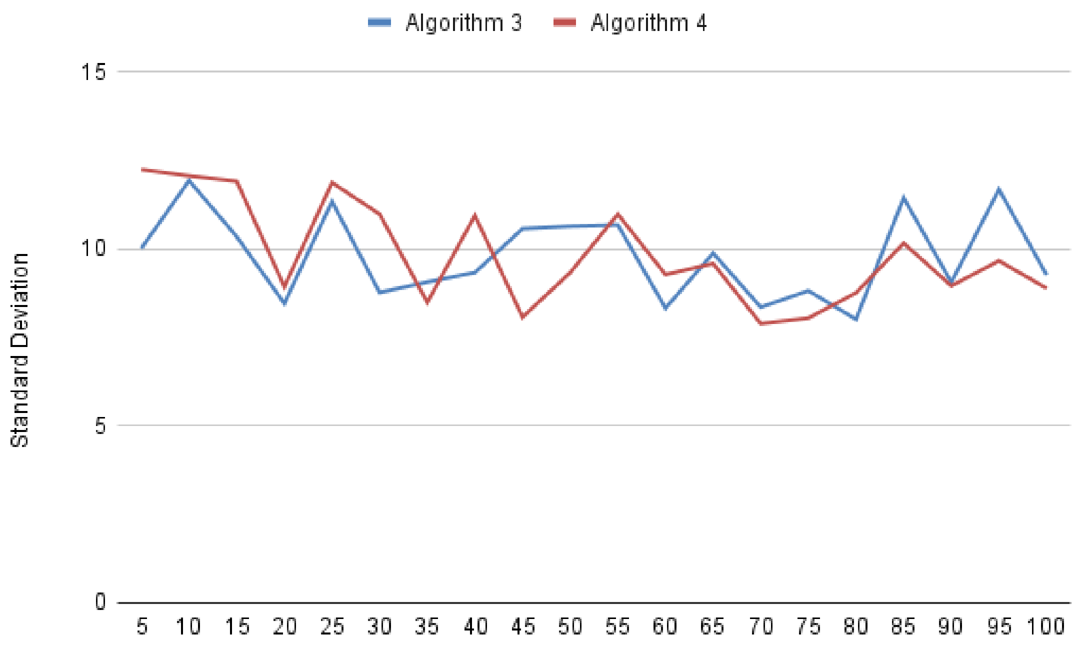


Figure 6.8: Comparing Standard deviation for Type 3 and Type 4

The following figures compare all the five algorithms discussed in this chapter.

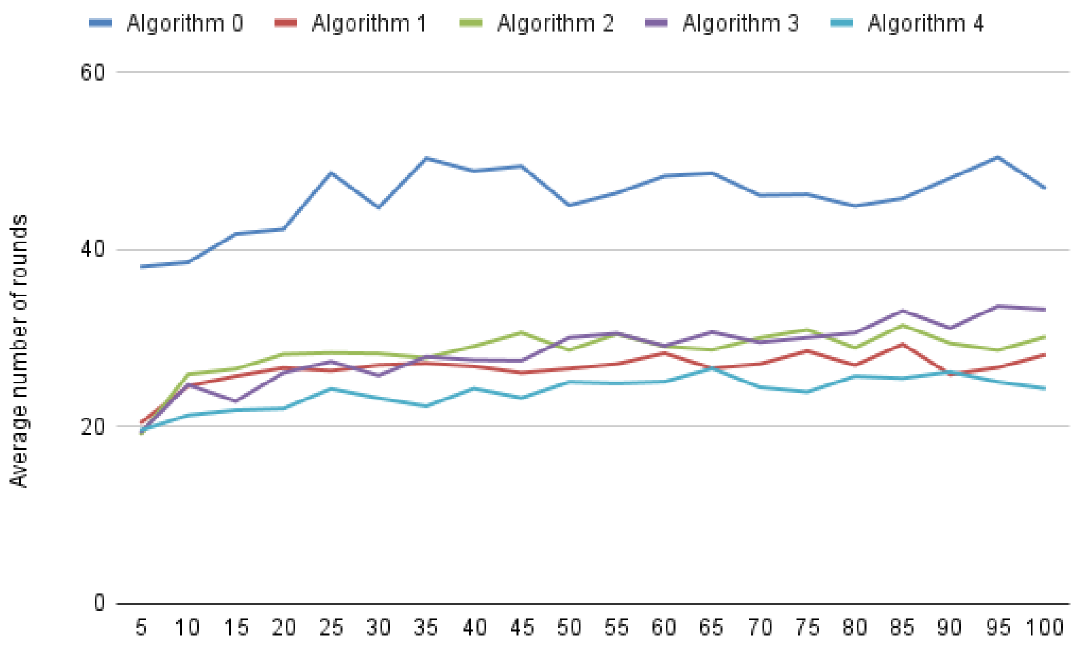


Figure 6.9: Comparing Average number of rounds required to gather for all algorithms

