UNIVERSITY OF LIVERPOOL

Distributed Navigation

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Andrew Collins

July 2013

Contents

Illustrations

List of Figures

List of Tables

List of Algorithms

Acronyms

AJAX Asynchronous Javascript and XML.

BBP Bailey-Borwein-Plouffe Algorithm.

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart.

CPU Central Processing Unit.

GNSS Global Navigation Satellite System.

GUI Graphical User Interface.

HTTP Hyper-Text Transport Protocol.

LCA Lowest Common Ancestor.

OCR Optical Character Recognition.

PRNG Pseudo-Random Number Generator.

SAP Self Avoiding Polygon.

SNAP Stanford Network Analysis Package.

TSP Travelling Salesmen Problem.

List of Symbols

α An agent.

$\mathbb{A}$ A set of agents $\mathbb{A} = \{\alpha_0, \alpha_1, \alpha_2, \ldots\}$.

$\mathcal{O}$ Asymptotic worst case for that an algorithm will perform in.

E A set of edges $E = \{e_0, e_1, e_2, \ldots\}$.

$G = (V, E)$ A Graph containing a set of Vertices V and a set of edges E.

$\mathbb{Z}$ The set of integers.

π ratio of circumference of circle to its diameter.

$\mathbb{R}$ The set of reals.

$-\infty$ Negative infinity.

$+\infty$ Positive infinity.

V A set of vertices $V = \{v_0, v_1, v_2, \ldots\}$.

Λ Vital Regions.

λ Vital Point.

Preface

The main part of this thesis consists of three papers which have been peer reviewed and accepted to three significant conferences. The chapters in this thesis are a reformatted version of those papers with minor amendments to make the thesis consistent throughout. In addition to the chapters related to published work an additional chapter is presented which provides discussion on the experimental and exploratory work that was also conducted by the author.

Firstly, Chapter 3 contains the paper, *"Synchronous rendezvous for location-aware agents"* which was co-authored with Jurek Czyowicz, Leszek Gąsieniec, Adrian Kosowski and Russell Martin, and accepted and published in the proceedings of the 25th *International Symposium on Distributed Computing.*

> We study rendezvous of two anonymous agents, where each agent knows its own initial position in the environment. Their task is to meet each other as quickly as possible. The time of the rendezvous is measured by the number of synchronous rounds that agents need to use in the worst case in order to meet. In each round, an agent may make a simple move or it may stay motionless. We consider two types of environments, finite or infinite graphs and Euclidean spaces. A simple move traverses a single edge (in a graph) or at most a unit distance (in Euclidean space). The rendezvous consists in visiting by both agents the same point of the environment simultaneously (in the same round).
>
> In this paper, we propose several asymptotically optimal rendezvous algorithms. In particular, we show that in the line and trees as well as in multi-dimensional Euclidean spaces and grids the agents can rendezvous in time $\mathcal{O}(d)$, where d is the distance between the initial positions of the agents.
>
> The problem of location-aware rendezvous was studied before in the asynchronous model for Euclidean spaces and multi-dimensional grids, where the emphasis was on the length of the adopted rendezvous trajectory. We point out that, contrary to the asynchronous case, where the cost of rendezvous is dominated by the size of potentially large neighbourhoods, the agents are able to meet in all graphs of at most n nodes in time almost linear in d, namely, $\mathcal{O}(d \log 2n)$.

We also determine an infinite family of graphs in which synchronized rendezvous takes time $\Omega(d)$. [53]

Chapter 4 contains the paper, *"Tell Me Where I Am So I Can Meet You Sooner"* which was co-authored with Jurek Czyowicz, Leszek Gąsieniec, and Arnaud Labourel, and was accepted and published in the proceedings of the 37th *International Colloquium on Automata, Languages and Programming.*

In this paper we study efficient rendezvous of two mobile agents moving asynchronously in the Euclidean 2d-space. Each agent has limited visibility, permitting it to see its neighbourhood at unit range from its current location. Moreover, it is assumed that each agent knows its own initial position in the plane given by its coordinates. The agents, however, are not aware of each others position. The agents possess coherent compasses and the same unit of length, which permit them to consider their current positions within the same system of coordinates. The cost of the rendezvous algorithm is the sum of lengths of the trajectories of both agents. This cost is taken as the maximum over all possible asynchronous movements of the agents, controlled by the adversary.

We propose an algorithm that allows the agents to meet in a local neighbourhood of diameter $\mathcal{O}(d)$, where d is the original distance between the agents. This seems rather surprising since each agent is unaware of the possible location of the other agent. In fact, the cost of our algorithm is $\mathcal{O}(d^{2+\epsilon})$, for any constant $\epsilon > 0$. This is almost optimal, since a lower bound of $\Omega(d^2)$ is straightforward. The only up to date paper [61] on asynchronous rendezvous of bounded-visibility agents in the plane provides the feasibility proof for rendezvous, proposing a solution exponential in the distance d and in the labels of the agents. In contrast, we show here that, when the identity of the agent is based solely on its original location, an almost optimal solution is possible.

An integral component of our solution is the construction of a novel type of non-simple space-filling curves that preserve locality. An infinite curve of this type visits specific grid points in the plane and provides a route that can be adopted by the mobile agents in search for one another. This new concept may also appear counter-intuitive in view of the result from [99] stating that for any simple space-filling curve, there always exists a pair of close points in the plane, such that their distance along the space-filling curve is arbitrarily large. [52]

Chapter 5 contains the paper, *"Optimal Patrolling of Fragmented Boundaries"* which was co-authored with Jurek Czyzowicz, Leszek Gąsieniec, Adrian Kosowski, Evangelos Kranakis, Danny Krizanc, Russell Martin and Oscar Morales Ponce in the proceedings of the 25th *ACM Symposium on Parallelism in Algorithms and Architectures.*

A set of mobile agents are deployed on a simple curve of finite length, composed of a finite set of *vital segments* separated by *neutral segments*. The agents have to patrol the vital segments by perpetually moving on the curve, without exceeding their maximum speed. The quality of patrolling is measured by the *idleness*, i.e. the longest time period during which any vital point on the curve is not visited by any agent. Given a configuration of vital segments, our goal is to provide algorithms describing the movement of the agents along the curve so as to minimize the idleness.

Our main contribution is a proof that the optimal solution to the patrolling problem is attained either by the cyclic strategy, in which all the agents move in one direction around the curve, or by the partition strategy, in which the curve is partitioned into sections which are patrolled separately by individual agents. These two fundamental types of strategies were studied in the past in the robotics community in different theoretical and experimental settings. However, to our knowledge, this is the first theoretical analysis proving optimality in such a general scenario. Throughout the paper we assume that all agents have the same maximum speed. In fact, the claim is known to be invalid when this assumption does not hold, cf. [62]. [54]

Finally Chapter 6 discusses unpublished experimental and exploratory work in the area of *parasitic computing*, *basic walk*, *graph visualisation and analysis* that was conducted in parallel to the previously mentioned chapters.

Abstract

In this thesis, a number of problems are looked at predominately in the area of mobile agent communication protocols. Particularly, with the recent uptake of unstructured, large and dynamic networks there is a demand for cheap, ubiquitous and reliable protocols that will assist in the support of the network through tasks such as information dissemination, information search and retrieval, and network monitoring.

One of the exciting new possible solutions to this is in the area of mobile agents where by a team of dedicated agents (physical or purely virtual) act independently or within a team on the network in the goal of solving simple, yet highly valuable, tasks. Independent agents may work within the bounds of their environment and without influence from their colleagues to solve simple or potentially complex tasks. Further, the agents may also act independently but in actuality work as a very complex team with simple underlying principles allowing for large scale networks and problems to be handled autonomously.

In this work, two problems are investigated in detail, the *Rendezvous Problem* and *Network Patrolling.* In the rendezvous problem, the goal is to identify algorithms that will permit two agents to *rendezvous* within some known or unknown environment. This problem, while at first can feel trivial, it can be incredibly difficult when it is realised that all agents are expected to be identical at wake-up time, and that methods must be found that allow for the breaking of symmetry. This thesis provides an investigation into a number of models and environments and tries to provide optimal algorithms for allow rendezvous to occur. Secondly in the area of network patrolling, this work investigates the problem where there exists an environment, in which parts of it are viewed as being vital to network integrity and as such must be monitored. When there are less agents than vital regions the challenge of identifying traversal routes that minimise *idle time* becomes apparent. In this work an algorithm is presented that minimises this in the ring environment.

Finally, this work also looks at other problems in distributed computing and provides exploratory foundation work that could provide alternative models for *routing problems*, *distributed processing*, *community identification*, and presents a number of open problems.

Acknowledgements

This project has been both an interesting and exciting time, while also challenging and testing. Undoubtedly this project would not have as successful without the people around me who have provided support and guidance. So in this small section of acknowledgements I would like to use this opportunity to thank those most notable.

Firstly and foremost, this project would not have been possible without those who have been responsible for the supervision of this project, Leszek A. Gąsieniec, Darek Kowalski, and Jurek Czyzowicz.

Secondly, many parts of this work were produced with the help of valuable contributions from my co-authors, Adrian Kosowski, Evangelos Kranakis, Danny Krizanc, Arnaud Labourel, Russell Martin, and Oscar Ponce. Additionally, and while not officially a co-author in a published work, I would like to thank the invaluable contributions made to this project by Evangelos Bampas and Ralf Klasing.

Further, I would also like to thank my examiners Prudence Wong and Costas Iliopoulos for both their thoroughness of this work and advice for future work.

