Foraging in the Presence of Obstacles

by

Yi Xuan Wang

A thesis submitted to the Faculty of Graduate
and Postdoctoral Affairs in partial fulfillment of
the requirements for the degree of

Master

in

Computer Science

Carleton University (Ottawa, Ontario)

©2017

Yi Xuan Wang

Abstract

This thesis focuses on investigating searching algorithms which can solve the Ants Nearby Treasure Search (ANTS) Problem in the presence of obstacles. In the ANTS problem, there are k ants (modelled as mobile agents) initially located at the origin in a two-dimensional grid. Pheromones are used as physical markers to allow the ants to perform collaborative search. The target treasure is located at an unknown location at distance D from the origin.

This thesis studied ant foraging in the systems with and without obstacles. A simple deterministic foraging algorithm is provided first, which uses synchronous identical ants for the environment without obstacle. This simple foraging algorithm is a spiral searching algorithm with comparable complexity to the main algorithm of C. Lenzen and T. Radeva [1]. One significant improvement is that the ants do not need to know the direction in which the nest lies. An extension of this algorithm using an additional marker can achieve a global termination and can allow all the ants to find the target.

Further, two additional types of algorithms for ant foraging in a $N \times N$ square terrain are introduced. The two types algorithms target different models which contain obstacles in different categories. The Zig-Zag foraging algorithm and the Up-Down foraging algorithm solve the searching problem using a single ant in a bounded environment with large obstacles. It is shown that the ant is able to successfully explore the entire terrain with pheromones provided that the passages between obstacles have width at least five cells. For the second type, the spiral searching algorithms work in the model with k ants in a wrap-around environment having randomly placed single cell obstacles. All of our algorithms are implemented in Netlogo and the corresponding simulations and detail explanations are presented by using examples.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Evangelos Kranakis for his guidance, patience, and unrelenting support throughout my study. I have been extremely lucky to have him as my supervisor who cared so much about my work, who responded to all my questions, and who encouraged me and guided me at all times. I would also like to thank Dr. Stefan Dobrev who contributed to the idea of the Up-down foraging in Chapter 5 after our discussions.

Special thanks to Steven Porretta who was willing to spend time on proofreading my thesis, even though he was busy with his own works. Without the support and friendship provided by him and the other colleges in my office, the Entrust Lab at Carleton, my school life would have been more difficult and less fun.

Last but not least, I would like to thank my Mom for her support and understanding. No matter what choices I make, she always stands on my side. I would also like to thank my boyfriend and all my friends for their love, support, and company during my graduate study.

Contents

Abstract	i
Acknowledgements	ii
List of Figures	v
List of Algorithms	vii
1 Introduction	1
1.1 Motivation	2
1.2 Model	2
1.3 Outline and Results	5
2 Literature Review	6
2.1 Ant-Like Mobile Agents	7
2.2 The ANTS problem	8
2.3 The Number of ants	11
2.4 The Power of Pheromones	13
2.5 The Optimal Utilization of Pheromone	14
2.6 The Capabilities of Ants	16
2.7 The Global Control of the System	17
2.8 The Fault-tolerance Design	19
2.9 The Presence of Obstacles	20
3 Foraging in the Environment without Obstacles	23
3.1 Ant Foraging with Multiple Ants	24
3.1.1 Simple Search Algorithm	24
3.1.2 Simple Search Algorithm with Global Termination	26
3.2 Ant Foraging with Multiple Nests	28
3.3 Conclusion	32
4 Zig-Zag Foraging in the Presence of Large Obstacles	34
4.1 Zig-Zag Foraging	35

4.2	Main Algorithms: Fillup and ExploreObject	36
4.3	Intuitive Discussion of Correctness	39
4.3.1	Single Rectangular Obstacle	39
4.3.2	Multiple Rectangular Obstacles without Overlapping	45
4.3.3	Complex Obstacles	50
4.3.3.1	Convex Obstacles	50
4.3.3.2	Obstacles with "Cusps"	52
4.3.3.3	Problematic Case	56
4.4	Conclusion	60
5	Up-Down Foraging in the Presence of Large Obstacles	62
5.1	Up-Down Foraging	63
5.2	Main Algorithms: Up-down Fillup and Two Rounds ExploreObject	63
5.3	Intuitive Discussion of Correctness	68
5.3.1	Obstacles with "West-Opening Cusps"	69
5.3.2	Obstacles with "Narrow Openings"	72
5.3.3	Multiple Complex Obstacles	73
5.4	Conclusion	75
6	Foraging in the Presence of Randomly Placed Obstacles	76
6.1	Modified Simple Spiral Search	77
6.2	"Pseudo-Random Walk" Search	79
6.3	"Pseudo-Random Walk" Search with Tolerance	80
6.4	Up-down Foraging vs "Pseudo-Random Walk" Algorithm	82
6.5	Modified Simple Algorithm vs "Pseudo-Random Walk" Algorithm .	85
6.6	The Obstacle Concentration Threshold	87
6.6.1	Blocking in a $2 \times n$ grid	88
6.6.2	Blocking in an $N \times N$ grid	93
6.7	Conclusion	95
7	Conclusion and Future Works	97
7.1	Conclusion	97
7.2	Future Works	98
	Abbreviations	100
	Bibliography	106

List of Figures

1.1	The ant can see its eight neighbour cells and can move to any of the four adjacent cells.	4
3.1	Movement of multiple ants using simple search algorithm, Algorithm 3.1.	25
3.2	Movement of multiple ants using simple search algorithm with notification, Algorithm 3.2.	27
3.3	Movement of multiple ants from multiple nests using simple search algorithm, Algorithm 3.1.	29
3.4	Movement of multiple ants from multiple nests using efficient search algorithm, Algorithm 3.3.	30
3.5	Movement of multiple ants from multiple nests using modified efficient search algorithm, Algorithm 3.4.	31
3.6	Movement of multiple ants from multiple nests using modified efficient search algorithm, Algorithm 3.4, for a long time.	32
4.1	Ant traversal of a terrain with obstacles.	36
4.2	Ant traversal of a terrain with the presence of a rectangle.	40
4.3	Ant traversal of a terrain with the presence of a rectangle.	45
4.4	Ant traversal of multiple rectangles environment	47
4.5	Ant traversal of multiple rectangles environment	49
4.6	A convex object	50
4.7	Ant traversal of an environment with multiple convex obstacles	51
4.8	Ant escape from no empty path situation.	52
4.9	A large object with cusps	53
4.10	Ant traversal of differently oriented cusps	55
4.11	Ant traversal of a obstacle with a west opening cusp.	56
4.12	The detection of the cusps using the modified exploration algorithm, Algorithm 4.3.	58
4.13	Ant traversal of obstacles with west opening cusps (the problematic case).	60
5.1	Ant traversal of a west opening cusp.	71
5.2	Ant traversal of an enclosure area with narrow opening.	72
5.3	Ant traversal of a terrain with complex obstacles.	74

6.1	Movement of multiple ants using modified simple search algorithm, Algorithm 6.1.	78
6.2	Movement of multiple ants using random search algorithm, Algorithm 6.2.	80
6.3	Movement of multiple ants using “pseudo-random walk” search with Tolerance, Algorithm 6.3.	82
6.4	Up-down foraging in the presence of randomly placed obstacles. . .	83
6.5	“Pseudo-random walk” foraging in the presence of large obstacles. .	84
6.6	Number of failures using Algorithm 6.1 and Algorithm 6.2 in 100×100 grid with the obstacle concentration $C_{obs} \leq 3.0\%$	86
6.7	Number of failures using Algorithm 6.1 and Algorithm 6.2 in 100×100 grid with the obstacle concentration $0 \leq C_{obs} \leq 20.0\%$	87
6.8	An obstacle at u and the associated set $S(u)$	88

List of Algorithms

3.1	SimpleSearch	24
3.2	Notification	27
3.3	EfficientSearch	29
3.4	EfficientSearchModified	31
4.1	ExploreObject (Right-hand version)	37
4.2	Fillup procedure	38
4.3	Modified ExploreObject (Right-hand version)	58
5.1	Up-down Fillup	65
5.2	ExploreObject (Right-hand version)	66
5.3	SecExpObject	67
5.4	SecUpdownfill	68
6.1	ModifiedSimpleSearch	77
6.2	RandomSearch	79
6.3	RandomSearch with Tolerance	81

Chapter 1

Introduction

The Ants Nearby Treasure Search (ANTS) problem was first introduced by Feinerman *et al.* in 2012 [2]. The main concept of the original study utilizes the knowledge and tools from distributed computing to study a biologically inspired scenario. The biological models also help to study the search strategies within the scope of distributed computing. The ANTS problem was introduced as a generalization of the classical cow-path problem [3], which is relevant to collective foraging in animal groups. The ANTS problem considers multiple identical agents instead of a single agent in the cow-path problem. In the ANTS problem, k identical ants (mobile agents) start at the origin (the nest) of a two-dimensional grid, and search for a treasure located at a distance D . The communication is limited between the ants after leaving the nest. The goal of the search is to locate the treasure as fast as possible. A trivial lower bound on the time required to find the treasure is $\Omega(D + D^2/k)$ [2].

1.1 Motivation

Few studies have been done on the ANTS problem in an environment with obstacles, even though the presence of obstacles in the system is a real problem in practice. The goal of this thesis is to provide simple algorithms that can work properly when there are obstructions. In this study, two different designs are used. One uses only a single ant in a closed environment with walls, and the other uses multiple ants in a wrap-around environment. In addition, this thesis covers two types of obstacles. In the closed environment, the obstacles are large objects that are convex and concave, while in the wrap-around environment, the obstacles are randomly placed small square objects. In all designs, pheromones are used as a marker to indicate the visited cells. For all models, the goal is to investigate the algorithms that enable the ants to successfully explore the entire terrain, and at least one of the ants finds the target.

1.2 Model

The model used in this study is based on the one introduced by C. Lenzen and T. Radeva [1], in which pheromones are used as markers. The use of pheromones is, in fact, a way of representing the limited communication between the mobile agents in the network. Several studies have been done to optimize the amount of pheromones required to find the treasures in this type of problem. However, in order to simplify the algorithms, the ants are allowed to leave pheromones on

every cell they visited. The large consumption of pheromones is a trade-off for the simplicity of the algorithms.

Beside the pheromones, the capabilities of the mobile ants should also be carefully designed. There are two typical mobile agents' designs in the ANTS problem studies: Finite State Machine (FSM) and Turing Machine (TM). In this thesis, the ants are designed to switch between states upon specific conditions and have constant memories. The memory size would have crucial impacts on the efficiency of the search. If the ants have extremely large memories, the ants can store the whole paths to predict and draw the entire environment, and then can determine the moves with complex calculations. However, in this thesis, the goal is to have algorithms as simple as possible and require less resources. In addition, the system can be designed to be synchronous or asynchronous. The simulations of the multiple ants algorithms are implemented in Netlogo in which all execution are synchronous. This difference between synchronous and asynchronous systems affects the control of the processes, but has few influences on the complexity of the algorithm.

The ants migrate in a $N \times N$ two-dimensional square grid with integer coordinates. While the ants traverse the grid searching for the target, the ants mark all visited cells in the grid by pheromones. All k ants are initially placed in the nest s which is considered to be the origin. The ants can move to the four adjacent grid cells, but can see the eight adjacent neighbours, as shown in Figure 1.1. Further, the releasing of a pheromone at a cell prevents duplicated search done by the same or different ants. As mentioned above, the motion of the ants in the system is

synchronous, therefore, the cells may be occupied by multiple ants at the same time. To prevent it happening, only one ant is released from the nest at a time. The time complexities of the algorithms are measured in terms of synchronous rounds.

FIGURE 1.1: The ant can see its eight neighbour cells and can move to any of the four adjacent cells.

The searching algorithms are deterministic and the ants can decide where to move next independently based on their local views. The ants can recognise walls and obstacles and can sense pheromones in the eight neighbouring cells to know whether they have been visited or not. However, the ants do not have any sense of direction and do not need to know the location of the nest. Further, there are several assumptions made about the environment and the objects used in this thesis. Different algorithms require different conditions in order to let them work properly. The assumptions are given in details in each chapter when the algorithms are introduced. The grid cell containing the target is always considered to be accessible, that is the obstacles do not have or form any “holes” around the target.

1.3 Outline and Results

First, a simple foraging algorithm for the environment without obstacles is introduced in Chapter 3. The main algorithm is as simple as the algorithm of [1] with comparable complexity. In this algorithm, however, the ants do not need to know the direction in which the nest lies. Chapter 4 introduces the Zig-Zag algorithm that can solve the searching problem in a bounded terrain in the presence of large obstacles in simple configurations. Another algorithm is introduced in Chapter 5, the Up-Down algorithm, that can deal with more complex obstacles than those in Chapter 4. These two algorithms use just a single ant and one type of pheromones. Finally, another group of algorithms is presented in Chapter 6 which handles the case of the presence of randomly placed square obstacles in a wrap-around environment. In this model, multiple ants are used to perform the searching task. This thesis mainly focus on presenting these heuristic algorithms. How each algorithm works in the targeting systems is explained step by step in each chapter and the comparisons between different types of algorithms are done by using the same conditions. All of the given algorithms are simulated and tested in Netlogo. To the best of our knowledge, there are no known publications in the area of distributed computing suggest any groups have solved all of the above cases.

Chapter 2

Literature Review

Insect societies with limited cognitive communication capacity have been studied in the areas of biology and computer science. In computer science, the behaviour of insects has been used as strategy to solve a distributed computing problem. The mobile agents are designed to imitate the behaviour of insects. Each agent is programmed to perform simple tasks with no awareness of the whole environment. Within this chapter, the examples of Ant-like mobile agents in previous studies are discussed, followed by the introduction of the ANTS problem which is a natural foraging inspired treasure searching problem. The following sections survey previous works done in the scope of the ANTS problem. These works focused on different aspects: the number of agents, the use of pheromones, the capabilities of the agents, and the global control. Two special designs are discussed in the last two sections: one is the fault-tolerance design and the other is dealing with the presence of obstacles.

2.1 Ant-Like Mobile Agents

Mobile agents are widely used across many areas of computer science. In this section, the applications of the ant-like mobile agents will be discussed. Mobile agents often refer to mobile robotics or codes that migrate throughout a large environment and interact with nodes or other agents operating in the environment. These agents have relatively small memory sizes and can either operate themselves or perform collaborative works.