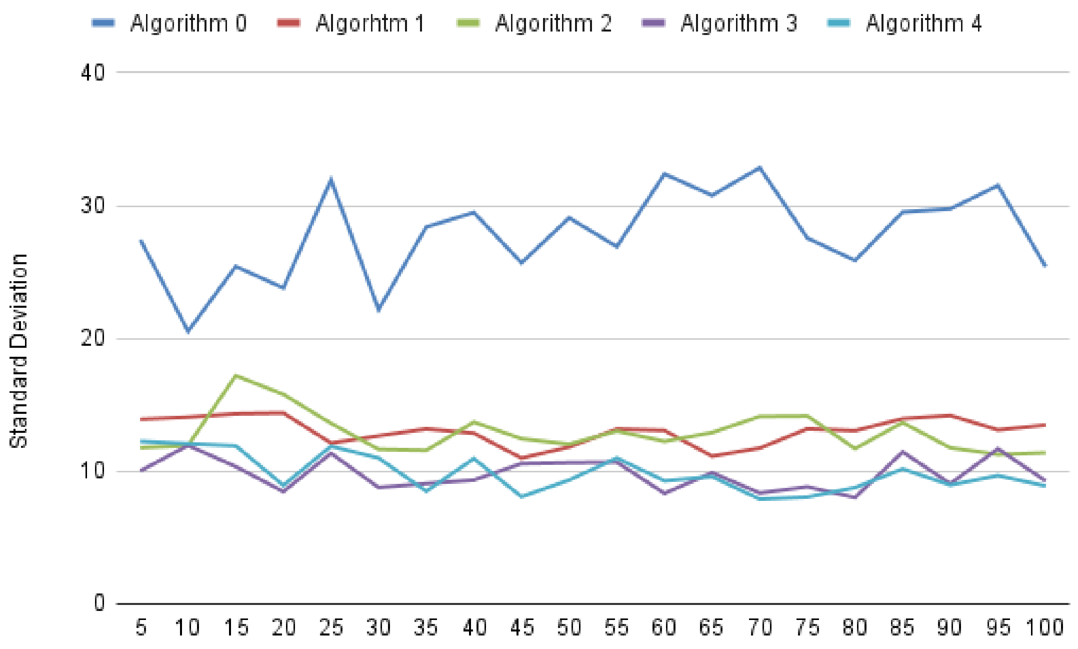


Figure 6.10: Comparing Standard deviation for all Algorithms

CHAPTER 7

Summary and Conclusion

The report discusses Distributed computing domain and how mobile agents use it to communicate, collaborate and solve problems. Followed by the problem statement that is Gathering mobile agents in a polygonal environment and all the related problems and algorithms that might help build an algorithm to solve the problem and analyze it.

Following this, discussed a few potential algorithms, each with unique methods of solving the problem of gathering. All these algorithms used random movement to some degree and were expected to use collaboration to gather efficiently.

The simulation framework was worked on, which can be used to run and simulate any algorithm. Each phase and the agents' functionality are explained, elaborating on how each phase is simulated just like an agent would in the real world. Following that, the simulation framework was put to use, and six different Gathering algorithms were tried and compared to each other. The results are studied and provided reasons to justify all of them.

7.1 Future scope

The simulation framework provided can be used to simulate various algorithms and can be used as a visual tool to teach distributed computing in mobile agents. The focus now is to turn this tool into an easy-to-understand and use framework for students newly entering the field so they can implement and test their algorithms.

The report discusses randomized algorithms for Gathering; the problem of deterministic Gathering of mobile agents in a polygon is yet to be studied similarly, which leaves room for further research and algorithm designing.

References

[1] Barath Ashok, John Augustine, Aditya Mehekare, Sridhar Ragupathi, Srikkanth Ramachandran, and Suman Sourav. "Guarding a Polygon Without Losing Touch". In: *International Colloquium on Structural Information and Communication Complexity*. Springer. 2020, pp. 91–108.

[2] Daisuke Baba, Tomoko Izumi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. "Linear time and space gathering of anonymous mobile agents in asynchronous trees". In: *Theoretical Computer Science* 478 (2013), pp. 118–126.

[3] Sébastien Bouchard. "On the Deterministic Gathering of Mobile Agents". PhD thesis. Sorbonne université, 2019.

[4] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. "Distributed computing by mobile robots: Gathering". In: *SIAM Journal on Computing* 41.4 (2012), pp. 829–879.

[5] Frank Hoffmann, Christian Icking, Rolf Klein, and Klaus Kriegel. "The polygon exploration problem". In: *SIAM Journal on Computing* 31.2 (2001), pp. 577–600.

[6] Adrian Kosowski. "Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network". PhD thesis. Université Sciences et Technologies-Bordeaux I, 2013.

[7] Karl J Obermeyer, Anurag Ganguli, and Francesco Bullo. “Multi-agent deployment for visibility coverage in polygonal environments with holes”. In: *International Journal of Robust and Nonlinear Control* 21.12 (2011), pp. 1467–1492.

[8] Hongtao Zhou, Wenfeng Zhou, and Wei Zeng. “Flocking control of multiple mobile agents with the rules of avoiding collision”. In: *Mathematical Problems in Engineering* 2015 (2015).