Finally, I would like to thank family and friends, who have tolerated the many hours I have dedicated purely to this project over the previous years.

Chapter 1

Introduction

In this thesis the field of *Computer Science* is investigated, and specifically problems in the area of the *Theoretical Computational Complexity* of certain *Distributed Algorithms*. Also within this work are other more applied aspects of distributed algorithms and graphs in general, which were of the authors interests, that were investigated during the main themes of this project. Firstly a general introduction to the two main themes discussed in this thesis are provided, though a more detailed introduction to the problems can be found in Chapter 2.

1.1 Rendezvous Problem

The *Rendezvous Problem* is defined as the problem of getting two or more entities to *rendezvous* in an environment, in the most simplest of terms this can be thought of as two individuals who agree to meet at a location only to later discover that the location they agreed to meet within is larger than visibility permits thus requiring the two individuals to explore their environment to find each other. Further, as a rule of this problem, it is expected that those attempting to meet do not know initially where the other person is, or are able to communicate. One of the most common examples of this problem is that two tourists agree to meet at *Central Park, New York* only to arrive and later realise that the park is far larger than they imagined and thus requiring the tourists to begin *searching* for the other person.

With the recent and continued popularity of large unstructured and often dynamic network environments it has become important for a need for robust and universal yet inexpensive distributed network algorithms. The objective of the network algorithms are to support the integrity of the basic functionality of the network purpose as well as to assist in specific tasks such as information dissemination, network exploration and searching, as well as monitoring and patrolling including that of supporting emergencies.

One of the unusual yet encouraging alternatives that could support networks such as these is through the use of dedicated teams of mobile agents that can operate independently of the network processes. Though while the use of agents may be a challenge to develop, there are a number of possibilities, such as, representing software agents [124] occupying network nodes or traversing between them, autonomous mobile robots [160] located in a geometric (real world) environment, or a group of people that have to meet in a city whose streets form a road network [8]. The independence of the agents allows for them to be adaptable to cases when the environment is either stable or unstable due to accidental or intentional failures within the network.

Further to this, agents can also be considered as representing other types of complex systems on their own. For instance a traditional communication network can be replaced by a more arbitrary environment in which a collection of networked or free-standing agents represent groups of humans, animals, vehicles or specialised robots which are asked to perform a dedicated computational task. This could be done in the form of a fully-coordinated effort or as a collection of (semi-)independent individual (possibly greedy) performances.

The rendezvous of agents is often a challenge on its own. Alternatively, it can be used as a subroutine in a range of basic network integrity and co-ordination mechanisms. The agent's ability to act autonomously including observation, communication and relocation impels the design and further implementation of efficient communication and navigation mechanisms. In this thesis this challenge of getting the agents to perform rendezvous are discussed by showing methods discovered in the published works [52, 53] for solving specific scenarios within the wider rendezvous problem.

1.2 Network Patrolling

Similarly to the previous topic, another problem that is becoming of interest due to both the continued popularity of large networks and also due to the rise of more complex and autonomous robotics is that of *Network Patrolling*. Protecting an environment through the use of a set of stationary or mobile point-guards has been studied before in various scenarios. The problem of patrolling a one-dimensional boundary using agents has many real-world applications, and is extensively studied under the names of *boundary patrolling* and *fence patrolling* in the robotics literature [62]. In order to prevent an intruder from penetrating into a protected region, the boundary of the region must be patrolled. Some parts of the boundary may be monitored with stationary devices like sensors or cameras (or they do not need to be monitored at all), while other portions require the aid of moving robots such as walking guards, illumination rays, mobile robotic devices, etc. Since the feasibility of an intrusion likely depends on the time during which the intruder remains undiscovered, it is important to design patrolling algorithms which minimise the time during which boundary points are unprotected.

In this the concept of agents is extended from rendezvous to that of network patrolling by the requirement that the agents are required to perform a given task autonomously and at some minimal cost. Where as in the previous case the mobile agents were required to locate each other, in this case the agents can be considered as trying to avoid each other and instead trying to locate areas to monitor that are not currently monitored.

While this at first may seem trivial, the task becomes difficult once the regions that are required to be monitored is greater than the number of mobile agents, thus requiring an agent to move between monitored regions.

1.3 Additional Topics

In addition to the main themes of this topic as discussed in §1.1 and §1.2 this work will also look at some of the more applied interests conducted during this project, which are also in relation to distributed algorithms.

There are three topics which were investigated, the first is in *Parasitic Computing*. Parasitic computing is the concept of exploiting existing algorithms and protocols that are being executed by a host to provide a resource of computing power for the parasite, without the hosts knowledge of the exploits occurring. This allows for a distributed algorithm to essentially latch on to an unknown host to bleed a small amount of Central Processing Unit (CPU) time that across a large network (e.g. the Internet) can result in the parasite having access to an immense amount of computing power.

The second topic that will be looked at is the *Basic Walk*, the basic walk is a deterministic counter-part to a *Random Walk*. In this topic the algorithm is performed on very large graphs, with analysis conducted on the output. In the results of this work it is discovered that there are various unexpected trends leading to the possibilities of interesting future directions.

Finally, the third topic to be discussed was a by-product of the previous topic. As larger and larger graphs were investigated it came to prominence that a tool would be required to allow for the efficient storage and visualisation of these graphs, and as the tools became more efficient the more interesting it came of interest to other parties in the fields of social media and biology. In this topic the tools that were built and the algorithms that were utilised are reviewed, as well as future directions considered.

1.4 Outline

This thesis presents work related to the main topic of this doctorate and as well as discussion on a number of a side projects undertaken through the duration. The topics are organised as follows:

Chapter Two continues the outlines provided in this chapter and goes into further detail. It provides a more detailed introduction and background to the topics that are mentioned above and looks at the background to algorithms and distributed computing, and how the rendezvous problem can be applied, as well as the interest of network patrolling in these contexts.

Chapter Three discusses in more detail the concept of *synchronous rendezvous* and shows the results that were obtained during the execution of this project. Specifically the results that were presented in [53] are discussed.

Chapter Four also looks into a topic of rendezvous, however, in this chapter the *asynchronous rendezvous* model is investigated. In this chapter the results produced for [52] are shown.

Chapter Five looks away from rendezvous and focuses instead on a slightly different topic, that of *network patrolling.* Once again this chapter is also dedicated to results produced in this project, specifically that of [54].

Chapter Six contains a look at some of the *other work* that was conducted by the author through the duration of this project that have either taken academic or industrial interest but do not fit into the preceding chapters. The topics of *parasitic computation*, *the basic walk*, and finally *graph visualisation* are mentioned in this chapter.

Chapter Seven shows the final conclusions of these works and looks at potential directions for future work.

1.5 Authors contribution

The work completed in Chapter 3, 4, and 5 were completed by the author with significant contributions by the co-authors. Specifically the author has a general interest in linear structures and all of these works make or developed on linear structures. In Chapter 3, the author contributed in the development of the solutions for the infinite line, the half-line and the tree environments, as well as contributed in discussions towards the remaining problems discussed. In Chapter 4 the author contributed towards the development of the tree structure of central and quad partitions which were an attempt to reduce a 2D space to that of a linear structure (space-covering sequence). In Chapter 5 the author contributed towards the development of the solution produced for the linear stuctures and also contributed towards the discussions in the ring environment which appear in the paper. Finally the work completed in Chapter 6 was completed solely by the author, however, specifically in terms of §6.3, additional development work has been contributed to this project from both under-graduate and post-graduate students with guidance from the author.

Chapter 2

Distributed Navigation

In this chapter the background work that has lead to the production of this thesis is introduced. This chapter is intended to be a lighter introduction to the topic for readers which are unfamiliar with the topics that will be discussed, however, readers acquainted with the topics may find that they can skip this chapter and proceed to Chapter 3.

First the field of computer science and specifically the concept of algorithms will be introduced, and further the difference of a traditional algorithm to that of a distributed algorithm. The significance of these two topics in computer science and examples of where they have been utilised will be highlighted.

Secondly the problems that have been worked on within this thesis will be discussed, from their origins and aims to their current form. Specifically, the rendezvous problem and network patrolling as well as the topographical models that these techniques are applied to.

Finally, the other content of this thesis that will be covered in the final chapters, in more detail, is introduced.

2.1 Algorithms

The word "algorithm" is a descendant of *al-Khwārizmī* from *Abū 'Abdallāh Muḥammad ibn Mūsā al-Khwārizmī*[1] (Father of Abdullah, Mohammad, son of Moses, native of Khwārizm), a Persian astronomer, geographer and mathematician. One of al-Khwārizmī's most famous works is the *"Al-kitāb al-mukhtaṣar fī ḥisāb al-ğabr wa'l-muqābala"* (*"The Compendious Book on Calculation by Completion and Balancing"*), which also where the word "algebra" (al-ğabr) originates from [113].

Since this work, a number of interpretations of the works title, and of the authors name have occurred which has confused the origins of the term algorithm leading to instances where Latin has been mistakenly suggested as the source. Further to this, many variations of the spelling have existed, though today the term and spelling "algorithm" is the preferred, which is defined by [163] as being:

[1]Variations of *Al-Khwārizmī's* are quite frequent across the literature. The only general consistency is in the surname, al-Khwārizmī

كتاب الخوارزمي

[illegible]

FIGURE 2.1: A page from *Al-Khwārizmī's "Al-kitāb al-mukhtaṣar fī ḥisāb al-ğabr wa'l-muqābala"*

Definition 2.1 (Algorithm)**.** A procedure or set of rules used in calculation and problem-solving; (in later use spec.) a precisely defined set of mathematical or logical operations for the performance of a particular task.