Scientists have been focusing their studies on ant-like mobile agents in various directions. Fukuda *et al.* [4] suggested introducing the idea of mobile agents into the context of distrusted computing. They referred these agents as “Messengers” who are autonomous “self-migrating” messages instead of the previously used passive data messages. The leader finding algorithms designed by Korach *et al.* [5] take advantage of a modular technique that uses token circulation which can also be viewed as an early form of mobile agents.

Another major application of ant-like mobile agents is in the Wireless Networks. In [6, 7], the ant agents are applied to the network routing to accumulate information about the connectivity of the nodes in the network. The ant-like mobile agents can also be used to build a network where each agent can function as either forward or backward agents [8, 9]. The use of these two types of agents helps to construct the shortest path between any two nodes in the network and, thus, optimize the network configuration [10]. Later, the same group [11, 12] modified and improved their ant-net design and introduced a model called AntHocNet. The AntHocNet

uses a routing algorithm where the mobile agents are created by the source node whenever it starts a communication with a destination node [13].

In the field of distributed computing, the Rendezvous problem, black hole searching problem, and map construction problem are the typical problems that all involve the application of the ant-like mobile agents. Moreover, mobile agents dispersal problem is another application of mobile agents, whose primary objective is to minimize the time to fill the entire region. In [14], algorithms are given for rapidly dispersing a swarm of robots in a *maze* with *doors*. In addition, asynchronous Finite-State mobile agents with constant memory and restricted visibility can be used to uniformly fill an unknown space [15].

In summary, Lange and Oshima [16] give out seven good reasons for using mobile agents. Out of those reasons, the mobile agents are used in the ANTS problems because of: the agents execute asynchronously and autonomously; they adapt dynamically; they can be robust and fault-tolerant; and they can achieve parallel processing.

2.2 The ANTS problem

The ants nearby treasure search (ANTS) is proposed by Feinerman *et al.* [2]. The main idea of their study is using the tools from distributed computing to explore the behaviour of cooperative collective searches by natural foragers. It is quite important in the biological field that in the situation of limited or even entirely no communication, how efficient the foraging search can be. The ANTS problem

is built as a generalization of the classical cow-path problem within the scope of natural algorithms.

The cow-path problem describes the circumstance that a cow is standing at a crossroads (an origin) with w distributed paths leading to different directions. Only one of the paths leads to a grazing field which is the target at distance D from the intersection. All of the other paths are misleading ends. The cow can only find the target field when standing on it. The cow must walk at least distance D to reach the target. The question is, with no prior knowledge, how to optimize the travelling distance.

The classical cow-path problems, as well as the ANTS problem, study the time complexity of finding the target as a function of the distance between the initial starting point and the location of the target. Baeza-Yates *et al.* [3] published their study on the cow-path problem and proved that the spiral search algorithm is optimal up to lower order terms in the two-dimensional case. Kao *et al.* [17, 18] presented the first randomized algorithm, called SmartCow, that solve the cow-path problem with w paths ($w \geq 2$) in optimal time. The main idea of the SmartCow algorithm is to explore a randomly chosen path up to a random portion of the geometric ratio. Lately, Lopez-Ortiz and Seet [19] studied the parallel searching strategy with multiple agents on the lattice and gave a lower bound which depends on the number of robots. This multi-agents model is much similar to the later defined ANTS problem. However, the path and the direction are pre-defined for each robot according to the number of robots.

The original model of the ANTS problem as designed in the paper of Feinerman *et al.* [2] contains k identical ant agents. These robotic agents are initially placed at a location, seen as the nest, in the two-dimensional plane. The aim of these agents is to search for a treasure located a distance D away from the nest. The agents have no prior information about the location of the treasure. This thesis is based on the same design in the original ANTS problem. As the goal of the general searching algorithm, the core of this study is to locate the treasure in a comparatively short amount of time as a function of the number of ants and the distance. Feinerman *et al.* also calculated the time complexity boundary. The time required for finding the target is $\Omega(D + D^2/k)$, with or even without the knowledge of the total number of agents. This boundary has been used in all of the ANTS problem studies afterwards.

To roughly show the simple spiral search algorithm explained and used in the next Chapter satisfy this lower bound, an N by N two-dimensional plane is used and the nest is located at the center. To discover the treasure at distance D from the central nest, the agents must explore at least all cells having a distance $d < D$ on the two-dimensional plane. By considering integer distances, the total area of these cells should be $(D - 1)^2$. Since there are k agents, on average, each ant should visit $(D - 1)^2/k$ cells. When each round is defined by that every agent gets a chance to discover a new cell, then the time cost for these agents to explore all cells within this $(D - 1)^2$ area is equal to the average cells each agent visited calculated above $(D - 1)^2/k$. Next, to reach the cell at distance D from the centre, there must be at least one agent travelled as far as distance D from the center.

Therefore, the time cost for travelling from center to the layer D from the center should be another factor of D . The overall time requirement should be a factor of $(D + D^2/k)$ which matches what is given as a boundary by Feinerman *et al.* [2].

2.3 The Number of ants

There are various directions one can investigate to improve the design of the ANTS problem. Emek *et al.* [20] focused their study on the requirement of the number of agents to solve the ANTS problem without the use of any forms of external markers. The authors demonstrated the dependencies of the minimum number of agents required on the capabilities of the agents and the timing parameters of the execution environment. In that paper, the authors also give the lower bound of different scenarios. To solve the ANTS problem, as they proved, at least four finite automata type agents are required, whereas in the randomized and synchronous models only three agents are necessary. Additionally, to solve the same searching problem, only a single “pushdown” automaton agent is needed.

In the studies by Emek’s group [21, 22], the authors established a model that using four types of agents. Three of the four agents are used as landmarks for the fourth agent, and the fourth agent performs the spiral searching in the field. The three landmark agents are NorthGuide, WestGuide, and EastGuide respectively and placed in different directions accordingly. The fourth searching agent is called the Explorer and will move around in the environment. During the exploration, the Explorer gives signals to the Guider agents to allowing them to expand the layer

confined. More detailed, when the NorthGuide meets an Explore, it will move one step towards the north. When the WestGuide meets an Explore it moves one step west and then one step south; while when the EastGuide encounters an Explore it moves twice east. After the movements, all Guides will wait for another round. The Explore will continuously perform the triangle search and by passing the Guides to expand the covering. Each time the Explore encounters a Guide, it will change the direction accordingly and move until meets another Guide. This design guarantees that every triangle search works correctly and ends within a finite time.

Later in the study by Emek *et al.* [23], they modified the model from using four asynchronous agents to use only three synchronous agents. There are two Guides in this model: one is a mark of the origin, and the other is a mark for the axis. The Explorer performs rectangle searches instead of triangular. Moreover, at the end of each round, when the explorer reaches the axis Guide, it will expand one step outwards further away from the origin. By using this protocol, the authors show that if each rectangle search is performed correctly, the entire grid is explored.

However, two agents or even one agent models seem to be hardly possible to solve the same problem. This type of Guides and Explores models are studied and well designed as a way to resolving the ANTS problem with the absent of markers or in addition to that [20–22]. In the next section, it can be seen that this limitation on the number of agents required can be overcome by the introduction of the pheromones. The pheromones are used as an alternative of the Guides and provides a way of limited communication once after the agents left the origin. The

addition of pheromones also efficiently reduced the number of agents required to solve the problem.

2.4 The Power of Pheromones

Various groups of scientists have been given out their designs of the ant foraging inspired algorithms. The strategies of communicating between the agents are implemented in different approaches as well. The most typical way, which is also what will be discussed in this section, is using physical marks that the mobile agents can leave behind on their trails. Other methods are, for example, using virtual pheromones [24], through wireless network[25–27], sensor network [28], deployable beacons [29], or robot chains [30, 31].

The pheromone is a type of chemical signal acting as a physical marker. The pheromone is introduced to the ANTS problem by Lenzen *et al.* [1]. It works as an alternative to enhance the communication in the environment between the agents, which is believed to be one of the functions of the pheromone as a biological marker in nature [32]. Although the use of pheromones during the foraging is still unclear, it is a more promising candidate for a naturally evolved foraging algorithm.

In this pheromone model designed by Lenzen *et al.* [1], the environment is an two-dimensional square grid, and the ants are deterministic agents. The ants can move to one of the four adjacent neighbours, and can mark the grid when standing on it by releasing a pheromone. The time complexity is represented in asynchronous rounds where each round is defined as the time cost for all ants take at least a

constant r steps after the end of the last round. The deterministic ants can decide the direction of their movements based on the local views of the four surrounding neighbours. Moreover, the mobile agents used in this model have a significant capability which is the sense of direction. The ants know the direction of the nest and can distinguish between clockwise and counter-clockwise.

With the introduction of pheromones, the searching algorithm is much simpler than the cases without pheromones. There is no longer a need for Guides to help the exploring agents determine where to move for the next step. At each step, the agent checks if the next grid in the clockwise direction has any pheromones, if not, then the agent will move to that spot. Otherwise, it will move to the grid that is further away from the nest. These rules are all based on the pre-factor that the all mobile agents have a sense of direction in the environment. This ant exploration algorithm ensured that, when one of the ants reaches the target at distance D from the nest, all cells that are at most D away from the nest have already been visited. Moreover, the time requirement, when considered in asynchronous rounds, is in order of $O(D + D^2/k)$ as proved by Feinerman *et al.* [2].

2.5 The Optimal Utilization of Pheromone

In the model of using pheromones, a fundamental question is how ants coordinate communication using optimal pheromones to forage for food. This problem was initially approached by [1] using distributed computing theory. By casting the

problem using a simple grid terrain model, they were able to examine the performance of simple pheromone algorithms. They showed that if the food is at a distance at most D from the nest then k ants required $\Omega(D + D^2/k)$ rounds to find the food if k and D are known. They also showed that for any constant $\epsilon > 0$, any $O(\log^{1-\epsilon} k)$ -competitive algorithm requires $\Omega(\log \log k)$ bits of auxiliary information.

On top of the establishment of this pheromone model, Afek *et al.* [33] focused their works on optimizing the consumption of the pheromone, since, pheromones are limited and costly resources in the real world. The set-up is very similar to the one defined by Lenze *et al.* [1], with several assumptions: the ants are deterministic as the previous case, the total number of participating ants is unknown to the ants, and the ants have no means of communication after leaving the nest other than pheromones. When the ants arrive at a grid cell, they will perform some internal computations: first, the ants will check if there is a pheromone at the current location. If there is no pheromone, they will have to choose whether to put one on that cell. After that, the ants can move to another cell determined by a given set of rules. All of these steps are count as a single atomic operation and run in constant time.

The ability of ants to determine the release of pheromone helps to confine the consumption of the pheromone which lead to a optimal utilization. Afek *et al.* [33] first proved that $\Omega(D)$ pheromones are required to find the treasure for Finite State Machine (FSM) type agents. The algorithms they presented are optimal for both the synchronous and asynchronous models. In both cases, it takes $O(D + D^2/k)$

rounds and uses $O(k)$ pheromones to find the treasure. For a more robust model, the Turing Machine (TM) type agents are utilized in the same study. It is shown that no pheromones are required to find the treasure if the TM is used. However, to locate the treasure at the optimal time, at $O(D + D^2/k)$ rounds, the lower bound of the pheromones required is $\Omega(k)$. Similarly to the FSM case, the algorithms for both synchronous and asynchronous models find the treasure in $O(D + D^2/k)$ rounds and reduce the use of pheromones to $O(k)$ from $O(D)$. The differences between the FSM type agents and the TM type agents together with the effects on the efficiency of the searching processes are discussed in the next section.

2.6 The Capabilities of Ants

As mentioned in the previous section, the characteristics of the ants is another way to improve the design. Their capabilities affect how these mobile agents operate. In addition, the size of local memories would also have crucial impacts on the efficiency of the search. Recall, there are two popular types of agents used in the ANTS problem: Finite State Machine [20, 22, 33–35] and Turing Machine [21, 33].

In the case of implementing the mobile agents as Finite State machines, the agents are controlled by three-tuple: a set of states Q , an initial state q_0 , and a transition function σ determining the next step. At each step, the agent can decide to move to North, East, South, West, or stay at the current state. The decisions are made according to the current state and the existence of a pheromone on the current grid cell. Another important assumption is the global orientation,

so that all the agents agree on the directions, which means the North axis refer to the same axis for all the ants. For this type of ants, there is no memory used and there is no global information other than the encoded constant number of states.

The other typical model of the mobile agents is the TM which takes advantages of the agents' local memories. One of the ways to implement the TM is using an infinitely long tape, with which the ant can read information from or write onto the tape. The utilization of the tape suggests that the ants will remember their paths or at least the number of steps during the searching. Compared to the FSM, the TM are defined, in addition to the set of states, the initial state and the transition function above, with the tape alphabet T . Therefore the decisions of next step will also depend on the contents of the memory representing tape. In the real world, one cannot assume to have unlimited memory, and people often in favour of the methodologies using fewer memories. Thus, a more realistic and practical model of the TM is to maintain a counter as information stored on the tape instead of the whole path.

2.7 The Global Control of the System

Another angle to look at the design of the ANTS problem is the global control of the whole system. The synchronicity of the processes is crucial to the entire problem. Whether the system is the synchronous system or asynchronous system will end up with a differences in the processes; however, it has few influences on the complexity of the algorithm [33]. The time requirements of these searching

algorithms are often computed as the number of rounds needed to finish the process. However, the definitions of the rounds are different in synchronous systems and asynchronous systems.

In synchronous model [19, 33], every ant move at the same time and each only takes a single step during each round. However, because all agents act simultaneously, there are chances of having collisions and duplication. The ant itself has no ability to observe the entire system and no awareness of what other ants are doing. Furthermore, the ants have no different initial knowledge and would follow the same pattern to perform their searching walks. Therefore, if two ants start the search at the same time, then both of them will operate in the same way. There is no way to separate them by themselves as long as they leave the nest together. One way to solve this problem of redundancy is to introduce an emission scheme to the algorithm. The emission scheme is a mechanism that decides the order and rate of ants leaving the nest. By changing the rate, one can change the time period between the releases of two agents to separate them. By using the emission scheme, the time it takes for k ants to leave the nest is a function $f(k)$ depending on k which must be greater than k .

In the asynchronous model [21, 22, 33], since the agents are most likely to take steps at different time, there is no need for the emission scheme to prevent duplications. To simulate an asynchronous system, the movements of the ants can be controlled by an adversarial scheduler. Only one single ant, chosen by the scheduler, makes a single move at a time. This process is repeated until one of the ants finds the treasure. With this set-up, it is guaranteed that no two ants will move at

the same time. Compared to the synchronous model, the round is defined in a different matter. One asynchronous round ends after every ant has made at least a single step since the last round. The initial round is where no ant took any step yet. Regardless of the difference in the round definition, the time complexity of both synchronous model and asynchronous model have the same lower bound of $O(D + D^2/k)$ in the ANTS problem [33].

2.8 The Fault-tolerance Design

When discussing an algorithm, one big concern is the ability of this algorithm to handle the failures occurred. In this section, two methods of achieving fault-tolerance in the ANTS problem are discussed. Langner *et al.* [34] using finite automata controlled ants provided a distributed protocol. There is an error checking mechanism which is capable of detecting and overcoming failures up to a maximum number f . The time requirement of this algorithm is $O(D + D^2/k + Df)$. In this mode, when agents are located in the same cell, they are allowed to exchange constant size messages. The operating system is synchronous. As discussed in the previous section, One-time unite is the discrete round that all ants take actions instantaneously and simultaneously. To simulate the failure events in the real world, an adaptive off-line adversary is used with the function specifying the failure time for each agent. The whole fault-tolerance design relies on an innovative key term, the “giants”, which is a cluster of x agents. All agents belong to the same *giant*

will perform as a union and act together during the execution. If there are g Giants used, then the system can tolerate $k/(g+3) - 1$ failures where k is the total number of agents.

Different from the previous method, Afek *et al.* [36] provide another way of reaching fault-tolerance by using pheromones instead of duplicated agents. The time complexity is $O(D + D^2/(k-f) + Df)$ by using their fault-tolerant FSM algorithm and the algorithm can tolerate up to f ($f < k$) ants fail during the search. The key concept is adding another phase, so-called verification phase, after the exploration phase in the algorithm from their previous studies [33]. An additional pheromone is emitted as an indication of the full exploration. During the verification phase, the mobile agents will traverse the layers and verify these indication markers. If there is any missing marker due to the failure of an agent, the checking agents will re-explore the incomplete layer to fix the problem. The utilization of the pheromone remains constant order. They proved that the lower bound $O(D)$ of pheromones is valid when using this fault-tolerant algorithm in both synchronous and asynchronous models.

2.9 The Presence of Obstacles

A real environment is not as simple as a two-dimensional flat plane. The ants need to overcome barriers of different sizes, shapes, and orientations. Since the ants have no information about the environment and a limited visibility, one way of looking at the ant foraging in a grid with obstacles is relating it to the searching in

a maze. Budach [37] gave proof that no single automaton can search all mazes. It was shown by Blum and Kozen in [38] that a maze can be covered by an automaton with two pebbles, and there are two automata which together can search all mazes. Hoffmann in [39] proves that an agent cannot traverse the entire labyrinth with a single pebble. Further, [40] offer a survey of two-dimensional automata theory while an introductory survey of non-heuristic algorithms on robot navigation in unknown terrains the reader is referred to [41].

In another study, Hoff *et al.* [42] simulate a continuous world with realistic physics and visualization to test their foraging algorithms. They illustrated two foraging algorithms to deal with the obstacles: virtual pheromone and cardinality. The two algorithms allow communications between mobile agents to perform collaborative work in the environment with and without obstacles. Instead of physical markers, they rely on simple, local, low-bandwidth, direct communication between the agents. A recently published research by Keller *et al.* [43] presented the method to overcome obstacle problem using the idea of Guides and Explorers similar to the model in [20]. In that study, they use the asynchronous model of the ANTS problem in [22]. No pheromones are used in the model presented by Keller *et al.* [43]; As they pointed out in the paper, the reason they did not use pheromones is that many species of ant in the real world do not use pheromones but use direct interactions through their antennae [44].

Different from the previous methods used in [42, 43], in this thesis, the pheromones are used to solve the problem of searching within the environment containing obstacles. There are different types of obstacles discussed in this thesis: convex

obstacles, non-convex (concave) obstacles and randomly placed obstacles. Different algorithms are given with respect to different environment set-ups.

Chapter 3

Foraging in the Environment without Obstacles

This chapter presents a simple algorithm using k ants searching for a treasure in a terrain without obstacles. The ants perform one direction spiral search starting from the central nest. After released from the nest, the ants do not have any sense of direction. The decisions are only made based on the local view of the neighbouring cells. In addition to the basic algorithm, a global termination process is added to let all ants stop the searching. In the last section of this chapter, multiple nests are used instead of a single central nest. The experimental results suggest that using multiple nests is relevant to having obstacles, since the pheromones released by the ants from one nest become barriers to the ants from another nest.

3.1 Ant Foraging with Multiple Ants

Throughout this section, k ants are initially located in a central nest s and the food is located at distance D from the source. The system is assumed to be large enough to allow the ants to finish full cycles of the spiral search. To achieve this assumption, a wrap-around system is used that when ants cross the boundary on one side they will come back from the opposite side.

3.1.1 Simple Search Algorithm

The first algorithm improves the algorithm of [1] in which the ants do not need to know the direction of the nest s . Consider the following algorithm for finding food located in a cell.

Algorithm 3.1 SimpleSearch

```
1: if  $LeftCell \neq visited$  then  
2:   turn left  
3:   move forward  
4:    $Cell = visited$ 
```

FIGURE 3.1: Movement of multiple ants using simple search algorithm, Algorithm 3.1.

In Figure 3.1, the ants are released from the central origin, the nest, and following Algorithm 3.1. The fundamental idea of this algorithm is the left-hand rule. Whenever the ants see an unvisited cell on the left, they turn left and take the cell.

Lemma 3.1. $\Omega(D^2/k)$ is a lower bound on the search time.

Proof. For the lower bound, consider the case where the target is located in a rhombus centred at the nest with grid radius D . The total area is $2D^2$, and therefore, with k ants, the time the algorithm takes is at least $2D^2/k$. $\square$

Lemma 3.2. When $k \geq 1$, Algorithm 3.1 ensures that at least one ant finds the food in time $O(D^2/k + D)$, where the distance D and the number k of ants are unknown to the ants.

Proof. To prove the upper bound, the total area the ants need to traverse is $4D^2$ which is a square with a edge of $2D$. More precisely, it can be shown that if the distance from the nest to the target is D then Algorithm 3.1 takes $4D^2/k + D$. Therefore the approximation ratio of this algorithm is 2. $\square$

Lemma 3.3. *At least one of the ants will find the food.*

Proof. Assume that the food was not found. This means that the ant has skipped the target cell located between the cells with a distance greater than $D - 1$ and less than $D + 1$. However, the ant would have made a left turn when covering the layer at $D + 1$ since it checks to see if layer at distance D has been covered. This is a contradiction. Therefore, the ant will always cover the target cell at distance D . $\square$

3.1.2 Simple Search Algorithm with Global Termination

The algorithm in the previous section ensures that at least one ant finds the target. This section shows an algorithm with a minor modification that allows the system to reach a global termination. An additional notification phase is introduced into the previous algorithm. In the system with multiple ants, once an ant finds the food, it notifies the other ants to make them stop the searching process and terminate. Algorithm 3.2 gives the detail of this notification algorithm.

Algorithm 3.2 Notification

```
1: if FoundFood then  
2:   while  $LeftCell = \text{visited}$  do  
3:     move forward  
4:      $Cell = \text{terminated}$   
5:   move forward  
6: else  
7:   SimpleSearch Algorithm 3.1
```

FIGURE 3.2: Movement of multiple ants using simple search algorithm with notification, Algorithm 3.2.

Lemma 3.4. *Algorithm 3.2 ensures that all ants find the food in time $O(D^2/k + D)$, where the distance D and the number k of ants is unknown.*

Proof. The time requirement of the searching phase is exactly the same as the previous algorithm in Lemma 3.2, which is $O(D^2/k + D)$. The newly introduced notification phase only requires a constant time $O(1)$. Therefore, overall time complexity is still $O(D^2/k + D)$. $\square$

3.2 Ant Foraging with Multiple Nests

This section investigates the case using more than one nests. With different initial positions, some ants will be closer to the target and have more possibilities to discover it. The ants are still identical following the same algorithm and using same pheromones. However, the difficulty caused by using multiple starting nests is that the pheromones released by some ants would be roadblocks for other ants.

First, the simple search algorithm Algorithm 3.1 from the previous section is applied to the case with two nests, shown in Figure 3.3. However, this algorithm does not work in this situation, since the ants only check the left cell and the front cell while they are travelling. This is not a problem when all ants start at the same location and turn to the same direction, but, when they start at different locations the previous checking condition is not enough. As shown in Figure 3.3, the ants will end up with keep travelling along the same lines since the left cells are covered by the ants from the other nest. Thus, a modification to the simple search algorithm is required to handle the case where ants start in nests at different locations.

FIGURE 3.3: Movement of multiple ants from multiple nests using simple search algorithm, Algorithm 3.1.

This new algorithm is faster than the previous algorithm. By checking the left, front, and right cells, the ants can avoid exploring cells that have already been covered after released from the nests. The algorithm is described in Algorithm 3.3 as follows.

Algorithm 3.3 EfficientSearch

```

1: if LeftCell  $\neq$  visited then
2:   turn left
3: else if FrontCell  $==$  visited then
4:   turn right
5: move forward
6: Cell = visited

```

Figure 3.6 shows an example using the EfficientSearch algorithm, Algorithm 3.3.

In this example, more than one ant has been able to successfully locate the target.

A new concern arises, specifically, the ants will get stuck when surrounded by pheromones. This is because the ants come from different nests and, therefore,

from different directions. This happens at a stage where the entire environment is almost visited. The ants will either discover the target and terminate, or be surrounded by pheromones and get stuck in a 2-by-2 square. One way to prevent the ants to be stuck is to add some randomness to the algorithm when they are surrounded by pheromones. The next algorithm showed below is a modified version that follows the above idea.

FIGURE 3.4: Movement of multiple ants from multiple nests using efficient search algorithm, Algorithm 3.3.

In this new algorithm below, Algorithm 3.4, when the ant finds out that all the neighbour cells are visited, it randomly moves to one of the four neighbours. In this way, since the decision is random, the ant will not follow the same path each time preventing it from being stuck. The cells that are marked by pheromones actually function as obstacles. The way the ant tries to avoid the visited cells is equivalent to the way it uses to walk around obstacles. This is associated with

Chapter 6 where a updated version of this type of spiral searching algorithms is used to deal with the randomly placed obstacles.

Algorithm 3.4 EfficientSearchModified

```

1: if Surround by Pheromones then
2:   move to one of the four neighbours
3: if LeftCell  $\neq$  visited then
4:   turn left
5: else if FrontCell  $==$  visited then
6:   turn right
7: move forward
8: Cell = visited

```

Two simulated results are shown in Figure 3.5. Two and four nests are used respectively. Several ants have found the target and terminated, while the rest ants are still walking around randomly. If the simulations run long enough, all ants will eventually arrive at the target.

FIGURE 3.5: Movement of multiple ants from multiple nests using modified efficient search algorithm, Algorithm 3.4.

FIGURE 3.6: Movement of multiple ants from multiple nests using modified efficient search algorithm, Algorithm 3.4, for a long time.

3.3 Conclusion

This chapter introduces four algorithms for ant foraging in two dimensional grid without obstacles. Algorithm 3.1 is a simple algorithm that allows at least one ant to discover a target located at a distance D away from the central nest. Multiple ants can be used and released from the same location performing left-hand spiral walks. A notification phase is added to Algorithm 3.2 to allow the global termination. In the two algorithms, the ants do not need to know the direction of the nest and do not need any sense of direction. The decision-making is only based on the presence of the pheromones in the neighbouring cells. Both algorithms have a time complexity of $O(D^2/k + D)$.

Two modified versions of the simple algorithm are examined to deal with the target searching problem when having more than one nest. Because of the unpredictable pheromones left by different ants from different nests, the situation is similar to the case of having obstacles. At least one of the ants can find the target by following Algorithm 3.3 or Algorithm 3.4. The time complexities of these two algorithms are not as obvious as the previous cases. The following chapters will show the searching problem of having obstacles in the environment.

Chapter 4

Zig-Zag Foraging in the Presence of Large Obstacles

This chapter presents algorithms that solve the problem of searching in an environment with large objects. It is different from the wrap-around environment in the previous chapter, the environment in this chapter is a finite two-dimensional grid bounded by walls. Only a single ant is used in the algorithms and it has the ability to see its eight neighbour cells and recognize the walls.

There are two conditions which have to be satisfied in order to let the main algorithms work properly. The assumptions are:

1. the ant starts from one of the four corners in an N by N grid, and
2. passages between the obstacles are to be separated by at least five cells.

For simplicity, the ant always starts at the south-west corner in all the examples given in this chapter. In addition to the above assumptions. The obstacles must

be in the target categories which will be discussed later in this chapter. The ant can traverse the entire terrain as long as these assumptions are true. The time complexity of the main algorithms is $O(ND_{max})$ where N is the size of the environment, and D_{max} is either D_{object} , the horizontal distances between the western wall and the furthest edge of the most eastern object, or D_{target} , the distance between the western wall and the target. More precisely, $D_{max} := \max\{D_{object}, D_{target}\}$. The consumption of pheromones (i.e., the amount of pheromones used) for is $O(A)$ where A is the total area of the unoccupied cells.

4.1 Zig-Zag Foraging

In the basic foraging algorithm, the ant leaves pheromones on newly discovered cells. The intuition behind this Zig-zag approach is to find a way to allow the ant to walk around the obstacles and be able to continue visiting the undiscovered cells. The function of the pheromone is the same as the simple algorithms in the non-obstacle case. When dealing with the presence of the obstacles, the new Zig-zag algorithm works as follow.

- Recursively **do**
 1. Zig-zag walk until you hit an object
 2. Go around the object using the left-hand rule until you return to start position;
 3. **do** zig-zag walk around the object filling up new cells.

The traversal of the single ant is depicted in Figure 4.1 in a terrain with obstacles. The two fundamental procedures employed are FillUp and ExploreObject for covering all adjacent cells and exploring new objects, respectively. The movement of the ant in the terrain will be explained where the main algorithms are introduced. The main difficulty of this approach is to show that the ant never gets lost or stuck in its foraging and never misses any cell of the grid.

FIGURE 4.1: Ant traversal of a terrain with obstacles.

4.2 Main Algorithms: Fillup and ExploreObject

The fundamental idea of the algorithms in this section is to cover the entire system by filling up cells from one side to the other horizontally (column by column). The main algorithm consists of two procedures: ExploreObject and FillUp. ExploreObject (Algorithm 4.1) is called upon when the ant sees a new object for the first time. The ant will hold its "right hand" or "left hand" on the wall and walk along

the object until it finishes a cycle and goes back to where it starts the exploration. At that point, it terminates the EXploreObject procedures and executes FillUp procedure (Algorithm 4.2), which involves covering all the adjacent cells from top to bottom.