Though while this definition owes at the very least its origins to 1811 according to [163] it is in 1937 that the modern concepts of an algorithm, and the computational model were formalised by Alan Turing [169].

As per the definition, an algorithm is considered as a strictly defined series of instructions that will accept a value or a set of values as an input parameter and then return a value or set of values as the output. Algorithms can be considered as a tool for solving various computational problems where their exists a problem scenario, some form of input, and some form of output. From this basic definition an algorithm can be viewed as the process that a computer undertakes to generate the answer for a given question.

Perhaps one of the most well known and certainly most important types of algorithms in computing is that of the *sorting algorithm.* A sorting algorithm is an algorithm which takes a series of input values (e.g. $1, 5, 3, 8, 0, 13, 1$) and returns the values sorted by a required criteria (e.g. in non-decreasing order: $0, 1, 1, 3, 5, 8, 13$). While this concept seems trivial and initially unimportant it is easy to see its significance when a computer must work with large amounts of data. For instance, consider a dictionary, due to the expectation that a dictionary is sorted one can generally identify the word that is being looked for by opening the book at the mid point, and then through gradually splitting the book further (i.e. at the quaternary points) and further the word should eventually be reached. In a dictionary of approximately $1,000$ pages one would hope to find the word that is being looked for after approximately 10 page turns (or $\log_2 n$ where n is defined as the number of pages). Now if the dictionary was not sorted then this method would be unsuccessful, and only a complete search from the first page to the last page would identify the word that one would like to find. So specifically, if their are $1,000$ pages it would require up to $1,000$ page turns (or n). From this simple example the reader can get a feel for the significance of sorting, because as the input grows, finding a specific record will become much more difficult in unsorted data than it would in sorted data. The concept of measuring expected run time is discussed in more detail in later sections.

It is for this very reason as to why sorting is so important in computing, as to work with very large datasets would be impossible if a computer could not accurately predict where a specific record within the dataset is to be found. Sorting of course may not refer to words or numbers, it could apply to any criteria (by date, location, etc.), and be ordered in any direction ascending, descending, or even randomly if required [56, 112, 129].

However, regardless of the efficiency (though with some exceptions) of an algorithm, the size of the input can eventually reach a point where a single processor may find it infeasible to run on a given input and it is in this scenario when the concept of distributed algorithms must be considered.

2.1.1 Distributed Algorithms

Traditionally an algorithm is designed such that a single processor may execute the algorithm for the given inputs in its entirety and be solely responsible for the production of the output. For many day-to-day requirements of computers this model is completely satisfactory, however, there exists cases where this model cannot currently, and perhaps may never, succeed. A good algorithm will generally be abstract enough such that it is not hardware or software dependent, and further to this a distributed algorithm will allow for a single algorithm to be run across various different hardware configurations in parallel, regardless of the hardware, speed or reliability of each processor.

At the time of writing, the world is currently in a position where it is becoming more and more connected, and more so since the popularity of the Internet and mobile communications. Further to this, computing has become incredibly inexpensive allowing for various, and possibly unexpected, devices to employ the use of small processors for the purpose of capturing or adapting to sensor results within the device. For instance the *CERN Large Hadron Collider* produces around 15 petabytes of data per year [125], which itself is large, however, even this is small compared to the envisioned exabytes of data that will be produced *per day* by the *Square Kilometre Array* [32], a figure that is believed to be larger than the current total amount of information available on the Internet. Figures that are simply impossible to process for a single processor. In combination with this issue, it is forecast that the theoretical limitations of *Moore's Law* are gradually being reached [111, 132]. It is for that reason algorithms which can be distributed across multiple processors must be considered, to enable a drastically higher theoretical limit to the amount of processing power that is available for a single task.

With, as mentioned, the advent of large scale networks such as the Internet and the coming availability of ubiquitous computing it now seems more sensible that such algorithms should be considered to counter the problem of large data and exploit the vast pool computing power available. Though this thesis by no means sees this as a new or unknown problem as there are a number of existing examples where distributed computing is being used to solve large data inputs. Already distributed computing can be found in applications related to institutes that utilise cloud computing [100], large scale scientific projects [135, 176, 186], mobile computing [31, 158], and various others.

2.2 Analysis of Algorithms

Computational Complexity Theory (or simply, *Complexity Theory*) is a well studied field within theoretical computer science and specifically, the *theory of computation*. The concept of complexity theory is to provide a means of classifying the difficulty of any given problem so as to allow for the classified problems to be related to each other. Within the wider field of complexity theory exists the concept of *Algorithm Analysis* (Or, the *Analysis of Algorithms*) which focuses on a similar but related challenge. Specifically the concept of algorithm analysis is to identify the difficulty of any given algorithm which then permits for any algorithms that solve the same tasks to be compared. The significance of this area is to identify the scalability of an algorithm. While for small inputs generally it could be said that "any algorithm is good enough" as computation speeds today can be sufficient for even the most difficult algorithms. However, once an input grows it needs to be understood how this will effect the run time or memory requirements of an algorithm, as it is common for these constraints not to scale linearly with the size of the input. However, while computation power is increasing, if an algorithm cannot scale then there may be only a tiny gain or even no gain if the size of the input increases too.

Consider Table 2.1 reproduced from [112] which shows that if an algorithm requires 2^n steps to complete, then it would require less than 1 second with an input of size 10, however, if the input size increases by a factor of 10 to 100 then it would require 10^{17} years to complete, even if computational power increases by the same factor then it would still require 10^{16} years to complete. However, if the algorithm was to require n^2 steps to complete then for both inputs it would be expected that the algorithm would terminate after 1 second. From this very simple example it can be seen that while initially, "any algorithm is good enough" when the input is very small, once the input starts to grow it becomes more and more important that efficient algorithms are required to be discovered. Further, the importance of why algorithms need to be analysed can also be seen.

Input Size	n	$n \log n$	n^2	n^3	1.5^n	2^n	$n!$
10	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
30	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 m	10^{25} Y
50	< 1 s	< 1 s	< 1 s	< 1 s	11 m	36 Y	v. long
100	< 1 s	< 1 s	< 1 s	1 s	12,892 Y	10^{17} Y	v. long
1,000	< 1 s	< 1 s	1 s	18 m	v. long	v. long	v. long
10,000	< 1 s	< 1 s	2 m	12 D	v. long	v. long	v. long
100,000	< 1 s	2 s	3 h	32 Y	v. long	v. long	v. long
1,000,000	1 s	20 s	12 D	31,710 Y	v. long	v. long	v. long

Table 2.1: The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time [112]. Key: s = second, m = minute, h = hour, D = day, Y = year

The two topics of most interest within complexity theory are those of *time complexity* and *space complexity*. Time complexity refers to what was previously discussed, the amount of steps that that an algorithm must perform to complete the given task for the specified input. Similarly space complexity refers to the amount of memory space required to complete a given task, in terms of algorithm efficiency it is both of these aspects that should be minimised. Realistically a good algorithm should be abstracted away from any specific hardware or language, which leads to the use of *Big O Notation*, as this notation allows for the generalising of algorithm behaviours without regard for the implementation.

Big O notation is used as a mathematical method for describing the asymptotic behaviour of functions. Its intent is to allow for the defining of a function's behaviour for very large inputs so that any pair of functions can be compared to identify which is more efficient than the other. In this work the symbol $\mathcal{O}$ is used to denote the asymptotic upper bound for the magnitude of a function in terms of another[2].

Big O notation can be used for analysing both the space and time complexity of algorithms.

2.3 Graph Theory

Throughout this work elements of *graph theory* and the *graph model* will be used. In the fields of mathematics and computer science the graph model is used as a means of constructing an abstract model of relationships between objects. Graph theory owes its history to Leonhard Euler's, *"Solutio problematis ad geometriam situs pertinentis"* (*"The solution of a problem relating to the geometry of position"*) which was published in the journal, *"Commentarii academiae scientiarum Petropolitanae"* in 1741.

In this work Euler discusses the problem, now known as, the *"Seven Bridges of Königsberg"*. The goal of this problem is to identify a path through the city of Königsberg

[2]Though in other works O may simply be used.

(formerly in Prussia, now known as Kaliningrad, Russia) that would involve crossing all seven bridges exactly once (shown in Figure 2.2). Further, the rules of the problem state that a bridge must always be crossed in its entirety and that to get to the parts of the city only connected by bridges, a bridge must be used.

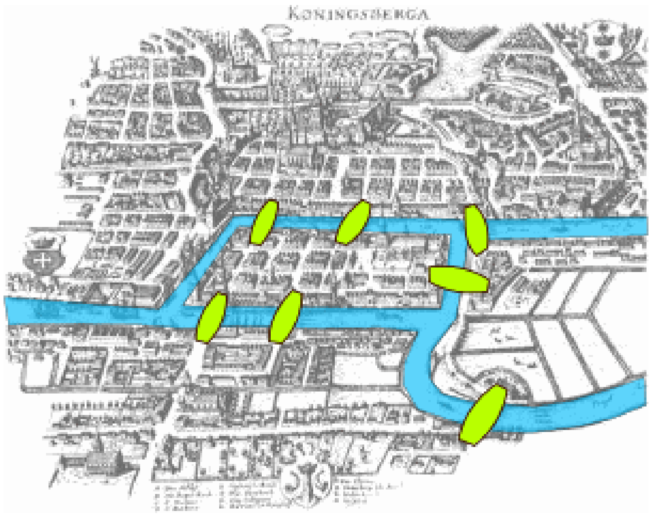

FIGURE 2.2: The problem of the Seven Bridges of Königsberg [178]

Euler identified and proved that no such solution to this problem exists, through the use of an abstract terminology that has since lead to the development of *graph theory*. In modern terms Euler showed that it was possible to reduce the problem such that there exists two elements, a set of *Vertices*[3], and a set of *Edges*[4], by mapping each land mass to a vertex and mapping each bridge to an edge connected to the two vertices that represent the two land masses the bridge connected. Thus producing the graph shown in Figure 2.3.