During the Fillup processes, if a new object is found, the ant will change the state and execute ExploreObject again. The entry point is marked with a particular configuration, an empty cell at the corner surrounded by pheromones, and when the entry point is reached after outlining the object, it will switch back to the FillUp procedure. However, after simulating the algorithm, it is found that the entry point marker is not necessary. If the ant encounters a pheromone while walking along the boundaries, it implies that the ant is back to the point where it left over from the FillUp process, as the filling-up covers column by column from one side to the other globally. The algorithm terminates when the ant finds the target or has searched entire area.

Algorithm 4.1 ExploreObject (Right-hand version)

```

1: if FrontCell == EntryMarker/visited then
2:   turn left
3:   move forward
4:   Fillup Algorithm 4.2
5: else
6:   if FrontCell == Object then
7:     while FrontCell == Object or RightCell != object do
8:       turn left
9:     else if FrontCell and LeftCell and RightCell != object then
10:      turn right
11:   move forward

```

Algorithm 4.1 is the right-hand version of ExploreObject procedures during which the object is always on the right side of the ant. The ant can encounter a new

object when travelling either from south to north or from north to south, therefore, the object can be either on its right side or on its left side. The left-hand version is similar, but, with the directions flipped. A variable can be used, requiring just constant memory, to record whether the object is on the right or left when first discovered.

Algorithm 4.2 Fillup procedure

```

1: if FrontCell or RightCell or LeftCell == Object then
2:   ExpObject Algorithm 4.1
3: else if FrontCell == Wall or visited then
4:   if LeftCell == visited then
5:     turn right
6:     move forward
7:     turn right
8:   else if RightCell == visited then
9:     turn left
10:    move forward
11:    turn left
12: else if LeftCell and RightCell == empty then
13:   if Rear – LeftCell == visited or object then
14:     turn left
15:     while FrontCell != visited and LeftCell == object do
16:       move forward
17:   else if Rear – RightCell == visited or object then
18:     turn right
19:     while FrontCell != visited and RightCell == object do
20:       move forward
21: else
22:   move forward

```

In Algorithm 4.2, there are three cases handled. When the ant discovers a new object, the ExploreObject procedure will be called. If the ant hits a wall or a visited cell, it reaches a turning point where it needs to move to the next column and walking back. This gives the zig-zag walking pattern. The last special case deals with the situation when there are empty cells on both sides. Since the ant can see all eight neighbouring cells, it can check the rear-left and rear-right cells

to determine the next step. This process makes sure the ant finds the most-west empty cell where it should start from. If none of the above cases happens, the ant simply moves forward along the same column. At all steps, whenever the ant sees the target in the neighbours, it stops at the cell next to the target and terminates.

4.3 Intuitive Discussion of Correctness

This section gives an intuitive discussion of the correctness of this algorithm and shows that it does solve the ANT problem in the environment with large obstacles which belong to particular categories. In the following subsections, this algorithm is tested by using different types of obstacles: rectangular objects, convex monotone objects, and objects with cusps. As mentioned before, the ant can start to search at any corner of the grid, but, here the south-west corner is used as the origin and the initial direction of the ant is facing to the north.

4.3.1 Single Rectangular Obstacle

To start with the simplest problem, the algorithm is tested in the case where there is only a single rectangular obstacle. The rectangular object is determined by four points: (x_1, y_1) , (x_1, y_2) , (x_2, y_1) and (x_2, y_2) in the two-dimensional N-by-N grid.

FIGURE 4.2: Ant traversal of a terrain with the presence of a rectangle.

Lemma 4.1. *The ant will traverse all cells with x -coordinate ($x < x_1$), as well as the cells either with $x = x_1$ and $y < y_1$ or with $x = x_1$ and $y > y_2$*

As shown in Figure 4.2, the ant starts at the south-west corner and walk along the western wall. It performs a zig-zag walk following Algorithm 4.2. Whenever the ant reaches to a wall, it will take one step to the direction that has not been visited before. After that, the ant will turn against the wall and continue to search on this new patch until it hits a wall again. The ant will stop this zig-zag walk pattern when it discovers an obstacle on one of its eight neighbours. If x_1 is even, the ant will cover the cells with $x = x_1$ and $y < y_1$ while moving from south to

north; whereas, if x_1 is odd the ant will traverse the cells with $x = x_1$ and $y > y_2$ in the opposite direction. This is saying that the ant will always discover the single rectangular obstacle at one of its corners: (x_1, y_1) or (x_1, y_2) .

Lemma 4.2. *The ant will traverse the entire perimeter of the rectangle.*

Once the rectangle is discovered, the ant switches to Algorithm 4.1 either using the Right-hand-rule or Left-hand-rule depends on current walking direction. By following the algorithm, the ant keeps the object always on the same side and walk along it until it goes back to the start point. Since the ant can see its eight neighbours, it will never lose track of the object. At the places like $(x_2, y_1 - 1)$ or $(x_2, y_2 + 1)$, for examples, when both left and right cells are empty, the ant will check the "rare-left" cell and the "rare-right" cell to look for the object. Then, the ant will turn to the direction having the obstacle and continue from there. The procedures of exploring the object terminate when the ant is back to the spot where it starts the exploration (finding a pheromone in front). By the end of this procedure, the ant will have traversed the entire perimeter putting a layer of pheromones around the rectangle and been ready to continue to perform zig-zag walks.

Lemma 4.3. *The ant will traverse the cells that are on the north and on the south to the object: i.e. the cells in the set $S = \{(x, y) | x_1 < x < x_2, y < y_1 \text{ or } y > y_2\}$.*

After explored the object, the ant changes back to Algorithm 4.2 and continue the filling up by zig-zag walks for the cells between the walls and the object. When the ant reaches a corner where there are no pheromones on both sides, the ant will

check the neighbours locally searching for a visited cell. If there is a pheromone on the "rare-left" cell, then the object is on the left side and the ant will turn left and keep walking until encounter a visited cell in front. The ant now finds an appropriate place to continue the zig-zag walk searching. Since the goal of the main algorithms is to keep filling up the environment from the most west to the east globally, the previous steps make sure the ant finds the most west start point and does not miss any cells.

Lemma 4.4. *The ant will traverse all cells with x -coordinate $(x > x_2)$.*

After traversing the cells between the objects and the walls, the ant will traverse all the cells in-between the north wall and the south wall. Without any barriers between the two parallel walls, the ant will simply walk straightly back-forth column by column as in Lemma 4.1. As before, when the cells on both left and right side are unvisited, the ant will move horizontally to the north determined by the existence of pheromones on neighbouring cells. This makes sure no cells (x, y) left uncovered if the cells $(x + 1, y)$ are already visited.

Lemma 4.5. *The time complexity of this algorithm in the presence of a single rectangle is $\Omega(ND_{max})$, where N is the size of the environment, and D_{max} is the maximum value between: D_{object} and D_{target} . D_{object} is the horizontal distances between the west wall and the furthest edge of the object, which is actually x_2 in the given example. D_{target} is the horizontal distance between the west wall and the location of the target x_{target} . The origin, $x = 0$ and $y = 0$, is the north-west corner of the terrain.*

As shown through Lemma 4.1 to Lemma 4.4, no cells are missed while the ant travelling from the north to the south globally. There are two different cases needed to be considered to calculate the time complexity for this zig-zag searching algorithm:

If $D_{object} < D_{target}$, when the ant discovers the target at x_{target} , it must have traversed all cells with $x < x_{target}$. Precisely, the total area the ant covered is

$$\begin{aligned}
Area_1 &= N(D_{target} - 1) - A_{obs} + y_{target} \\
&\text{or} \\
&= N(D_{target} - 1) - A_{obs} + (N - y_{target}) \\
&\approx O(ND_{target} - A_{obs})
\end{aligned} \tag{4.1}$$

where A_{obs} is the total area of the obstacle. The ant moves with a speed of 1, therefore the upper bound of time requirement is $O(ND_{target} - A_{obs})$.

If $D_{object} > D_{target}$, the worst case is the target is located in the area that will be covered later. Depending on the parity of x_1 , the ant will either search on the north side to the object first or the south side. After the first part is done, the ant will walk around the object and arrive at the other side, and start to traverse this part from the most west unvisited position, as explained in Lemma 4.3.

In the one rectangle example, assume $x_1 < x_{target} < x_2$ and $y_{target} > y_2$. In addition, x_1 is even as shown in Figure 4.3. Before arriving at the corner of the rectangle, the ant must have covered an area of Nx_1 . After that, the ant will first finish searching on the area of the cells having $y < y_1$, which is $(x_2 - x_1)y_1$, and then walk a distance of roughly the half of the perimeter, $(x_2 - x_1) + (y_2 - y_1)$, to arrive

at the north part. Finally, the ant has to travel an area of $(N - y_2)(x_{target} - x_1)$ to find the target. Overall, the area covered by the ant in this case is

$$\begin{aligned}
Area_2 = & N \times x_1 + (x_2 - x_1) \times y_1 \\
& + (x_2 - x_1) + (y_2 - y_1) + (N - y_2) \times (x_{target} - x_1)
\end{aligned} \tag{4.2}$$

Since x_2 can be much greater than x_{target} , and y_1 can be much greater than $N - y_2$, Equation 4.2 can be reduced to

$$Area_2 \approx ND_{object} - A_{obs} \tag{4.3}$$

As a result, the overall upper bound is the maximum of Equation 4.1 and Equation 4.3. Furthermore, A_{obs} can be much smaller than ND_{max} . Therefore, the upper bound of the time complexity of the ant foraging using Algorithm 4.1 and Algorithm 4.2 in the presence of a rectangular obstacle is $\Omega(ND_{max})$.

FIGURE 4.3: Ant traversal of a terrain with the presence of a rectangle.

4.3.2 Multiple Rectangular Obstacles without Overlapping

This subsection shows that the main algorithm: Algorithm 4.1 and Algorithm 4.2 is able to handle the situation with multiple rectangular obstacles. The ant is designed to be able to switch between the two states: the ExpObj and Fillup algorithms, upon specific conditions. With the presence of multiple rectangular obstacles, the ant is still able to traverse the entire system, including the spaces in between the obstacles as long as they are more than five cells wide. An example with four parallel rectangles is used to explain the whole process. Eventually, the ant terminates either by finding the target or arriving at the eastern wall of the system.

Suppose the objects are $\{rect_1, rect_2, \dots, rect_k\}$ and the coordinates of a rectangle i where $(1 \leq i \leq k)$ are $(x_{rect_i,1}, y_{rect_i,1}), (x_{rect_i,1}, y_{rect_i,2}), (x_{rect_i,2}, y_{rect_i,1})$ and

$(x_{rect_i,2}, y_{rect_i,2})\}$. As mentioned in the beginning of this chapter, an important assumption is that the passages between the horizontal edges of any two objects are at least five cells. Two cells needed for the explorations of the passages of the two neighbouring obstacles, and another one is needed if the ant has to walk around the object to find the west-most possible start point to continue the zig-zag walk, as explained in Lemma 4.3. Two more cells are required to allow the ant to turn and follow the zig-zag pattern. Although it is possible to let the ant recognize the situation that two objects are too close to each other and only have a one cell wide path between, and let the ant walk straight through that path, but it will require a design with a smarter ant and additional memory. Therefore, for this zig-zag algorithm, the assumption stays the same and the ant keeps following the same walking pattern.

Lemma 4.6. *The ant will traverse all cells (x, y) with $x < \text{Min}\{x_{rect_i,1}\}$, and the entire perimeter of a rectangle j with $x_{rect_j,1} = \text{Min}\{x_{rect_i,1}\}$.*

It is similar to Lemma 4.1 and Lemma 4.2 for a single rectangle. After traversing c columns, $c < \text{Min}\{x_{rect_i,1}\}$, the ant encounters the west-most obstacle, as before, assuming the ant starts from the south-west corner globally. In Figure 4.4, $Rect_1$ is discovered first. By switching to the explore object procedures, Algorithm 4.1, the ant traverses the entire perimeter of $Rect_1$. When the exploration terminates, the ant is back to the fill-up state and continue with Algorithm 4.2 from where it is left, $(x_{rect_1,1}, y_{rect_1,1} - 1)$.

FIGURE 4.4: Ant traversal of multiple rectangles environment

Lemma 4.7. *The ant switch between Algorithm 4.1 and Algorithm 4.2, based on the discovery of new objects and the termination of the explorations around them. In this way, the ant is able to recognize all obstacles and explore their boundaries.*

Whenever the ant sees a new obstacle in one of its eight neighbour cells, it switches to the exploration state and walks along the perimeter of that obstacle. This exploring process, Algorithm 4.1, makes sure that the ant can distinguish new objects and objects which are already explored. During the processes, the ant is putting a layer of pheromones around the object. As the fundamental design of the pheromone model, the ant always avoids going back to the visited cells. Therefore, the layers are barriers preventing the ant from exploring the obstacles have already been discovered.

The order of how the ant discovers the objects in the environment depends on how the objects are distributed. Figure 4.4 are two examples with four rectangles. Two different cases are compared: one is traversing the second west-most rectangle, $Rect_2$, first then $Rect_3$, and the other is traversing $Rect_3$ first then $Rect_2$.

As shown in Figure 4.4, in the left example, $x_{rect_{1,2}}$ of the west-most rectangle $Rect_1$ is less than the $x_{rect_{3,1}} - 1$ of the third west-most rectangle $Rect_3$. After explored $Rect_1$, while walking in the zig-zag pattern, the ant runs into $Rect_2$ at $x_{rect_{2,1}}$.

In the second example in Figure 4.4, $Rect_1$ is extended such that there is only one column between $Rect_1$ and $Rect_3$ which is covered by the exploration, i.e. $x_{rect_{1,2}} + 1 \geq x_{rect_{3,1}}$. In this case, $Rect_1$ becomes a local barrier that separates the columns into two parts. Different from the order in the example on the left, the ant discover $Rect_3$ before $Rect_2$, since $x_{rect_{3,1}}$ is a local minimum in the south part formed by $Rect_1$. After traversed $Rect_3$, the ant passed the $Rect_1$ barrier and is able to reach to $Rect_2$. In addition, if $Rect_1$ is long enough, so that $x_{rect_{1,2}} \geq x_{rect_{4,1}}$, then $Rect_4$ would be discovered before $Rect_2$.