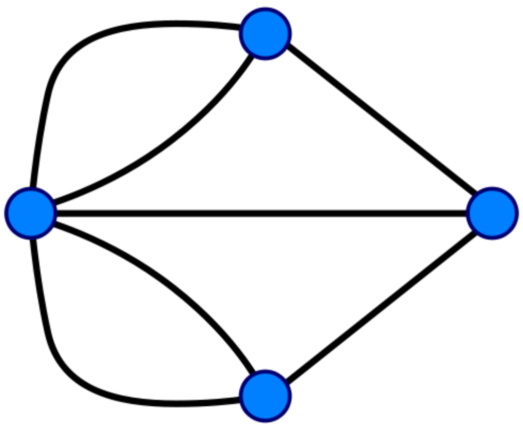

FIGURE 2.3: Abstract graph corresponding to bridges of Königsburg [178]

[3]Sometimes referred to as a nodes or points, though in this work the terms vertex and vertices are preferred.

[4]Similarly, sometimes referred to as lines.

In modern graph theory the concept of mapping objects and their relations to that of a graph is re-used and denoted using $G = (V, E)$, where a graph is a set of vertices V and a set of edges E. Through modelling, using this concept, many problems can be reduced to something that can easily be studied using established techniques, and consequently, through studying problems using this model many solutions can be applied to various complex problems. Common applications modelled through this technique are social networks [134], computer networks [127], traffic networks [128], and various others.

2.4 Studied Problems

While these areas are becoming more established as demand increases, this work looks at a lesser investigated area, specifically the concept of *distributed navigation*. In this thesis identification of the time complexity of a number of theoretical models that are designed to map directly to abstract distributed processes will be shown.

The models will be reduced to methods for solving the *Rendezvous Problem*, and for performing *Network Patrolling*.

2.4.1 Rendezvous Problem

In the rendezvous problem two, or more, mobile agents are required to meet, or more aptly, *to rendezvous*. Though in this work only two agent rendezvous is investigated. All of the agents are placed at different positions within the environment, however, initially each agent does not know the location of any other agent. Thus this leads to the interesting challenge of discovering algorithms that allow for all agents to rendezvous. To do this, an agent is required to explore its current environment, or at the very least part of it, until it can locate all of the other agents. As per the prior section big O notation will be used to denote the expected time complexity of the solutions, however, in this scenario the initial distance of the agents from each other is used as the measure of cost and the agents must rendezvous relative to this cost.

One of the early references to the rendezvous problem is a 1960 work from the field of political science. In this early work, Schelling [156] discussed the co-ordination problem whereby there exists two players that must attempt to successfully guess the location of a common meeting point. The players are *only* provided with one opportunity to meet and if the attempt fails then the players *fail*.

Since this work, the rendezvous problem has been redefined and was informally re-introduced by Alpern [5], in 1976. In this early work Alpern introduced a number of problems such as the *Astronaut Problem* and the *Telephone Problem*, of which so far despite the large body of research on rendezvous problems, do not yet have solutions:

Definition 2.2 (Astronaut Problem)**.** Two astronauts land on a spherical body that is much larger than the detection radius (within which they can see each other). The body does not have a fixed orientation in space, nor does it have an axis of rotation, so that no common notion of position or direction is available to the astronauts for coordination.

Given unit walking speeds for both astronauts, how should they move about so as to minimise the expected meeting time T (before they come within the detection radius)? [10]

Definition 2.3 (Telephone Problem)**.** In each of two rooms, there are n telephones randomly strewn about. They are connected in a pairwise fashion by n wires. At discrete times $t = 0, 1, 2, \ldots$ players in each room pick up a phone and say "hello". They wish to minimise the time T when they first pick up paired phones and can communicate. What common randomisation procedure should they adopt for choosing the order in which they pick up the phones? [10]

In later years the topic became more popular due to the influential work by Anderson and Weber [17] on discrete location rendezvous. Since then the topic has continued to get more popular and formulation of the problems has continued through the work of Alpern [6]. The topic has reached such a level in popularity that there are now a number of documents providing overviews, such as the two extensive surveys by Alpern [6, 7], and also a book by Alpern and Gal [10] which discusses various methods of performing rendezvous using randomised techniques. Further other authors have also written exhaustively on sub-problems within rendezvous such as the book by Kranakis et al. [122] which focuses on rendezvous in the ring, and the book by Alpern et al. [12] that provides an overview to various aspects of the topic suitable for various fields such as computer science, mathematics and biology. In addition to this the topic can also be looked at from different aspects such as was investigated by Olfati-Saber et al. [144] in his work related to the consensus problem.

In general, the results produced so far in the field can be classified as those considering a form of *geometric setting*, such as, rendezvous in the line [14, 28, 29, 95], in the ring [66, 72, 120, 122] whereby agents must meet in a linear structure, or rendezvous in the plane [15, 16, 61, 130, 165, 167]) whereby the agents must meet in a more (real-world) 2D or 3D environment, and those considering rendezvous in graphs [6, 11].

Secondly, the environment an agent is deployed within can be classified as either *finite*, or of *unbounded* in size. While in the finite case, exploration of the environment can be feasible, however in the unbounded case, having the agents perform rendezvous requires the discovery of search algorithms that preserve the locality of the solution. Without this requirement the complexity of the solution could be unbounded. In addition to this, another important network attribute refers to the localisation property, whether the agents have knowledge of their current location, and also to the sense of direction. For example, in the geometric setting these two properties refer to the system of co-ordinates accompanied by geographic directions, i.e. through having access to a *positioning system* such as a Global Navigation Satellite System (GNSS). Flocchini et al. [87] have previously shown that by providing a sense of direction it can greatly improve the chance of solving and the efficiency to a number of problems in distributed computing, with Barriere et al. [26] further showing it to be important in rendezvous as well. However, the network itself can be reliable, or it can report to its users inaccurate and potentially unusable information and in such error prone network rendezvous time can be largely elongated or meeting may prove to be impossible [71].

Finally, another critical property refers to the availability of a *global clock* to the agents within the network. In particular, in a synchronous network it is assumed access to the global clock allows agents to co-ordinate their actions, including their movement, using time frames such that at each time frame an action is performed. In contrast, in asynchronous networks it is assumed their is no accessible global clock, and thus the speed with which an agent performs an action is indeterminable and thus cannot be co-ordinated. In this case rendezvous occurs either through utilisation of predefined trajectories [168], or through the analysis of the current configuration of the network [150].

The rendezvous problem can be examined in different ways and different communities are known to look at the problem differing directions. For instance the computer science community are known for showing interest in identifying solutions to the problems with a focus on efficient algorithms in terms of the resource usage (i.e. time, or memory). Meanwhile the operations research community has shown more interest in identifying methods for maximising the chance of the two agents meeting and also reducing the time it takes to meet. However, regardless of the different directions the communities are taking, both share a common interest in performing rendezvous in the line, of which so far there are various deterministic and non-deterministic results in this environment.

In regards to the line, it was Alpern who introduced and provided a solution to the problem of symmetric rendezvous. Alpern [6] provided a solution of $5d$ where d is defined as the original distance between the agents within the environment. In this result Alpern proposed that the agents should initially select a direction at random and traverse towards the position at d from their current position. Once at this destination the agents should then return in the opposing direction for $2d$ steps at a unit speed. Further, Alpern and Gal [9] provided a proof stating that for all symmetric rendezvous strategies, the agents should meet after at least $3.25 \cdot d$ steps. While Alpern and Gal [9] discussed lower bounds, other authors such as Anderson and Essegaier [14] proposed that when mixed movements are used there will be an upper bound of $4.5678 \cdot d$. However, since these results in 1995 work has continued to refine them with Baston [27] making minor improvements in 1999 to provide an upper bound of $4.4182 \cdot d$. Later still, Uthaisombut [170] showcased a fresh look at a mixed strategy that gave another slight improve to the upper bound with $4.3931 \cdot d$. His results also gave an argument for a lower bound being revised to $3.9546 \cdot d$. However, in an even more recent result, Han et al. [103] considered both of these bounds and provide arguments for an upper bound of $4.2574 \cdot d$ and lower bound of $4.1520 \cdot d$, with also a conjecture that the rendezvous is asymptotically equal to $4.25 \cdot d$.

While most of the deterministic rendezvous algorithms in the line use asynchronous models which uses the total walk distance of any agent as the cost of the rendezvous process, there is however work in other environments. Dessmark et al. [69] for instance discuss efficient rendezvous strategies in tree environments and propose for rendezvous occurring with a cost of $\mathcal{O}(n)$ on a route of length n. There is also work by De Marco et al. [67] which considers the infinite line with the difference that the agents are not only labelled but also know the value of d. In this work, rendezvous is shown to occur in $\mathcal{O}(d|L_{min}|^2)$, where $|L_{min}|$ is defined as the length of the smallest label and $\mathcal{O}(d^3 + |L_{max}|^3)$, where $|L_{max}|$ is defined as the length of the largest label, if d is not known. Recently, Stachowiak [166] improved the result for when d is not know known to $\mathcal{O}(d \cdot \log^2 d + d \cdot d \log d|L_{max}| + d|L_{min}|2 + |L_{max}||L_{min}| \log |L_{min}|)$.