In general, each rectangular obstacle is a local barrier that breaks the columns located into two parts vertically. Depending on which part the ant is walking on, it always discovers the obstacle with a local minimum x-coordinate first. Eventually, all the objects will be discovered one after another, no matter in which order, and the ant will never re-discover the obstacles that have already been visited.

Lemma 4.8. *The ant traverses all cells among the rectangular obstacles and between the walls of the terrain.*

After the last exploration and before the next exploration round, Algorithm 4.2 is used. During the fill-up procedures, the ant performs zig-zag walks on the unvisited cells inside the environment. The ant walks column by column from the

west to east. The columns are cells between two walls, or between a wall and a pheromone layer, or between two pheromone layers of the explored obstacles. The fill-up processes of the above example using four rectangular obstacles are shown in Figure 4.5.

As explained in the previous section of a single rectangle case, whenever the ant reaches a corner, it will move to the west-most point possible on the same row and start to fill up again from that point. The ant does not have the global orientation, but, this can be overcome by viewing on the neighbouring cells. The direction, left or right, has the pheromone agrees with the global west which is side the ant started from. The whole algorithm terminates when the ant finds the target or arrives at one of the two corners on the east of the terrain.

FIGURE 4.5: Ant traversal of multiple rectangles environment

4.3.3 Complex Obstacles

In this section, the algorithms are applied to the systems in the presences of more complex obstacles rather than rectangles. Similar to the previous section, this section gives heuristic explanations of how the algorithms work in each situation with different shapes of obstacles. There are two examples of having convex obstacles and two more examples of having obstacles with “cusps” which is a characteristic shape defined later in this chapter.

4.3.3.1 Convex Obstacles

The first type of the complex obstacles is the convex object. For a convex object, the function that defines the movement of the ant along the perimeter is monotone. In this thesis, since the obstacles always consist of size one squares in the two-dimensional grid, one can simply consider the motion functions as discrete functions. With this idea, by only focusing on the monotonic increasing and decreasing patterns, as an example, the polygon shown in Figure 4.6 is equivalent to a hexagon.

FIGURE 4.6: A convex object

In Figure 4.7, an example is shown with three convex obstacles. Similar to the case with multiple rectangles, the ant discovers the objects one by one while performing the zig-zag walk. As before, the transition from fill-up state to explore-object state is caused by discovering a new object. The termination of the exploration leads to a reverse transition that the ant continues to fill up the remaining cells. Eventually, the ant terminates by either finding the target or arriving at the other side of the environment.

FIGURE 4.7: Ant traversal of an environment with multiple convex obstacles

Experimental Claim 4.1. *The ant is able to escape from a narrow, one-column, path.*

One particular case that needs to be taken care of is when the ant walks on a column, the neighbouring columns are already visited. The ant will end up in a situation where it is standing on a cell surrounded by pheromones, which is like standing at the bottom of a pit. An example is shown in Figure 4.8. There is no empty path without pheromones for the ant to move onto. One simple solution used in the Fillup Object algorithm (Algorithm 4.2) is to turn back and

walk straight until there is at least one of the neighbour cells is empty. This will prevent the ant from being stuck inside. Since there is a way that leads the ant to the cell, if the ant walks back along its trail, there must be an escape from that cell. After escaping from the situation, as soon as the ant sees an unvisited cell, it moves onto the new cell and continues its zig-zag walks.

FIGURE 4.8: Ant escape from no empty path situation.

4.3.3.2 Obstacles with "Cusps"

To make the obstacles more complex, for the next step, a terminology from geometry is used. The "U-shape" compartment on a given obstacle is defined as *cusps*. As a singularity point in a curve, a cusp in an obstacle is where the motion function of the ant is not monotone. An example of a large object with cusps is shown in Figure 4.9. If the number of cusps on an obstacle equals to zero, then this object is a convex polygon; whereas, if it is greater than zero, then the object is a non-convex polygon. In the following examples, the main algorithm is tested with obstacles having cusps.

FIGURE 4.9: A large object with cusps

Experimental Claim 4.2. *The ant can traverse obstacles with any numbers of cusps.*

By following the right-hand-rule or left-hand-rule, this is obvious. The exploration procedure, Algorithm 4.1, is still valid for this type of obstacles.

Experimental Claim 4.3. *If the ant has covered cells around c cusps, the ant can also cover cells around $c + 1$ cusps, as long as they are not open to the west.*

The zig-zag walking pattern allows the ant to fill up the tunnels formed by cusps, as long as they are not open to the west. Suppose the ant has finished searching part of the environment and has covered c cusps. The ant continues to perform zig-zag walks, and then, encounters the next cusp which can open to the north, the south, or the east. The cusp opens to the west is a special case which will be discussed in the next section. Figure 4.10 shows an example for each of the three cases.

The processes of traversing the cells inside the cusps open to the north and to the south are similar. There are no special strategies required for them. If the next cusp is open to the north, the ant simply goes down to the bottom of the cusp, and then turns back moving away from the bottom of that cusp; whereas, if the cusp opens is to the south, the ant walks up to the "ceiling" and turn back. The ant continues the zig-zag until filling up the cusps. Eventually, after visited all cells inside the cusps, the ant leaves the cusps and keeps moving on without going back again. When the width of the cusp is odd, the ant ends up in a situation where it is inside the cusp surrounding by the pheromones. As mentioned in Claim [4.1](#), for this situation, the ant turns back and walks straight on the cells that contain pheromones. Therefore, it can always escape from cusps open to the north or to the south.

If the next cusp is open to the east, the ant moves to the most west point available inside the cusp, and starts to fill up the area from there. The turning point is where the ant detects that the left and right cells have no pheromones, but its local rare-right cell has. Therefore, the ant realizes that the object is on its right, and it turns to the right and moves until hit pheromones. This stop point is where the ant should continue to follow the zig-zag fill up procedures. In this way, the ant fills up the cusp area from the inside out.

FIGURE 4.10: Ant traversal of differently oriented cusps

The design of this algorithm leads to a problematic case if the opening of the cusp is to the west which is the direction the ant starts foraging from. This case is discussed in the following section. In the above example, the ant is able to handle one more cusp, opening to the north, east, or south, while in the state of zig-zag filling up. It has seen that the algorithm works for convex objects, with the number of cusps $c = 0$. If the ant has covered c cusps, it can also covers $c + 1$ cusps according to Claim 4.3. As a conclusion, this algorithm can solve the searching problem in the environment with the presence of obstacles, where the

obstacles are large objects with c cusps, $c \geq 0$, and the cusps are not open to the west.

4.3.3.3 Problematic Case

The one case not covered in the above section is when the cusps open to the west of the environment. Since the ant always fills up the environment from the west to the east globally, there are two consequences if the next cusp is open toward the west. Both of them will cause a failure of the main algorithm. The two situations with the west opening type cusps are shown in Figure 4.11. In one of the situations, a leftover area would remain uncovered, whereas, the other would make the ant get into a trap where it cannot escape from by using the previous strategies.

FIGURE 4.11: Ant traversal of an obstacle with a west opening cusp.

In Figure 4.11(a), when the ant finished the exploration and switch to the fill-up state at point $(x_{p1}y_{p1})$, it turned to face south and keep walking to $(x_{p2}y_{p2})$. At

there, even though the ant saw its left side is empty locally, it could not know the area is completely blocked. This information can be obtained if the ant has a global view of the environment, which also requires more powerful design. The ant would not be able to come back to fill it up once it passed $(x_{p2}y_{p2})$. It continued to fill up the area below the object with that area left unvisited. For the second case, in Figure 4.11(b), the ant walked into the cusp while perform zig-zag filling up. When it arrived at cell $(x_{p3}y_{p3})$, there is no clear escaping path, which is different from the one in Figure 4.8 and Claim 4.3. Even if the ant turns back and walks straight, it would still be bound on the other side of the cusp.

The formation of this problem is that the ant always tries to fill up the environment from one side (west) to the other (east), while the exit of the cusp is opposite to the travelling direction. Thus, the ant would either miss the area and never come back, or walk into the trap and cannot escape.

To solve this problem, the first attempt is to invent a way to allow the ant to recognize the potential problematic places. One of the characteristics of the cusps is that there are two same direction turns the ant needs to make contiguously, and at the turning points, two of the four adjacent cells are occupied by objects. This trait is used to make a pre-detection during the exploration of the object. The ant can leave a marker at these cells. The following algorithm is a modified version of the exploration process, Algorithm 4.1.

The modification in Algorithm 4.3 is adding an new variable called "inturn" to Algorithm 4.1. This variable is used as a counter to allow the ant to realize the target configurations could cause troubles. Only a constant memory space is

Algorithm 4.3 Modified ExploreObject (Right-hand version)

```
1: if FrontCell == Marker/secphe then
2:   fillupObj
3: else
4:   if FrontCell == Object then
5:     while FrontCell == Object and RightCell != object do
6:       turn left
7:       inturn += 1
8:   else if FrontCellLeftCellRightCell != object then
9:     if inturn == 2 then
10:      put special marker
11:     if inturn > 0 then
12:       inturn = 0
13:     turn right
14:   move forward
```

required to hold this variable. When the counter reaches to two, the next time the ant makes a turn, it would leave a special marker to indicate the potential problematic place. In Figure 4.12, an example is given where the green points are the indication marks made by following the new method, Algorithm 4.3.

FIGURE 4.12: The detection of the cusps using the modified exploration algorithm, Algorithm 4.3.

After having this pre-detection, two methods were tried to solve this problem. An example is shown for each method in Figure 4.13. The first method makes the ant act in the same way as what is done in the east opening cusps, like a mirror image. The ant walks inside the passage until reaches the vertical edge, and then starts to fill up from inside out. However, this breaks the rule of always letting the ant fill up from the west to the east, and may cause chaos in some configurations. The second method is to allow the ant to change the direction of the filling up process. Instead of filling up vertically, the ant will switch to horizontal filling up. However, it is hard to determine when should the ant switch back to the vertical filling up again. In conclusion, both methods require more delicate designs, a smarter ant and detail analysis. In addition, they both change the walking pattern in the filling up procedures, Algorithm 4.2. The associated code for the two methods is put in the Appendix section. The next chapter introduces a new algorithm that can easily deal with the problem caused by the problematic configurations of the obstacles, and it even has simpler rules and reduces the ambiguity in the way of how the ant encounters obstacles.

FIGURE 4.13: Ant traversal of obstacles with west opening cusps (the problematic case).

4.4 Conclusion

This chapter introduces an algorithm which consists of two parts: the exploration of the obstacles, and the filling up process. This algorithm runs at an optimal time complexity, since in order to search the entire system the ant has to visit every cell at least once. The goal is to solve this problem by using simple designs and less resources. The mobile agent used in this algorithm requires only constant memory and limited local visions.

To sum up, the ExploreObject (Algorithm 4.1) and the Fillup (Algorithm 4.2) can solve the searching problem in a finite environment with the presence of large obstacles with no cusps opened to the west. The time complexity is $O(ND_{max})$, same as in Lemma 4.5. The ant still needs to travel far enough to be able to walk around the obstacles, or to reach the target. This requires exactly D_{max} even in

the complex obstacle cases. The longest distance the ant needs to travel is the size of the finite grid, N . The consumption of the pheromones is $O(A)$ where A is the area of the two-dimensional grid minus the total areas of the obstacles. This is optimal if the goal is to let the ant traverses the entire terrain.

Chapter 5

Up-Down Foraging in the Presence of Large Obstacles

This chapter introduces a new algorithm that solves the problem left by the end of Chapter 4 in searching in an environment with large obstacles. The model used in this Chapter is exactly the same as the one in the previous Chapter. A single ant foraging in a finite two-dimensional grid bounded by walls. The ant has constant memory and can only see its eight neighbour cells locally. The ant starts at the south-east corner of the terrain and the initial direction is to the west.

The time complexity of both main algorithms are $O(ND_{max})$ where N is the size of the environment, and D_{max} is either D_{object} , the horizontal distances between the starting wall and the furthest edge of the furthest object, or D_{target} , the distance between the wall and the target. The consumptions of pheromones for both approaches are $O(A)$ where A is the total area of the unoccupied cells.

5.1 Up-Down Foraging

This new algorithm is inspired by the Zig-zag foraging. The same pattern that the ant walks straight in one direction until stopped, and then, turns to opposite direction and walks straight back is kept. However, instead of moving to a new column of cells, the ant now walks on the same column just visited. The time complexity is twice as the one of the Zig-zag foraging. In this algorithm, the time is sacrificed for reducing the multiplicity of how the ant encounters the obstacles. In the fill-up procedures, the ant has two states: up and down. Unvisited cells are only covered in the “up state”, except for the perimeters of the obstacles covered in the exploration procedures. In the “down state”, the ant simply repeats the old route in the reverse direction. Thus, new obstacles are only discovered in the “up state”. In addition, the searching direction is never changed in the “up state” and is the initial direction, therefore, the obstacles are always discovered while the ant is travelling from the south to the north. This is the key point of this algorithm that allows it works for complex obstacles with arbitrary configurations.

5.2 Main Algorithms: Up-down Fillup and Two Rounds ExploreObject

The up-down foraging algorithm in this chapter is similar to the zig-zag foraging algorithm in Chapter 4 where both consist of two main parts: the Up-down Fillup part, and the ExploreObject part. The up-down filling up procedures are used

for searching the system column by column. The ExploreObject is called upon the discovery of a new obstacle. There is an additional round of the exploration in the up-down foraging, where the ant traverses the areas next to the obstacles, especially the ones which are enclosures and partially surrounded by the obstacles.

Algorithm 5.1 is the main filling up procedures. The ant searches for target and leaves pheromones on unvisited cells while walking in the initial direction. Whenever the ant hits a wall or a visited cell, it walking back along the same column in the opposite direction. A variable “turn” is used to allow the ant to keep track of the angles it has turned. This variable helps the ant to tell the direction. If “turn” equals zero, it means the ant is facing the direction same as its initial direction, which the “North” in all examples in this chapter. There are two checking conditions in Algorithm 5.3:

The first if-statement capture the discovery of a new obstacle. If one of the neighbouring cells contains obstacles, the ant leaves a special marker to indicate the starting point of the exploration phase, and then calls Algorithm 5.2. The special marker can be another type of pheromone, or a particular configuration: an empty cell at the corner of the obstacle surrounding by pheromones. This special marker is required for the terminations of the following two exploration phases, Algorithm 5.2 and Algorithm 5.3.

The second checking condition deals with the case of going back to an already discovered obstacle where the ant has not finished the second exploration round. The existence of these unfinished obstacles is due to the circumstance that the ant finds a new obstacle while filling up the areas around the unfinished obstacle

during the second exploration round. In this case, the ant switch to Algorithm 5.3 to traverse the cells around that unfinished obstacle.