Deterministic Rendezvous

One of the fundamental issues in the deterministic rendezvous algorithms is the problem of *symmetry breaking*, since identical agents starting at some symmetric positions may never meet, indefinitely locked to performing symmetric moves at a distance d between each other. One possible way to break symmetry is to have agents identified with different *labels*. For the case of agents which are unlabelled, i.e. *anonymous*, [90] studied rendezvous in trees, assuming that agents have bounded memory. However, trees are a special case in which rendezvous is often feasible, supposing that neither vertices nor agents are labelled or marked. Rendezvous in a graph is feasible, see [63], if and only if the starting positions of the anonymous agents are asymmetric, i.e. the views of the graph from the initial positions of the agents are distinguishable, cf. [181]. In [184] the problem of gathering many agents with unique labels was studied. In [69, 118] deterministic rendezvous in graphs with labelled agents was considered. One usual approach used for labelled agents consists in finding a (usually non-simple) cycle in the graph, computable by an agent placed at any starting vertex, and an integer bijection f on the set of labels. The agent α goes $f(\alpha)$ times around such cycle and different agents are forced to meet (see e.g. [60, 63, 67, 118]). In [63] it was proved that the log-space memory is sufficient in order to decide whether a given instance of the rendezvous problem is feasible for any graph and any initial positions of the agents.

However, in most of the papers above, the *synchronous* setting was assumed. In the *asynchronous* setting it is assumed that the timing of each action may be arbitrarily slowed down by an adversary. The efficiency of the asynchronous algorithms is determined by the worst-case possible behaviour of the adversary. Asynchronous gathering in geometric environments has been studied, e.g. in [48, 86] in different models than this: anonymous agents are oblivious (they can not remember past events), but they are assumed to have at least partial visibility of the scene. The first paper to consider deterministic asynchronous rendezvous in graphs was [67], where the complexity of rendezvous in simple classes of graphs, such as rings and infinite lines, was studied for labelled agents. In [61] the feasibility of asynchronous rendezvous for labelled agents was considered both for graphs and the 2D-space. It was proved that rendezvous is feasible in any connected (even countably infinite) graph. For this purpose, an enumeration of all quadruples containing possible pairs of agent labels and their starting positions is considered. According to this enumeration, the routes of the agents involved in each quadruple is extended in such a way that their meeting is always ensured. If the graph is unknown, the enumeration is constructed while the graph is explored. The cost of the rendezvous algorithm is exponential in the original distance between the agents. On the other hand, asynchronous rendezvous is infeasible for agents starting at arbitrary positions in the plane, unless the agents have an $\epsilon > 0$ visibility range, see [61]. The assumption that the agents operating in the geometric environment has a bounded, non-zero visibility range is very natural (cf. [16, 61, 96, 130]). The process of learning, and the adaptivity of the agents depends on their memory as well as on observation and communication abilities. For example, in some models it is assumed that the agents are memoryless, where the agents rely on the use of a random walk procedure [55]. The random walk is an example of a randomised procedure requiring access to random bits.

Earlier it was discussed that there are two types of environments that a network may be classified as: geometric, or graph based. The graph based environment models the nodes of the network as vertices. However, the vertices themselves may have the property that they are labelled and thus each point within the network is identifiable, or the vertices may have no identifying qualities and thus in this case the network is *anonymous*. An anonymous vertex can only be identified by it's degree, which in some cases can be quite revealing, however, should any pair of vertices have the same degree then these are impossible to differentiate. Further, exploration of the neighbours does not resolve this in certain graphs, for instance a *complete bipartite graph* has only two vertex types, regardless of the number of vertices. Breaking symmetry can depend on whether the agents can be given identities (such as through adopting the identifying of the vertex they initiate at) [69, 117], or the agents can mark the vertices they traverse[5] (e.g. see [29, 65, 121, 122]). Finally, rendezvous can also be achieved through exploring the asymmetric positions of the agents in the graph (e.g. [63]).

In Chapter 3, which is based on [53], yet another method of breaking symmetry is used, by always providing to each agent information about its current position in the environment (this implies that the environment may not be unknown to the agent). This assumption that the agents know their initial location in the geometric environment was considered in the past in the context of geometric routing, e.g. [1, 35, 119, 123], where it was assumed that the agent knows its own position as well as the position of the destination, or broadcasting, (cf. [82, 83]), where the position awareness of the broadcasting vertex only was admitted. Such assumption, partly encouraged by the availability and the popularity of GNSS, is sometimes called *location awareness* of agents or vertices in the network, and it often leads to better bounds of the proposed solutions. More recently Dieudonné and Pelc [70] showed that rendezvous is also possible when only the initial positions of the agents are known, however, this result came at a higher cost.

[5]In the literature, this is some times referred to as leaving a *pebble* at the vertex, or by having an agent mark a *whiteboard* on the vertex

Further, in Chapter 4, which is based on [52], the rendezvous problem for location-aware agents in the asynchronous case is discussed. Previously, the asynchronous rendezvous was studied by De Marco et al. [67] for lines and rings, while for arbitrary graphs [67] gave an exponential-time rendezvous procedure, under the condition that the bound on the size of the graph is known to the agent. This condition was suppressed in [64], where feasibility of asynchronous rendezvous was settled for arbitrary (even infinite) graphs and geometric environments. Both approaches in [67], and in [64] lead to very inefficient, exponential-time rendezvous algorithms for labelled agents. However, in [52] the concept of covering sequences that permitted location aware agents to meet along the route of polynomial length in d in multi-dimensional grids was introduced. This result was further advanced in [25], where the proposed algorithm provides a route, leading to rendezvous, of length being only a poly-logarithmic factor away from the optimal rendezvous trajectory. The inherent bottleneck, however, in asynchronous location aware rendezvous is in scenarios where there are potentially a large number of local neighbourhoods. In the worst case, every agent must search through its entire neighbourhood of radius d when, e.g. the other agent is immobilised by the adversary that controls the actions of both agents.

The main emphasis in this thesis is on *local rendezvous*, i.e. the agents are expected to meet without visiting remote parts of the network. In this setting the rendezvous cost tends to be proportional to d. Some local rendezvous strategies were studied in the geometric setting where the agents have either a complete or limited visibility [165]. E.g. Ando et al. [18] studied convergence stability of multiple agents represented as points on the plane. In a similar geometric setting Cohen and Peleg [50] considered convergence properties of gravitational algorithms including scenarios populated by crash faults.

The synchronous deterministic rendezvous for labelled agents in graphs was first studied in [69], where the main result was a polynomial-time rendezvous algorithm. The authors of [118] and [168] independently extended the approach of [69] to the case of agents starting their movement with an arbitrary delay. However the algorithms from [69, 118, 168] are highly polynomial in the size of the network. Thanks to the location awareness assumption, the approach used in Chapter 3 results in a variety of very efficient algorithms, linear or slightly super-linear in the initial distance d between the agents, working, in some cases, also for infinite graphs and multi-dimensional grids and spaces.

2.4.2 Network Patrolling

The second problem that is investigated in this work is that of network patrolling. In network patrolling the previously discussed model of mobile agents is used to monitor and protect an environment. Protecting an environment through the use of a set of stationary or mobile point guards has been studied before in various scenarios. The problem of patrolling a one-dimensional boundary using mobile agents has many real-world applications, and is extensively studied under the names of *boundary patrolling* and *fence patrolling* in the robotics field [62]. In order to prevent an intruder from penetrating into a protected region, the boundary of the region must be patrolled. Some parts of the boundary may be monitored using a variety of stationary devices and sensors, such as motion sensors, cameras, or microphones (or they do not need to be monitored at all), while other parts of the environment may require the aid of mobile robots such as walking guards, illumination rays, mobile robotic devices, or others. In this work it is considered that the feasibility of an intrusion being likely wholly depends on the time during which the intruder remains undetected, so for that reason the focus is aimed towards the discovery of patrolling algorithms which minimise the time during which boundary points are unprotected.

The act of *patrolling* is defined as the perpetual process of surveillance consisting of walking around a terrain in order to protect or supervise it; it is performed either in a static or in a dynamically changing environment. It has been studied extensively in the robotics literature (cf. [4, 43, 77, 79, 80, 104, 131, 183]) and it is sometimes viewed as a variant of *coverage* - a central task in robotics. Patrolling can be useful in settings where objects or humans need to be rescued from a disaster environment, but also network administrators may use agent patrols to detect network failures or to discover web pages which need to be indexed by search engines, cf. [131].

Similarly, *boundary patrolling* may be motivated by the task of detecting intruders (from the exterior) in a two dimensional terrain by patrolling its boundary. There exist several studies on boundary patrolling (cf. [3, 79, 80, 148]); often the approach followed is ad hoc, emphasising either experimental results (e.g. [131]), or uncertainty of the model and robustness of the solutions when failures are possible (e.g. [79, 80, 104]), or non-deterministic solutions (e.g. [3]). Several fundamental concepts, including models of agents (e.g. visibility or depth of perception), locomotion, (relative movement of the agents in the environment or motion co-ordination), means of communication, as well as measures of algorithm efficiency can be found in the experimental paper [131].