If there are no walls or visited cells in the front, the ant follows the main up-down walk rules. If the ant faces to the north ($turn = 0$) and reaches a wall or a visited cell, the ant turn back and walk until reaches another wall or visited cell. When the ant reaches a wall and faces to the south ($turn = 180$), it moves to the unvisited neighbour on its left and starts searching on the new column facing to the north. Whenever the “turn” variable is larger than 360, it is updated to the remaining after subtracting 360.

Algorithm 5.1 Up-down Fillup

```

1: if anyneighbors == object and turn == 0 then
2:   leave a special marker
3:   ExpObject Algorithm 5.2
4: else if Frontcell == object and leftcell == visited and rightcell == visited
   then
5:   turn left
6:   turn += 90
7:   SecExpObject Algorithm 5.3
8: else
9:   if (Frontcell == wall or Frontcell == visited) and turn == 0 then
10:    turn back
11:    turn += 180
12:    while Frontcell != wall do
13:      move forward
14:    if (Frontcell == wall or Frontcell == object) and turn == 180 then
15:      turn left
16:      turn += 90
17:      move forward
18:      turn left
19:      turn += 90
20:    move forward

```

In addition to the changes in the filling up process, another difference between the Up-down foraging and the Zig-zag foraging is there are two rounds of the exploration around the obstacles. Algorithm 5.2 is called upon the discovery of a new

obstacle. The function is traversing the perimeter of the newly discovered obstacle, which is the same as the function of Algorithm 4.1 in Chapter 4. However, when the first exploration phase terminates, the second exploration phase is triggered instead of the filling up procedures. In addition, the “turn” variable still keeps track of the angles turned as in Algorithm 5.1 above. The rest of algorithm is exact the same rules which make sure the ant walks along the edges of the obstacle without getting lost.

Algorithm 5.2 ExploreObject (Right-hand version)

```

1: if Found special marker then
2:   SecExpObject Algorithm 5.3
3: else
4:   if FrontCell == Object then
5:     while FrontCell == Object or RightCell != object do
6:       turn left
7:       turn += 90
8:   else if FrontCell and LeftCell and RightCell != object then
9:     turn right
10:    turn -= 90
11:    move forward

```

After the ant finishes the first phase of the exploration, it switches to the second phase, Algorithm 5.3. The goal of the second exploration procedure is to let the ant fill up all cells that are “above” the obstacle using up-down foraging pattern. Algorithm 5.3 terminates when the ant completes a full cycle and hits the special mark again. Whenever the ant sees an empty cell in the north of the environment, it performs the up-down fill-up procedure to traverse all unvisited cells along the same column. The up-down filling up procedures called by Algorithm 5.3 are pulled out as a separated algorithm, Algorithm 5.4, which is explained later. The ant can find the targeted unvisited cells under two different contentions. In one circumstance, the ant is facing to the north, and sees the front cell is empty. The

other case is when the ant is facing to the east, and finds an empty cell on its left.

In both cases, Algorithm 5.4 is called. If none of the above cases happens, the ant simply keeps walking along the edges until the starting point.

Algorithm 5.3 SecExpObject

```

1: if Found special marker then
2:   Updownfill Algorithm 5.1
3: else if Frontcell! = object and Frontcell! = visited and turn == 0 then
4:   SecUpdownfill Algorithm 5.4
5: else if leftcell! = object and leftcell! = visited and turn == -90or270 then
6:   turn left
7:   turn += 90
8:   SecUpdownfill Algorithm 5.4
9: else
10:  if All four neighbors are not objects then
11:    turn right
12:    turn -= 90
13:  else if Frontcell == object then
14:    turn left
15:    turn += 90
16:  move forward

```

Algorithm 5.4 is called by Algorithm 5.3 to cover the unvisited empty cells in the north. This algorithm is also capable of handling multiple obstacles. While the ant is walking to the north, the ant may discover a new obstacle. If an obstacle is seen in the front cell and its perimeter is not marked by pheromones, then the obstacle has not been explored by the ant yet. The ant switches to the ExpObject algorithm (Algorithm 5.2) to explore the newly discovered obstacle. After the ant finished the searching around the second obstacle, it will move back to the first one and carry on from where it left, seen in Algorithm 5.1. Each time Algorithm 5.4 is called, it only operates once, and by the end, it calls back to the second exploration phase, Algorithm 5.3.

Algorithm 5.4 SecUpdownfill

```
1: while Frontcell! = wall and Frontcell! = visited do
2:   if Frontcell == object and Rightcell! = visited and Leftcell! = visited
   then
3:     leave a special marker
4:     turn left
5:     turn += 90
6:     ExpObject Algorithm 5.2
7:   else
8:     move forward
9:   turn back
10: turn += 180
11: while Allneighbors! = object do
12:   move forward
13: turn left
14: turn += 90
15: SecExpObject Algorithm 5.3
```

5.3 Intuitive Discussion of Correctness

This section discusses the correctness of this new algorithm by showing several examples. This up-down foraging algorithm is comparable to the zig-zag foraging in Chapter 4. The two algorithms have same strategies of searching the environment column by column and exploring obstacles whenever disordered. Thus, it is obvious to see that the up-down algorithm can also solve the simple cases: a single rectangular obstacle, multiple rectangular obstacles, and convex obstacles, in Chapter 4. Therefore, the main focal point here is to explain how the new up-down walking pattern handles the cases that are hard for the zig-zag algorithm to solve. The combination of the new filling up procures and the two rounds of explorations allow the ant to search in the environment with the presences of very complex obstacles.

5.3.1 Obstacles with “West-Opening Cusps”

The first example demonstrates how this new algorithm solves the problem left at the end of Chapter 4. Figure 5.1 shows a single obstacle with one west-opening cusp. By following Algorithm 5.1, the ant first searches the terrain column by column walking in the up-down pattern until (x_1, y_1) . At that point, the ant sees an obstacle in the neighbouring cell, so it leaves a special marker to indicate the starting point. After left the marker, the ant switches to the exploration state and starts to walk along the perimeter of the object until back to (x_1, y_1) again. The termination of the first round of the exploration triggers the transition to the second round, Algorithm 5.3.

As shown in Figure 5.1, during the second round of exploration, the ant is walking along the obstacle again and traversing all cells in the north of the obstacle. In addition to the cells between the northern wall and the northern branch of the obstacle in Figure 5.1, the area inside the cusp is also covered. When the ant arrives at (x_2, y_2) , it sees the unvisited neighbouring cell and switches to Algorithm 5.4. As the ant walking along the inner edge of the southern branch, it is constantly filling up the cusp by calling Algorithm 5.4 repeatedly. Algorithm 5.3 terminates when the ant arrives at the marker again.

The ant leaves the marker and walks back along the column that has been traversed once. The rest of this searching process is to keep walking column by column. Different from the examples seen in Chapter 4, there are no areas missed during the process. The ant does not get stuck inside the cusp neither, since the ant

always walks along the edges of the obstacles. The way of this algorithm deals with the cusps in other directions is similar to this example. Therefore, the ant is able to search in the environment with the presence of obstacles with cusps by the up-down forging.

FIGURE 5.1: Ant traversal of a west opening cusp.

5.3.2 Obstacles with “Narrow Openings”

In addition to the obstacles with cusps, the up-down algorithm is able to solve the problems with another troublesome configuration: an enclosure area with a narrow opening, like a hole. Figure 5.2 shows an example of this case. By using the perimeter of the object as a guide, the ant would still be able to go back to the start point through all the time. Therefore, the ant never gets lost inside the environment. This allows the ant to handle the searching in an environment with much more complex obstacles.

FIGURE 5.2: Ant traversal of an enclosure area with narrow opening.

Experimental Claim 5.1. *The ant never gets stuck inside the areas around the obstacles.*

As shown above, the ant keeps walking along the perimeter of the obstacle, therefore, it will always go back to the starting point after completely traversing the whole perimeter.

Experimental Claim 5.2. *The time complexity is twice of ND_{max} which is $O(ND_{max})$. The whole process requires $O(A)$ pheromones, where A is the area of unoccupied cells.*

The only difference between the up-down foraging and the zig-zag foraging in Chapter 4 is that the ant covers every cell at least twice. Therefore, the time requirement is two times that of Lemma 4.5. However, the ant only leaves pheromones on the cells for the first time it walks onto them, thus, the amount of pheromone consumed is the same as of the zig-zag foraging. One pheromone on each unoccupied cell gives a total of $O(A)$ pheromones.

5.3.3 Multiple Complex Obstacles

The algorithms can also be applied to the situations with multiple obstacles. Figure 5.3 is an example with multiple complex obstacles including cusps and holes. One more thing to notice here is, since the ant does not perform zig-zag walks between the obstacles, the limit of the distance between two object edges can be reduced. One cell is required for each exploration layer and another one cell is for moving in between the obstacles. As shown in Figure 5.3, at any stages, if the ant encounters a new obstacle, it marks the point and starts the exploration (Algorithm 5.2) simultaneously. After that, Algorithm 5.3 makes sure the ant covers the cells on the north of the obstacles. The ant switches between the algorithms to handle multiple obstacles. Since the start point and the finish point is the same cell, the ant does not leave any obstacles unfinished. It can always carry on

from where it left over. Eventually, the ant stops when it finds the target or has traversed the entire environment.

FIGURE 5.3: Ant traversal of a terrain with complex obstacles.

5.4 Conclusion

The up-down foraging is introduced in this chapter as an improvement of the zig-zag foraging. Both are based on the idea of exploring the objects and filling up areas around them. The up-down foraging is able to solve the searching problem using single ant in the environment with very complex obstacles including the existence of cusps and holes, but, this algorithm is less efficient in time. The constraints of the model used by the algorithms are: the environment must be finite, and the obstacles are relatively large. The goal is to solve the searching problem by using less complicated algorithms, and simple design agent with constant memory and limited local visions. Even though the time complexity is doubled, it is still $O(ND_{max})$ and the efficiency of this algorithm is still valid.

Chapter 6

Foraging in the Presence of Randomly Placed Obstacles

This chapter presents the algorithms solving the searching problem in a completely different model from Chapter 4 and Chapter 5. The system is designed to be a wrap-around environment in which the ants can finish full cycles of the spiral searching process. In the wrap-around system (also known as two-dimensional torus), when the ants cross the edge on one side, they will come back from the other. This model is similar to the one used in Chapter 3. The algorithms in this chapter do not rely on the ants' recognition of the walls, which are necessary for the zig-zag and up-down algorithms. More than one ant can be used to localize the target synchronously. The nest locates at the centre of the grid also known as the origin. The main idea of the algorithms is to apply the spiral searching in Chapter 3 to solve the foraging problem in the presence of randomly placed obstacles.

In this environment, the obstacles are size one blocks randomly placed in the grid. For a given obstacle concentration C_{obs} , the total number of obstacles equals to the concentration times the size of the grid, $N_{obs} = C_{obs} \times N^2$. To place these obstacles, N_{obs} cells are randomly chosen to be obstacles.

6.1 Modified Simple Spiral Search

The first algorithm is a modified version of the SimpleSearch algorithm in Chapter 3 (Algorithm 3.1). An additional checking condition for obstacles is added to that algorithm. The new version of the SimpleSearch is described in Algorithm 6.1. At each step, the ant checks if its left neighbouring cell has a pheromone or an obstacle. If the cell is empty, the ant moves onto it. Otherwise, the ant checks the other neighbouring cells in clockwise, and moves onto the first one comes without obstacles.

Algorithm 6.1 ModifiedSimpleSearch

```

1: if  $LeftCell! = visited$  and  $LeftCell! = Obstacle$  then
2:   turn left
3: while  $FrontCell == Obstacle$  do
4:   turn right
5: move forward
6:  $Cell = visited$ 

```

Figure 6.1 shows an simulated example using eight ants. The ants are released from the central nest in sequence. Each time, when an ant hits an obstacle, the ant takes a turn around the obstacle and changes its direction. Eventually, at least one of the ants finds the target which is shown as a yellow square in Figure 6.1. The location of the target is randomly chosen for all simulations in this chapter.

FIGURE 6.1: Movement of multiple ants using modified simple search algorithm, Algorithm 6.1.

Experimental Claim 6.1. *At least one of the ants will reach a local termination by finding the target when the obstacle concentration is low.*

As shown in Figure 6.1, while time is increasing, the total area covered by the ants is expanding from the centre. The ants follow the rules of walking around the obstacles and avoiding the visited cells. Therefore, when the obstacles are spread wild enough, the ants will not get stuck and will always find new cells after travelling for certain distances. As a result, at least one of the ants will reach the target.

This modified simple search algorithm can solve the searching problem when the obstacle concentration is low. However, as the concentration is increasing, the system is getting denser, and the chance of the ants getting stuck and failing to find the food is also increasing. If the system is dense enough, the ants may

end up with being bounced back and forth in a local area formed by several obstacles. Therefore, one solution to prevent such situation from happening is to add randomness to the algorithm. In this way, the ants will not follow the exact same path each time, thus, can escape from those local traps.

6.2 “Pseudo-Random Walk” Search

A new version of the SimpleSearch algorithm is introduced in this section which uses the idea of introducing randomness to the simple algorithm. It can help to prevent the ants from being blocked while searching in the environment. In this algorithm, when the ants find out themselves are surrounded by the visited cells and/or the obstacles, they will move onto a randomly chosen adjacent cell without obstacle. In this way, if any ant falls into a local trap area, the path, chosen by the ant, is random each time. The ants keep moving onto different cells rather than stopping on a cell or repeating same walking paths. In Algorithm 6.2, the first if-statement handles the case where all neighbouring cells contain either pheromones or obstacles. If that happens, the ant moves onto one of the non-obstacle cells. Otherwise, it follows the same logic as in Algorithm 6.1.

Algorithm 6.2 RandomSearch

```

1: if Not any? Neighbors = empty cell then
2:   availableNeighborList = List[neighbors(!=Obstacles)]
3:   randIndex = random(0, availableNeighborList.size)
4:   aNeighbor = availableNeighborList[randIndex]
5:   move to aNeighbor
6: else
7:   Algorithm 6.1 ModifiedSimpleSearch

```

Figure 6.2 shows a simulation using Algorithm 6.2. All of the ants keep moving around and searching for the target without being stopped or blocked by the obstacles. As long as there is an accessible path to the target, at least one of the ant will eventually find it, if the simulation is run long enough. However, if the target is inaccessible, in a closure formed by obstacles, there is no way for the ants to find the target. For this kind of random walk, it is hard to predict the running time, and if the target is not available, the ants will keep moving forever.

FIGURE 6.2: Movement of multiple ants using random search algorithm, Algorithm 6.2.

6.3 “Pseudo-Random Walk” Search with Tolerance

As pointed out above, the obvious problem of the previous algorithm is the exponential or unpredictable long running time. In order to reach a termination, a

new parameter is introduced, named as “tolerance”. A constant memory is used by the ants to store a counter. Every time the ants step on an already visited cell, the counter is incremented; while, each time they explore an unvisited cell, the counter is reset to zero. If the counter stored by an ant exceeds this predefined “tolerance”, then the ant dies. Therefore, the algorithm either ends with at least one ant finding the food or all ants die after exceeding the tolerance. This new random walk search with tolerance is described in Algorithm 6.3.

Algorithm 6.3 RandomSearch with Tolerance

```

1: if counter > Tolerance then
2:   die
3: else
4:   if Not any? Neighbors = empty cell then
5:     availableNeighborList = List[neighbors(≠Obstacles)]
6:     aNeighbor = availableNeighborList[randIndex]
7:     move to aNeighbor
8:     counter++
9:   else
10:    Algorithm 6.1 ModifiedSimpleSearch
11:    counter=0

```

Figure 6.3 is a simulated example by using Algorithm 6.3 after adding the tolerance factor. The tolerance is set to 50 and eight ants are used for this example. The obstacle concentration C_{obs} is 25%. As a result, three out of the eight ants successfully find the target, while the rest die inside the system. The correlation is clear that a high tolerance should be used for a high obstacle concentration, however, it is hard to come up with an equation to get an appropriate tolerance for a given concentration. This tolerance factor should also depend on the number of ants used.

FIGURE 6.3: Movement of multiple ants using “pseudo-random walk” search with Tolerance, Algorithm 6.3.

6.4 Up-down Foraging vs “Pseudo-Random Walk”

Algorithm

In this section, the two main algorithms: the up-down foraging in Chapter 5 and the pseudo-random walk algorithm in Chapter 6 are compared. The examples illustrate that how the two algorithms work in the presence of the alternative type of obstacles. The results show that each algorithm has different advantages and should be applied to specific models. This explains why two different types of algorithms are used for the two different types of obstacles in the two different types of environments, respectively.

In Figure 6.4, the square obstacles are randomly placed in the bounded system with a concentration of $C_{obs} = 8\%$. This model is same as the one used in both Chapter 4 and Chapter 5. In both examples, the single ant has eventually located

the target by following the up-down foraging algorithm. However, when looking at the pheromone trails, it is noticed that, in the first case, the ant successfully traverses all cells until it reaches the target, whereas, in the second example, there are cells left uncovered. If the target locates in these unvisited cells in the north of the two obstacles in the left example of Figure 6.4, then the ant would fail to find the target. The distribution of the square obstacles has a crucial impact on the success of the searching. In addition, the randomly distributed obstacles may end up with very complex systems which are difficult to handle using the two rounds of the exploration processes and cause troubles in the switching between the different states of the up-down foraging algorithm. Moreover, the up-down foraging algorithm covers each cell at least twice. It requires more time to locate the target than the spiral searching algorithms in this model.

FIGURE 6.4: Up-down foraging in the presence of randomly placed obstacles.

In the next step, the test is done by using the “pseudo-random walk” algorithm in the presence of large obstacles. In this part, the simulations have the same models as in the previous sections in this chapter. The wrap-around environment is used

and the ant starts at the central nest, however, in order to compare with the up-down foraging algorithm, only one ant is used in the two examples in Figure 6.5. As a result, the ant is still able to locate the target in both cases. However, for the more complex system in the left example in Figure 6.5, there are several missed areas which would hardly ever be searched by the ant. Different from the up-down foraging algorithm, the ant does not use the perimeter of the obstacles as a guide, therefore, if the obstacles have very “rough” surfaces, the spiral searching algorithms would have difficulties to perform the searching around these obstacles.

FIGURE 6.5: “Pseudo-random walk” foraging in the presence of large obstacles.

As a conclusion, the two types of algorithms would succeed in the alternative model if the system is relatively simple having less complex obstacle configurations. However, a better choice to solve the searching problem in the presence of randomly placed square obstacles should be the spiral searching algorithms, while, the up-down foraging algorithm works better in the presence of large complex obstacles.

6.5 Modified Simple Algorithm vs “Pseudo-Random Walk” Algorithm

In this section, the Modified Simple Search algorithm (Algorithm 6.1) is compared with the “Pseudo-Random Walk” Search algorithm (Algorithm 6.2). For a given obstacle concentration, the question is which algorithm is more efficient, or in other words, which one has a lower failure rate.

As it will be seen in a later example that at high concentrations C_{obs} , both algorithms fail to finish the searching, due to the complexity of the system and the high probability of having an inaccessible target. Therefore, to make the result clear, the comparison is done by using low concentrations, $C_{obs} \leq 3.0\%$. A total number of 25 experiments are done in a wrap-around system with size $N = 100$ for each algorithm. For each simulation, the system is randomly generated, that is, for a given concentration C_{obs} , a total number of $C_{obs} \times N^2$ cells are randomly chosen to be obstacles.

In order to make the simulations stop eventually preventing them from running forever in failure cases, a tolerance factor of $T = N^4$ is used for both algorithms. Recall the definition of the tolerance factor, if after T steps the ant cannot find an unvisited cell, then it dies in the system. Each time the ant finds an unvisited cell, the counter is reset to zero. If no more alive ants left and none of the ants has found the target, then the simulation terminates and is counted as a failure. The total number of failures is counted for each algorithm and plotted in Figure 6.6. According to the plot, it is obvious that Algorithm 6.2 has a lower failure rate than

Algorithm 6.1 at low concentrations. The next step is to test the two algorithms with a larger range of concentrations C_{obs} .

FIGURE 6.6: Number of failures using Algorithm 6.1 and Algorithm 6.2 in 100×100 grid with the obstacle concentration $C_{obs} \leq 3.0\%$.

Figure 6.7 presents the results of extending the testing range of the obstacle concentrations to $0 \leq C_{obs} \leq 20.0\%$. From the plot, it is noticed that there is a plateau in the failure rate plots for both algorithms at the high concentration end. As the obstacle concentration increases, the failure rates are saturated for both algorithms. After a threshold, the failure rate is close to 100%, which implies that it is unlikely for the ants to find the target in those environments. This threshold phenomenon is what should be expected to have in this model. A mathematical derivation of this threshold is given in the next section.

FIGURE 6.7: Number of failures using Algorithm 6.1 and Algorithm 6.2 in 100×100 grid with the obstacle concentration $0 \leq C_{obs} \leq 20.0\%$.

6.6 The Obstacle Concentration Threshold

Instead of a wrap-around system, this section uses a finite $N \times N$ two-dimensional grid. If the obstacle concentration is C_{obs} , then for a randomly chosen cell, the probability of the cell has an obstacle is actually equal to the concentration, $p = C_{obs}$ and $0 < p < 1$. Therefore, a cell is occupied by an obstacle with a probability p , and it is not occupied with a probability $1 - p$. The probability p for each cell is independent. Here, the focus is on the probability for an ant will be blocked and never find the target in the two-dimensional environment. This probability should relate to the threshold has been seen in Section 6.5. To investigate this threshold phenomenon, it is started with the special case of a $2 \times N$ grid, and then proceed to the more general case of an $N \times N$ grid [45].

6.6.1 Blocking in a $2 \times n$ grid

Consider the spiral foraging algorithm on a $2 \times N$ grid with toroidal wrap around along the rows. For some p , with $0 < p < 1$, an obstacle is placed in a given cell with probability p . The question is what is the probability that foraging is blocked.

Let the vertices of the grid be (i, j) with $i = 0, 1$ and $j = 0, 1, \dots, n - 1$. For each vertex $u = (i, j)$ let $S(u) = \{(i + 1 \bmod 2, (j - 1) \bmod n), (i + 1 \bmod 2, j \bmod n), (i + 1 \bmod 2, (j + 1) \bmod n)\}$. Observe that $v \in S(u)$ iff $u \in S(v)$, moreover a set U of obstacles does not block foraging iff $u \notin S(v)$. To analyze this question, the first and second moment methods are used.

FIGURE 6.8: An obstacle at u and the associated set $S(u)$.

Theorem 6.1. *Foraging can be completed successfully for any source-destination pair in a $2 \times n$ grid iff there is no blocking pair of objects.*

Proof. If there is no blocking pair of objects then a foraging ant using a simple extension of the standard ant foraging algorithm will find the treasure for any source-destination pair. But also conversely, if a foraging ant can find the food for any source-destination pair then there cannot exist a blocking pair of objects. $\square$

The next step is to study the probability that foraging is blocked for a set of objects randomly placed in a $2 \times N$ grid. By using first moment method, the following theorem can be proved.

Theorem 6.2. *If $p < (3n - 6)^{-1/2-\epsilon}$, for a constant $\epsilon > 0$, then the $2 \times N$ grid does not have a pair of blocking obstacles w.p. at least $1 - o(1)$.*

Proof. Let p be the probability that a cell is occupied by an obstacle and $1 - p$ otherwise. Let $W = \{u, v\}$ run over pairs of adjacent cells. A pair W of adjacent cells is called *blocking* if they are both occupied by obstacles. The number of adjacent pairs is easily seen to be $3n - 6$. Let N be the random variable that counts the number of blocking pairs. For each set W of adjacent cells, let N_W be the random variable indicating with a 1 that the pair W is blocking and is 0 otherwise. Clearly, the probability that a pair is blocking is p^2 . It follows that

$$E[N_W] = \Pr[N_W = 1] = p^2 \tag{6.1}$$

$$E[N] = \sum_w E[N_W] = (3n - 6)E[N_W] = (3n - 6)p^2, \tag{6.2}$$

where the sum above runs over all blocking pairs. Now apply the first moment method.

$$\Pr[N > 0] \leq E[N] = (3n - 6)p^2. \tag{6.3}$$

This completes the proof of Theorem 6.2. □

To proceed, the second moment method is used. Recall from Chebyshev's inequality that

$$\Pr[N = 0] \leq \Pr[|N - E[N]| \geq E[N]] \leq \frac{\text{Var}(N)}{E[N]^2}. \quad (6.4)$$

Therefore, the next thing is to show

$$\text{if } E[N] \rightarrow \infty \text{ and } \frac{\text{Var}(N)}{E[N]^2} = o(1) \text{ then } \Pr[N = 0] \rightarrow 0. \quad (6.5)$$

The value resulting from these conditions will yield a lower bound for the probability p . The goal is to prove the following theorem.

Theorem 6.3. *If $p > (3n - 6)^{-1/2+\epsilon}$, for a constant $\epsilon > 0$, then the $2 \times n$ grid has a pair of blocking objects w.p. at least $1 - o(1)$.*

Proof. (Theorem 6.3) To confirm the validity of property (6.5), $\text{Var}(N)$ needs to be calculated. The approach of [46][page 19] is used to this end.

Estimating the variance is the most complex of the computations required to prove the theorem.

$$\begin{aligned} \text{Var}(N) &= \sum_W \text{Var}(N_W) + \sum_{W \neq W'} \text{Cov}(N_W, N_{W'}) \\ &\leq \sum_W E[N_W] + \sum_{W \neq W'} \text{Cov}(N_W, N_{W'}) \\ &= E[N] + \sum_{W \neq W'} \text{Cov}(N_W, N_{W'}), \end{aligned} \quad (6.6)$$

where in the inequalities above the inequality $Var(N_W) = E[N_W] - E[N_W]^2 \leq E[N_W]$ is used, which is valid since $N_W^2 = N_W$ as N_W is a random variable assuming only values 0 and 1.

It remains to compute $Cov(N_W, N_{W'})$. Since the random variables $N_W, N_{W'}$ are identical, from the definition of the covariance it is derived that

$$\begin{aligned} Cov(N_W, N_{W'}) &= E[N_W N_{W'}] - E[N_W]E[N_{W'}] \\ &= E[N_{W'}|N_W = 1]E[N_W] - E[N_W]E[N_{W'}] \\ &= E[N_{W'}]^2 F(W, W'), \end{aligned}$$

where the function $F(W, W')$ is defined as follows.

$$F(W, W') := \frac{E[N_{W'}|N_W = 1]}{E[N_{W'}]} - 1 = \frac{\Pr[N_{W'} = 1|N_W = 1]}{\Pr[N_{W'} = 1]} - 1. \quad (6.7)$$

Observe that $\frac{1}{E[N]} \rightarrow 0$, whenever $E[N] \rightarrow \infty$. Therefore if combining Chebyshev's Inequality (6.4) with Inequality (6.6) and Property (6.5), it is concluded that the desired threshold will be derived from

$$E[N] \rightarrow \infty \text{ and } \frac{\sum_{W \neq W'} Cov(N_W, N_{W'})}{E[N]^2} \rightarrow 0. \quad (6.8)$$

By combining Property (6.8) with Identity (6.7), the conditions that will determine the threshold for the probability p are derived.

Experimental Claim 6.2 (2nd Moment Convergence Conditions). *The threshold value for the radius will be determined by satisfying the conditions:*

$$E[N] \rightarrow \infty \text{ and } \frac{1}{(3n-6)^2} \sum_{W \neq W'} F(W, W') \rightarrow 0, \quad (6.9)$$

where the function $F(W, W')$ is defined in Equation (6.7). $\square$

To ensure that the first convergence Condition in (6.9) is satisfied Formula (6.2) is used. It turns out that $E[N] = (3n-6)p^2 \rightarrow \infty$ provided that $p > (3n-6)^{-1/2+\epsilon}$. The second convergence Condition in (6.9) involves a sum of the terms $F(W, W')$, where $W \neq W'$ range over all possible blocking pairs. If W, W' have no vertices in common then the random variables $N_W, N_{W'}$ are independent. Therefore $F(W, W') = 0$. It is enough to consider the case $W \cap W' \neq \emptyset$. This reduces to the case $|W \cap W'| = 1$. In the sequel it will be convenient to bound from above the term $F(W, W') + 1$ instead of the term $F(W, W')$. Indeed, to satisfy the right-hand property in Condition (6.9), it is enough to satisfy $\frac{1}{(3n-6)^2} \sum_{W \neq W'} (F(W, W') + 1) \rightarrow 0$.