The fundamental measure for evaluating the efficiency of patrolling is the criterion of *idleness*, first introduced in [131]. The general idea is to measure algorithmic efficiency by frequency of visits of the points of the environment by incoming agents (cf. [4, 43, 79, 80, 131]). As such the idleness is sometimes viewed as the average (cf. [80]), worst-case (cf. [183]), probabilistic (cf. [3]) or experimentally verified (cf. [131]) time elapsed since the last visit of a node (cf. [4, 43]). Also, in some papers the terms of *blanket time* (cf. [183]) or *refresh time* (cf. [148]) are being used instead, so as to indicate a similar measure of algorithm efficiency. Several approaches to patrolling based on idleness criteria were surveyed in [4], including machine learning, negotiation mechanisms for generating paths, heuristics based on local idleness, as well as an approximation to the Travelling Salesmen Problem (TSP).

Some papers study patrolling based on swarm or ant-based algorithms (cf. [81, 133, 183]) and explore various agent capabilities (sensing, memory, locomotion, etc.). The *skeletonisation technique*, where a terrain is first partitioned into cells is often applied in geometric environments prior to employing graph-theoretic methods in discrete time.

In graph environments, cyclic strategies often rely on either TSP-related solutions or spanning tree-based approaches ([94, 148]). For the case of boundary patrolling where the agents maintain distinct maximal speeds partial solutions for small numbers of agents were proposed (cf. [62]).

One may also consider as a variant of patrolling the problem of searching a graph or polygon by teams of agents (cf. e.g. [89, 182]), which are looking for a stationary or mobile intruder. This falls into the vastly investigated domain of cops and robbers (see [88]).

The patrolling problem may be viewed as a version of an *art gallery* question, in which a set of stationary or mobile guards have to protect a given geometric environment (see [136, 145, 159]). In the setting with stationary guards, in most research papers the number of guards, needed to view the entire environment, has to be minimised. The problem is NP-hard and many approximation and inapproximability results were obtained (cf. [78, 98]). For the case of mobile guards, often known as the *watchman route problem*, the question of a single watchman was most often addressed. The optimisation criterion is the path length traversed by the watchman, so that every point of the environment is seen from some position on the path. This is closely related to the TSP. Unsurprisingly, many general watchman route problems are NP-hard (e.g. watchman tours of simple polygon with holes, suggested in [44] and corrected in [74]), touring a sequence of non-convex polygons [73], or link-distance watchman tours of simple polygon with holes [20]). However, for many specific cases polynomial-time algorithmic solutions are available. The solution for simple polygons was proposed by [45], while [46], [110] and [58] solve, respectively, the “zookeeper route”, “safari route” and “aquarium keeper” problems.

In the *m-watchmen routes* problem, the sum S of m path-lengths must be minimised, so that each point of the environment must be seen from some position of one of the watchmen (cf. e.g. [140, 147]). Clearly, S decreases with increasing m. Hence at one extremity, the case is obtained when m is large enough to have $S = 0$ (m stationary guards for art gallery are sufficient), and on the other extremity the single watchman question arises. As the m-watchmen routes problem is NP-hard for simple polygons (cf. [2]), some restricted classes of polygons were considered in [42, 141].

Central to the watchman route problem is the notion of *visibility*. Some papers (e.g. [110, 143]) considered limited visibility of the mobile (e.g. [143]) or stationary (e.g. [110]) guards. This work corresponds to a patrolling problem in the case of *zero visibility*, in which the agent sees only the point of the environment at which it is currently present.

As mentioned a boundary may be monitored by a number of different types of devices from static devices to completely mobile devices. In chapter 5 the scenario is shown in which only a finite number n of boundary partitions, referred to as *vital regions*, need to be patrolled by a set of k agents. The remaining part of the boundary, referred to as *neutral regions*, do not have to be monitored by the agents, but may nevertheless be traversed by an agent since this may be the way to reach one vital region from another.

Chapter 5 discusses the problem of patrolling with the goal of minimising the idleness of points located in the vital regions, i.e. the longest time during which such a point remains unvisited by an agent. It is assumed that at any time during the traversal, the speed of each agent cannot exceed a certain maximum value, identical for all agents. The goal of this work is to define a set of functions describing the trajectories of all the agents in time.

The most common heuristics adopted in the past to solve a variety of patrolling problems include the *cyclic strategy*, where agents move in one direction around the cycle covering the environment, and the *partition strategy*, in which the environment is partitioned into sections patrolled separately by individual agents (or subsets of agents), using the terminology introduced by Chevaleyre [43]. However, to the best of the authors' knowledge, no theoretical studies formally proving the optimality of such approaches in this setting were done previously.

It is worth noting, that in the more heterogeneous scenario where agents have different maximum speeds, neither the cyclic strategy nor the partition strategy leads to the optimal performance. In fact, it has been shown in [62] that for the case of 3 mobile agents with different maximal speeds patrolling a cycle (forming a single vital region), neither a partition strategy nor a cyclic strategy is optimal. It turns out that a specific hybrid strategy is better than each of these two fundamental approaches. See also [109].

2.4.3 Additional Work

In this section the topics that are to be discussed in Chapter 6 are introduced, specifically those of parasitic computation, the basic walk, and graph visualisation and analysis.

Parasitic Computation

The amount of computational power available to the planet is immense though it requires little thought to realise that a large volume of that power is currently unused, and essentially wasted. The amount of processing power that the average user has available to them is perhaps far surpassing their needs as a high performance processor is barely utilised for the purposes of browsing web pages or writing documents. For this reason it could be no surprise that many modern processors may be relatively idle for the duration of their life time, and as processors become more powerful along with the combination of more efficient algorithms, the time in which a processor is fully utilised will only decrease.

As was shown earlier in §2.1.1, there are methods to combat this through the use of large scale distributed computing projects, however, in this work a slightly different model is considered that may reduce some of the barriers to these projects, if implemented correctly and ethically. Specifically, it will be shown that through the use of existing technology and software algorithmic parasites can be developed that are able to spread across hosts and be executed unknowingly by the host to exploit unused resources for the parasite's originator. As mentioned this concept has existed, though it will be shown that it can be achieved without the requirement of installing additional software and becomes almost feasible to be executed by any types of hardware or software for as long as it has a modern standards compliant web browser installed. In Chapter 6 there is a further look into this model and the results from successfully building a prototype that allowed for the generation of the binary expansions of π. This work was inspired by work undertaken previously by the author in [51].

Basic Walk

Also inspired by [51], is another interesting aspect of this work that refers to developing new alternative approaches in time and energy efficient distributed communication. Imagine, e.g. an agent in a network at a source vertex s that is destined for another vertex, t. If the topological structure of the network is unknown one of the simplest methods for routing the agent is by utilising the *random walk* principle, whereby the agent at each vertex selects a neighbouring vertex at random and traverses towards it. This process is repeated until t is reached. Apart from the routing time that is likely to be excessive, either the agent or the network nodes (even in static networks) must have access to a Pseudo-Random Number Generator (PRNG), and though despite its simplicity can be shown as comparable to more intelligent techniques in some environments [92].

Two major and interesting alternatives to the random walk are the *rotor-router* (or *Propp machine*) [23, 24, 183] and the *basic walk* [59]. In many previous approaches, focus has been on the performance of one agent performing graph exploration, however, relatively little is currently known in the case where multiple agents are expected to collaborate within these models. The initial focus of this part of the work is on the ***random*** *basic walk*.

The significant feature of the basic walk is it allows for the formation of multiple directed cycles with at least one cycle of a size linear to $|V|$. The basic walk is performed on the network formed of a graph $G(V, E)$ where V is a set of vertices and E is a set of directed edges, and specifically the graph should hold the property that it is undirected. The network G is seen as a digraph where each vertex v has a set of incoming edges $v_{\overleftarrow{E}} \subset E$ and a set of outgoing edges $v_{\overrightarrow{E}} \subset E$. To enable the basic walk to be performed by an agent, each vertex must label each of its own $\overrightarrow{e}$ with a random unique label (port number) from the set of integers $\{0, \ldots, deg(v) - 1\}$.

To initiate the process, an agent first selects an outgoing edge $v_{\overrightarrow{u}}$ to a neighbouring vertex u (at random or through some other method) and traverses to the neighbouring vertex, and arriving through $u_{\overleftarrow{v}}$ ($v_{\overrightarrow{u}} \equiv u_{\overleftarrow{v}}$). At u, the agent must identify the associated port number i of the edge $u_{\overrightarrow{v}}$, i.e. that of the port number associated with the outgoing edge to v. Following this the agent can then select the next vertex to traverse to by identifying the $u_{\overrightarrow{e}}$ associated with the port number $(i + 1) \mod \deg(u)$.

Once this process has been repeated until an edge previously visited is found, a unique directed cycle will have been discovered. Should this process be repeated on all remaining unvisited edges, it will lead to the partitioning of a graph into a series of directed cycles (rings). This leads to the motivation of this work which is to apply the basic walk to any unknown general graph and reduce it to a series of rings, up on which more efficient algorithms can be utilised.

In §6.2 this model is explored in more detail and empirical results are shown when performing this algorithm on various types of generated and real world graphs. Theoretical ideas are also proposed as to why the results produced are created.

Graph Visualisation and Analysis

Through the work that was conducted in the previous section it became important to develop a software solution that would be capable of generating and processing graphs of various types. As the work advanced it became more critical that the software would be able to both import and export graphs, as well as handle graphs with millions of vertices or edges. To assist in verifying the correctness of algorithms developed for §2.4.3 a rendering component was created to allow for any graph to be visualised and interacted with.