Consider two pairs W, W' under the condition that W is a blocking pair. Since they share exactly one point the remaining point of W' must be adjacent to this common point. Therefore the probability that W' is a blocking pair given that W is a blocking pair is at most p . Therefore in this case it is concluded from (6.1) that

$$F(W, W') + 1 = \frac{\Pr[N_{W'} = 1 | N_W = 1]}{\Pr[N_{W'} = 1]} \leq \frac{1}{p}.$$

Now, return to the right-hand limit in Condition (6.9). First, count the number of

pairs W, W' such that $|W \cap W'| = 1$. Indeed, there exist n possible points that can be common to W and W' . Apart from these common points there remains one point that may belong to W and one other point that may belong to W' . Therefore there exist $3n$ possible pairs W, W' such that $|W \cap W'| = 1$. It is concluded that the following inequalities are satisfied

$$\begin{aligned} \frac{1}{(3n-6)^2} \sum_{W \neq W'} (F(W, W') + 1) &= \frac{1}{(3n-6)^2} \sum_{|W \cap W'|=1} (F(W, W') + 1) \\ &= \frac{1}{(3n-6)^2} \frac{3n}{p}. \end{aligned} \quad (6.10)$$

It is easy to check that the sum in (6.10) tends to 0 as $n \rightarrow \infty$, provided that $p > (3n-6)^{-1/2+\epsilon}$. This completes the proof of Theorem 6.3. $\square$ $\square$

6.6.2 Blocking in an $N \times N$ grid

For an $n \times n$ grid which is the terrain the ants walk on, there is a total of n^2 cells. Assume the obstacles are single cells and suppose there is a total of a such obstacles. Now perform the following experiment:

1. Place a obstacles at random without repetition in cells
2. Place an ant at random in a cell other than the ones chosen above.
3. Execute the spiral algorithm.

If the ratio e/n^2 is small (i.e., the obstacles are sparse), then the ant succeeds and covers the whole square finding the treasure, whereas, if the ratio e/n^2 is large

(i.e., the obstacles are dense), then the ant fails to cover the whole square and cannot find the treasure. Thus, our conjecture is that if the ratio is small the ant will find the treasure and survive else it will die! In particular, the aim is to know for what value of the ratio e/n^2 does this happen.

The questions are: Is there a value T , which is the threshold, such that i) if the ratio $e/n^2 < T$ the ant survives, and ii) if the ratio $e/n^2 > T$ the ant dies? Can those experimental results be used to determine this ratio T ?

Call two cells u, v adjacent if they touch at an edge or at a corner. The question is clearly related to *site* percolation theory in the diagonal lattice [47, 48]. There is a critical value p_c^{site} such that a sequence of adjacent objects placed at random percolates from North to South (or from East to West), if $p > p_c^{site}$, and there is no such percolating sequence, if $p < p_c^{site}$. In particular, if $p > p_c^{site}$ then an ant will be blocked by objects and cannot complete its foraging. In citations [47, 48] there are several results concerning estimates for the critical probability p_c^{site} can be found. (see also [49][Lecture 10]).

Now it is observed that if an ant is in the interior of a cycle of adjacent objects then it will not be able to complete its foraging. Therefore, the next interesting question is what is the probability that there is no cycle of adjacent objects. This can also be restated as the probability that a set of objects forms a forest. This approach can be analyzed by using the first and second moment methods. However, details are beyond the scope of this thesis. The forthcoming paper [45] is going to continue the investigation with more details.

6.7 Conclusion

This chapter mainly focuses on the searching algorithms in the wrap-around environment having randomly placed square obstacles. More than one ant are used and start foraging from one central nest. The model in this chapter is similar to the one used in Chapter 3. Three spiral searching algorithms are introduced which all are based on the SimpleSearch algorithm, Algorithm 3.1. The first modified algorithm introduced is Algorithm 6.1 which has an additional checking condition for the presence of obstacles. This algorithm allows at least one ant to find the target when the obstacle concentration is low, but may fail to work with high concentrations. The second algorithm involves a “pseudo-random walk” in order to work for the high obstacle concentrations. In Algorithm 6.2, if the ants are fully surrounded by pheromones and/or obstacles, they randomly choose one adjacent cell having no obstacles and move onto that cell. This prevents the ants from being stuck on a single cell or repeating the same walking paths. The last algorithm, Algorithm 6.3, includes an additional tolerance factor T . If after T steps the ants cannot find a unvisited cell, then the ants terminate the process and die on the current cells. This tolerance factor provides a limitation (a cap) on some levels for the running time of foraging in really complex environments having high obstacle concentrations.

Later in this chapter, Section 6.4 compares the up-down foraging algorithm with the “pseudo-random walk” algorithm to show the differences between the two algorithms. Furthermore, Section 6.5 makes another comparison between the modified simple algorithm and the “pseudo-random walk” algorithm. The modified simple

algorithm can succeed at low concentrations. However, the “pseudo-random walk” algorithm is more suitable for high concentrations. Last but not least, it has been seen that there is a threshold in the obstacle concentration from the experimental results and mathematical deductions. If the obstacle concentration of the environment exceeds the threshold, it is more likely for the ants to fail to find the target by following the spiral searching algorithms.

Chapter 7

Conclusion and Future Works

This chapter gives an overall summary of the work. More subtle, in the last section, the remaining open problems are discussed, as well as the directions of possible future studies which could be studied based upon the results of this thesis. For full details of each algorithm introduced, the reader could read the introductory and the concluding section of each individual chapter.

7.1 Conclusion

Several algorithms for ant foraging have been introduced. The algorithms presented in Chapter 3 are in the type of spiral searching algorithms. The algorithms allow multiple ants to search a target without knowing the direction of the nest in a wrap-around environment having no obstacles. A different model is used in Chapter 4 and Chapter 5. The environment becomes a finite two-dimensional terrain surrounded by walls. A single ant is involved and starts searching from

one of the corners. Chapter 4 introduces the Zig-zag algorithm that can solve the searching problem in the presence of large obstacles in simple configurations. The main algorithm in Chapter 5 is the Up-down algorithm, which can deal with more complex obstacles than those in Chapter 4. In addition, another group of algorithms is presented in Chapter 6. These algorithms are based on the spiral searching algorithms in Chapter 3. These modified spiral searching algorithms allow the ants to forage in the two-dimensional torus in the presence of randomly placed square obstacles. Thorough explanations and heuristic discussions of these algorithms are given in the related chapters. For the purpose of this thesis, the rest proofs of these algorithms are left to the future works [45], however, all simulated results of these algorithms are presented in the chapters. Additional details of the analysis will appear in the forthcoming paper [45].

7.2 Future Works

For future works [45], the plan is to give thorough proofs of all these algorithms introduced in this thesis. More tests should be done on the implemented algorithms by using randomly generated systems. An additional function should be added to produce obstacles in random configurations which can be classified into different categories.

Another focal point of future works could be the designs of the mobile agents. One could invent a much smarter type of mobile agents with more memory size and capable of complex calculation processes. The mobile agents could also have

an enhanced visibility and/or embedded GPS devices which allow the agents to have the sense of direction and aware of global orientation. Instead of ants, a grasshopper model or even a bee model could also be used to solve the searching problem. [\[45\]](#)

Abbreviations

ANTS	A nts N earby T reasure S earch
FSM	F inite S ate M achine
TM	T uring M achine

Bibliography

- [1] C. Lenzen and T. Radeva. The power of pheromones in ant foraging. In *1st Workshop on Biological Distributed Algorithms (BDA)*, 2013.
- [2] O. Feinerman, A. Korman, Z. Lotker, and J. S. Sereni. Collaborative search on the plane without communication. In *Proceedings of the 2012 ACM symposium on Principles of distributed computing*, pages 77–86. ACM, 2012.
- [3] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993.
- [4] M. Fukuda, L. Bic, M. B. Dillencourt, and J. M. Cahill. Messages versus messengers in distributed programming. *J. Parallel Distrib. Comput.*, 57(2):188–211, 1999.
- [5] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Trans. Program. Lang. Syst.*, 12(1):84–101, 1990.
- [6] R. R. Choudhury, S. Bandyopadhyay, and K. Paul. A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents. pages 145–146, 2000.
- [7] S. Appleby and S. Steward. Mobile software agents for control in telecommunications networks. *BT Technol J*, 18(1):68–70, 2000.
- [8] G. D. Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *J. Artif. Intell. Res. (JAIR)*, 9:317–365, 1998.

- [9] G. D. Caro and M. Dorigo. Mobile agents for adaptive routing. pages 74–85, 1998.
- [10] M. Dorigo, G. D. Caro, and Luca Maria Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.
- [11] G. D. Caro, F. D., and L. M. Gambardella. Anthocnet: An ant-based hybrid routing algorithm for mobile ad hoc networks. pages 461–470, 2004.
- [12] G. D. Caro, F. Ducatelle, and L. M. Gambardella. Anthocnet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *European Transactions on Telecommunications*, 16(5):443–455, 2005.
- [13] F. D., G. D. Caro, and L. M. Gambardella. An analysis of the different components of the anthocnet routing algorithm. pages 37–48, 2006.
- [14] T. R. Hsiang, E. M. Arkin, M. A. Bender, S. P. Fekete, and J. S. B. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. *CoRR*, cs.RO/0212022, 2002.
- [15] E. M. Barrameda, S. Das, and N. Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In *Algorithms for Sensor Systems - 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers*, pages 228–243, 2013.
- [16] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89, 1999.
- [17] M. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.*, pages 441–447, 1993.
- [18] M. Kao, Y. Ma, M. Sipser, and Y. L. Yin. Optimal constructions of hybrid algorithms. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on*

- Discrete Algorithms. 23-25 January 1994, Arlington, Virginia.*, pages 372–381, 1994.
- [19] A. López-Ortiz and G. Sweet. Parallel searching on a lattice. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 125–128, 2001.
 - [20] Y. Emek, T. Langner, D. Stolz, J. Uitto, and R. Wattenhofer. How many ants does it take to find the food? In *Structural Information and Communication Complexity - 21st International Colloquium, SIROCCO 2014, Takayama, Japan, July 23-25, 2014. Proceedings*, pages 263–278, 2014.
 - [21] Y. Emek, T. Langner, D. Stolz, J. Uitto, and R. Wattenhofer. How many ants does it take to find the food? In *Structural Information and Communication Complexity*, pages 263–278. Springer, 2014.
 - [22] Y. Emek, T. Langner, J. Uitto, and R. Wattenhofer. Solving the ANTS problem with asynchronous finite state machines. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 471–482, 2014.
 - [23] Y. Emek, T. Langner, J. Uitto, and R. Wattenhofer. Solving the ANTS problem with asynchronous finite state machines. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 471–482, 2014.
 - [24] D. Payton, M. Daily, R. Estkowski, M. Howard, and C. Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2001.
 - [25] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Mataric. Blazing a trail: Insect-inspired resource transportation by a robot team. In *Distributed Autonomous Robotic Systems 4, Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems, DARS 2000, October 4-6, 2000, Knoxville, Tennessee, USA*, pages 111–120, 2000.

- [26] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Mataric. Whistling in the dark: cooperative trail following in uncertain localization space. In *Agents*, pages 187–194, 2000.
- [27] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Mataric. LOST: localization-space trails for robot teams. *IEEE Trans. Robotics and Automation*, 18(5):796–812, 2002.
- [28] K. OHara, D. Walker, and T. Balch. The gnats low-cost embedded networks for supporting mobile robots. *Multi-Robot Systems. From Swarms to Intelligent Automata*, III:277–282, 2005.
- [29] E. Barth. A dynamic programming approach to robotic swarm navigation using relay markers. *the 2003 American Control Conference*, 6:5264–5269, 2003.
- [30] S. Goss and J. L. Deneubourg. Harvesting by a group of robots. *First European Conference on Artificial Life*, MIT Press:195–204, 2003.
- [31] B. Werger and M. J. Mataric. Robotic food chains: Externalization of state and program for minimal-agent foraging. *International Conference on Simulation of Adaptive Behavior*, pages 625–634, 1996.
- [32] D. E. Jackson and F. L. Ratnieks. Communication in ants. *Current Biology*, 16(15):R570–R574, 2006.
- [33] Y. Afek, R. Kecher, and M. Sulamy. Optimal pheromone utilization. In *Proceedings of 2nd Workshop on Biological Distributed Algorithms (BDA), Austin, Texas, USA*. October 2014.
- [34] T. Langner, J. Uitto, D. Stolz, and R. Wattenhofer. Fault-tolerant ants. In *Distributed Computing*, pages 31–45. Springer, 2014.
- [35] S. Koenig and Y. Liu. Terrain coverage with ant robots: a simulation study. In *Proceedings of the fifth international conference on Autonomous agents*, pages 600–607. ACM, 2001.

- [36] Y. Afek, R. Kecher, and M. Sulamy. Optimal and resilient pheromone utilization in ant foraging. *CoRR*, abs/1507.00772, 2015.
- [37] L. Budach. Automata and labyrinths. *Mathematische Nachrichten*, 86(1):195–282, 1978.
- [38] M. Blum and D. Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 132–142, 1978.
- [39] F. Hoffmann. One pebble does not suffice to search plane labyrinths. In *Fundamentals of Computation Theory*, pages 433–444. Springer, 1981.
- [40] K. Inoue and I. Takanami. A survey of two-dimensional automata theory. *Information Sciences*, 55(1):99–121, 1991.
- [41] N. S. V. Rao, S. Kareti, W. Shi, and S. S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report, No. ORNL/TM-12410, Oak Ridge National Lab., TN (United States), 1993.
- [42] N. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal. Two foraging algorithms for robot swarms using only local communication. In *2010 IEEE International Conference on Robotics and Biomimetics, ROBIO 2010, Tianjin, China, December 14-18, 2010*, pages 123–130, 2010.
- [43] K. Barbara, L. Tobias, U. Jara, and W. Roger. Overcoming obstacles with ants. In *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS)*, 2015.
- [44] N. Pinter-Wollman, A. Bala, A. Merrell, J. Queirolo, M. C. Stumpe, S. Holmes, and D. M. Gordon. Harvester ants use interactions to regulate forager activation and availability. *Animal Behaviour*, 86:197–207, 2013.
- [45] J. Czyzowicz, S. Dovrev, L. Gasieniec, E. Kranakis, and Y. Wang. Ant foraging in terrains with obstacles. In *Preparation*, 2017.

- [46] J. H. Spencer. *Ten lectures on the probabilistic method*, volume 52. SIAM, 1987.
- [47] B. Bollobas and O. Riordan. *Percolation*. Cambridge University Press, 2006.
- [48] D. Stauffer and A. Aharony. *Introduction to percolation theory*. CRC press, 1994.
- [49] G. Yakov. Sinai. probability theory: An introductory course, 1992.