While many existing products exist that are capable of this (E.g. Gephi [97], Cytoscape [57], and Tulip [21]), at the time development started no product was suitable for the needs of project particularly due to the requirements of port numbering, thus requiring a new product. Initially the project was known under it's development workspace name "GUI_Graph" due it being quite simply a graph with a Graphical User Interface (GUI). However, as the project evolved and the user gained the ability to interact and draw graphs themselves the project changed to "GraphDraw", a name which has so far stuck, though it would not be surprising to the author that should development continue this will inevitably have to change again. For the purposes of this thesis, the software that was constructed will be referred to as "GraphDraw".

As GraphDraw evolved to handle large graphs it became critical that several aspects were factored in. Firstly the software needed to be performance aware to allow for larger and larger graphs to be processed within reasonable time and memory costs. Particularly this requirement meant that the software could be used in desktop environments where processing and memory capacities could be constrained, thus demanding all algorithms adopted should be implemented as optimally as feasible through the use of high quality algorithms and good programming practice. Secondly, the visual component itself needed to be able to support very large graphs as the very purpose of the software would be to support the user in visually inspecting the correctness of graphs generated or algorithms executed to provide intuition into results obtained. Further, not only should the software render the graph but it should also allow for the user to interact with the graph with ease. Finally the software needed to support common data interchange formats to allow the user to quickly import and export their graphs with other tools.

Interestingly as the development advanced, GraphDraw caught the attention of Dollywagon Ltd., a media science organisation interested in the visualisation and analysis of social media data. Social networks and social media have recently become popular topics due to social networking web sites such as Facebook, Twitter, Linkedin and many others. Facebook [84] currently state they have 1.11 billion monthly active users, with a daily active reaching 665 million. While Linkedin and Twitter currently acknowledge they have over 200 million active users [142, 177].

More recently, while the joint work has continued, at the time of writing, with Dollywagon, GraphDraw has been adopted by other departments within the authors institution as potential replacement for the competing products listed above.

In §6.3 algorithms adopted by GraphDraw are discussed in terms of their effectiveness and potential future directions.

Chapter 3

Synchronous Rendezvous for Location-Aware Agents

This chapter introduces the results for performing synchronous rendezvous in various settings. This work was accepted to the 25th *International Symposium on Distributed Computing*, c.f. [53].

In this work it is assumed that the network environment is represented by a graph (finite or infinite) in which mobile agents visit nodes by traversing edges in both directions. The approach is also extended to the continuous, geometric setting. It is assumed that all agents move with a uniform speed. More precisely, a mobile agent requires a unit of time to traverse an edge of the graph, or a segment of length 1 in the geometric setting. Each agent can access its clock and the agents' clocks are synchronised to tick at the same time moments. An agent can count the number of rounds (clock ticks) and use this information to plan its actions. It is assumed that both agents start their actions simultaneously, i.e. both clocks indicate the same time. Agents are anonymous. However, each agent is aware of its initial location that can be adopted as its unique label. We also assume that the agents are fully aware of the network topology. The agents can meet each other on vertices of the graph, and they can pass through each other without meeting, while traversing an edge in opposite directions. Deterministic rendezvous procedures (i.e. no randomisation is allowed) are focused on. The movement of an agent is determined by its current location and the time on its clock.

Specifically for this work, the algorithms are developed with the intention of allowing two agents to perform rendezvous, however, some algorithms may permit (with minor modification) for more than two agents to perform rendezvous. Though more than two agent rendezvous is not considered in this work.

3.1 Linear Time Rendezvous on the Infinite Line

The first case that is considered here refers to the infinite line where line $L = (-\infty, +\infty)$. Firstly, note that the agents are aware of their initial location in the line and have a sense of (positive or negative) direction. Further, note that in this setting it is trivial to meet agents at, say, the origin of the line as the agents know their own location and thus the distance to the origin. This, however, will not guarantee local rendezvous. I.e. the agents may have to traverse a very long distance, much longer than the distance d that separates the initial positions of the agents.

Instead, the agents travel in a *zig-zag* fashion at increasing distances. For agents starting at distance d, this guarantees rendezvous in time $\mathcal{O}(d)$.

Initially it is assumed that all agents are originally situated at positions within $\mathbb{Z}$ in the line, and they all begin the rendezvous procedure at time $t = 0$. Later, the positions within $\mathbb{R}$ are also discussed.

To conduct rendezvous in the line, the algorithm is performed in a series of *iterations.*

Each iteration with the index $i \geq 1$ consists of two stages: *stage 1*, and *stage 2.* First, set $\ell_i = 2^{i-2}$, for $i \geq 2$. Informally speaking, during iteration i, the agents are initially divided into *odd* and *even* groups located at a distance of $2\ell_i$ from each other. The *odd* agents move to the right a distance of ℓ_i (stage 1), then to the left a distance of $2\ell_i$ (stage 2). The *even* agents move left a distance of ℓ_i (stage 1), then right a distance of $2\ell_i$ (stage 2). Thus, groups of agents meet both of their neighbouring groups, and then some of the groups will merge at the end of the iteration.

More formally, the following can be defined $F_1^c(k) = \{k\}$ for each integer k. Then, the following invariant can be demonstrated. During each iteration $i = 1, 2, 3, \ldots$, the agents are partitioned according to their initial positions on the line into the following groups at the end of stage 1 and stage 2:

1. $F_i^h(k) = \{k \cdot 2^i - 2^{i-1}, k \cdot 2^i - 2^{i-1} + 1, \ldots, k \cdot 2^i + 2^{i-1} - 1\}$ for $k \in \mathbb{Z}$, on the conclusion of stage 1.

2. $F_i^c(k) = \{k \cdot 2^i, k \cdot 2^i + 1, \ldots, k \cdot 2^i + 2^i - 1\}$ for $k \in \mathbb{Z}$, on the conclusion of stage 2.

These two groups F^h and F^c correspond to the location of the agents after each stage, specifically with F^h representing the positions of the agents *half-way* through completing an iteration (Stage 1) and F^c representing the positions of the agents after *completing* the iteration (stage 2).

Further there are two more invariants, namely:

3. $\mathit{label}(\alpha) = k$ for all agents $\alpha \in F_i^c(k)$, for all $i > 0, k \in \mathbb{Z}$.

4. At the end of iteration $i \geq 1$, the agents in group $F_i^c(k)$ are located at position $k \cdot 2^i + \frac{1}{2}(2^i - 1)$ on the line.

Theorem 3.1. *For two agents α_1, and α_2 starting at distance d (and at integer points) on the line L, Algorithm 1 permits rendezvous within at most* $6d$ *synchronised rounds.*

Proof. The four invariants mentioned above are easy to establish by induction.

First note that using the definitions of the sets $F_i^h(k)$ and $F_i^c(k)$ above that for all $i \geq 1$ and $k \in \mathbb{Z}$, the following would hold:

Algorithm 1: Rendezvous on the infinite line

```
$\ell \Leftarrow \frac{1}{2}$
**for all** $\alpha \in \mathbb{A}$ **do**
  label($\alpha$) $\Leftarrow$ position of $\alpha$ on the line
**end for**
**for** $i \Leftarrow 1, 2, 3, \ldots$ **do**
  **for all** $\alpha \in \mathbb{A}$ **do**
    **Stage 1** {form the groups $F_i^h(k)$}
    **if** *odd*(label($\alpha$)) **then**
      move right distance $\ell$
    **else**
      move left distance $\ell$
    **end if**
    **Stage 2** {form the groups $F_i^c(k)$}
    **if** *odd*(label($\alpha$)) **then**
      move left distance $2\ell$
    **else**
      move right distance $2\ell$
    **end if**
  **end for**
  $\ell \Leftarrow 2 \cdot \ell$
  label($\alpha$) $\Leftarrow \left\lfloor \frac{\text{label}(\alpha)}{2} \right\rfloor$
**end for**
```

$$F_i^h(k) = F_{i-1}^c(2k-1) \cup F_{i-1}^c(2k),$$

$$F_i^c(k) = F_{i-1}^c(2k) \cup F_{i-1}^c(2k+1).$$

Start with sets $F_1^c(k) = \{2k, 2k+1\}$ for all $k \in \mathbb{Z}$. Before the iteration with index 2, note that each $F_1^c(k)$ is located at position $2k$.

Inductively, the agents in the group $F_i^c(2k)$ (with label $2k$, by assumption) will first meet those in group $F_i^c(2k-1)$ (with label $2k-1$, again by assumption) in stage 1 of iteration $i+1$, and will then meet those in group $F_i^c(2k+1)$ (with label $2k+1$) in stage 2 of iteration $i+1$.

Now, since $label(F_i^c(2k)) = 2k$ (i.e. $label(\alpha) = 2k \ \ \forall \alpha \in F_i^c(2k)$) and $label(F_i^c(2k+1)) = 2k+1$, it can be found that all agents in $F_{i+1}^c(k) = F_i^c(2k) \cup F_i^c(2k+1)$ will have label k at the end of iteration $i+1$. Further, by assumption, the group $F_i^c(2k)$ begins iteration $i+1$ at location $2k \cdot 2^i + \frac{1}{2}(2^i - 1)$ in the line. During iteration $i+1$, this group first moves left a distance of 2^{i-1}, then right for a distance of 2^i. Hence, this group ends iteration $i+1$ at the location

$$2k \cdot 2^i + \tfrac{1}{2}(2^i - 1) - 2^{i-1} + 2^i = k \cdot 2^{i+1} + \tfrac{1}{2}(2^{i+1} - 1).$$

In a similar manner, it can be shown that group $F_i^c(2k+1)$ ends iteration $i+1$ at the exact same location as group $F_i^c(2k)$, the pair of them together comprising $F_{i+1}^c(k)$.

Finally, by the end of iteration $i \geq 2$, each agent has met all other agents that began the rendezvous procedure at distance at most 2^{i-1}, and each agent has moved a distance of $3\sum_{j=1}^{i} \ell_j$, where $\ell_j = 2^{j-2}$, as before.

Consider two agents that begin the rendezvous at positions $\alpha_1 < \alpha_2$. Then, let i be the integer such that $2^{i-1} < d = \alpha_2 - \alpha_1 \leq 2^i$. From the claim above, α_1 and α_2 have met by the end of iteration $i+1$. Thus, this process takes time

$$\begin{aligned} 2 + 3\sum_{j=2}^{i+1} \ell_j &= 2 + 3\sum_{j=2}^{i+1} 2^{j-2} = 2 + 3\sum_{j=1}^{i} 2^{j-1} = 2 + 3\cdot(2^i - 1) \\ &= 6\cdot 2^{i-1} - 1 < 6d \end{aligned}$$

□

Now consider non-integer starting positions of the agents. For $d \in \mathbb{R}_{>1}$, let us define the function $T(d) = 6\cdot 2^{i-1} - 1$, where i is the integer that satisfies $2^{i-1} < d \leq 2^i$. From Theorem 3.1, agents starting on integer points at distance d rendezvous in time at most $6d$. This can be carried over to arbitrary (non-integer) starting points and distances, at least in the case where $d \geq 1$.

Lemma 3.2. *Suppose two agents α_1, and α_2 are placed in the line L at distance $d \geq 1$. Then, Algorithm 1 can be adapted so that α_1 and α_2 are able to rendezvous within $T(\lceil d \rceil) < 6d$ synchronous rounds.*

Proof. The adaptation of Algorithm 1 is a natural one to consider. An agent not beginning at an integer point initially adopts as its first move as a contiguous final segment of the first move of a close by (possibly hypothetical) agent that did begin at an integer point. If the starting location of α is not an integer, then consider $\alpha - \lfloor \alpha \rfloor$ and $\lceil \alpha \rceil - \alpha$. Either one of these two quantities is smaller than the other, or they are equal, i.e. α is closer to one of two integers, or $\alpha = \lfloor \alpha \rfloor + \frac{1}{2}$. In the first case, α adopts the final segment of the first move of the (hypothetical) agent at $\lfloor \alpha \rfloor$ or $\lceil \alpha \rceil$, and then adopts the label of that agent and its behaviour for the remaining part of the rendezvous procedure. (If $\alpha = \lfloor \alpha \rfloor + \frac{1}{2}$, then α can arbitrarily adopt the label and procedure of $\lceil \alpha \rceil$.)

Now assume that $d > 1$. (The case of $d = 1$ is easily handled by a special analysis very similar to the one given below.) Assume that $\alpha_1 < \alpha_2$ and, as before, let $d = \alpha_2 - \alpha_1$. Let us write $d = d^* + \epsilon$, where $d^* = \lfloor d \rfloor$ and $\epsilon < 1$.

There are integers $x < y$ such that $d^* = y - x$

$$x - 1 < \alpha_1 \leq x < y \leq \alpha_2 < y + 1.$$

Since $\epsilon = (x - \alpha_1) + (\alpha_2 - y) < 1$, either $x - \alpha_1$ or $\alpha_2 - y$ is strictly smaller than $\frac{1}{2}$. Assume that $x - \alpha_1 < \frac{1}{2}$. (The other case is similar.)

In this case, by the end of iteration 1, α_1 has adopted the behaviour of (the hypothetical) agent x. Similarly, α_2 has adopted the behaviour of either agent y or $y+1$ (depending upon α_2's exact location in the interval $[y, y+1)$). This means that α_1 and α_2 will meet in the same time that it takes the agents at positions x and $x + \lfloor d \rfloor$ or x and $x + \lceil d \rceil$ to meet. In particular, this means that α_1 and α_2 will meet in time (at most) $T(\lceil d \rceil)$.

As before, let i be the integer such that $2^{i-1} < d \leq 2^i$. Then also note that $2^{i-1} < \lceil d \rceil \leq 2^i$. As has been noted already that α_1 and α_2 will meet in time $T(\lceil d \rceil)$ and since

$$\frac{T(\lceil d \rceil)}{\lceil d \rceil} \leq \frac{T(\lceil d \rceil)}{d} < \frac{6 \cdot 2^{i-1} - 1}{2^{i-1}} < 6$$

this establishes the bound of the lemma. □

In general, it is impossible to define a deterministic rendezvous procedures for agents that start at an arbitrarily small distance $d > 0$ and to guarantee that they will meet in time $\mathcal{O}(d)$. In the case where the agents are located very close to each other this algorithm guarantees rendezvous on the conclusion of iteration 2, i.e. in time 4.

3.2 Linear Time Rendezvous in Trees

In this section it is shown that the time complexity of rendezvous in trees is $\mathcal{O}(d)$. It is first shown that the two agents can meet in time $< 12d$ in the half-line $[0, +\infty)$, where they are allowed to move *only* in one direction towards the location 0. Later it is shown how to adopt this algorithm to obtain a $\mathcal{O}(d)$ time rendezvous in trees.

3.2.1 One-Way Rendezvous in the Half-Line

Consider a half-line $L' = [0, +\infty)$ where the agents α_1 and α_2 are labelled by their initial integer positions p_1 and p_2 respectively. Assume also that this time the agents can move only in one direction towards the closed end (0) of L'.

The algorithm is executed in iterations formed of stages 1 and 2. The following invariants are used. During each iteration $i = 2, 3, \ldots$ the agents are partitioned according to their labels. Specifically, notation F^h and F^c are re-used to to define invariants that will hold after each stage of this algorithm. F^h denotes the location of the agents after executing stage 1 of the algorithm (a *half iteration*), F^c denotes the location of the agents after executing stage 2 (a *full iteration*) of the algorithm:

1. $F_i^h(k) = \{(k+1)2^i - 2^{i-1} - 1, \ldots, (k+1)2^i + 2^{i-1} - 2\}$, for $k > 0$, and $F_i^h(0) = \{0, \ldots, 2^i + 2^{i-1} - 2\}$ on the conclusion of stage 1,

2. $F_i^c(k) = \{(k+1)2^i - 1, \ldots, (k+2)2^i - 2\}$, for $k > 0$, and $F_i^c(0) = \{0, \ldots, 2^{i+1} - 2\}$ on the conclusion of stage 2.

Furthermore, it is assumed that the agents with labels in $F_i^h(k)$ and $F_i^h(k)$ are aligned at position $k \cdot 2^i$ on the conclusion of respective stages. For the completeness of the argument observe that before the rendezvous algorithm is executed, $F_0^c(k)$ contains the agent with label k located at position $k \geq 0$. Also on the conclusion of iteration 1 each $F_1^c(k)$ contains agents with label $2k+1$ and $2k+2$ located at position $2k$ on the line.

Algorithm 2: One-way rendezvous

```
for all $\alpha \in \mathbb{A}$ do
  label($\alpha$) $\Leftarrow$ position of $\alpha$ in the line
end for
for $i = 1, 2, 3, \ldots$ do
  stage 1: form $F_i^h(k) = F_{i-1}^c(2k) \cup F_{i-1}^c(2k+1)$ by moving agents grouped in
  $F_{i-1}^c(2k+1)$ by $2^{i-1}$ positions towards 0
  stage 2:
  if $k > 0$ then
    form $F_i^c(k) = F_{i-1}^c(2k+1) \cup F_{i-1}^c(2k+2)$
  else
    form $F_i^c(k) = F_i^h(k) \cup F_{i-1}^c(2k+2)$ by moving agents grouped in $F_{i-1}^c(2k+2)$
    by $2^i$ positions towards 0
  end if
end for
```

Proposition 3.3. *Algorithm 2 has the* enclosure property, *i.e. for any three agents α_1, α_2 and α_3 located initially at positions $0 \leq p_1 < p_2 < p_3$ when agents α_1 and α_3 meet, they also meet α_2.*

Proof. The enclosure property is a straightforward consequence of the fact that groups of agents formed on the conclusion of stages 1 and 2 form partitions in which each group is a contiguous segment of positions in L'. □

Using reasoning similar to the proof of Theorem 3.1 the following is obtained.

Theorem 3.4. *Two agents α_1, and α_2 executing Algorithm 2 and initially located at distance d, on integer points in the half-line $L' = [0, +\infty)$, require at most $12d$ synchronised rounds to rendezvous.*

3.2.2 Rendezvous in trees

In this section it is assumed that the agents α_1 and α_2 are located at some two vertices p_1 and p_2 in a finite tree T. The vertices in the tree are uniquely identified and the agents are aware of the entire structure of the tree, which is held in memory[1]. This allows the agents to select independently a unique vertex in T that becomes the root r of T. Assume that d_1 and d_2 are respective distances from p_1 and p_2 to r. Without loss

[1]However, while not discussed specifically in this work as the focus is on time rather than memory, it would be feasible to reduce this memory requirement as in principle the agents only need to know the direction of the (path towards the) root vertex.

of generality, assume that $d_2 \leq d_1$. Let x be the Lowest Common Ancestor (LCA) for p_1 and p_2 in T with respect to the root r, where d_x is the distance separating x from r. For the purpose of rendezvous, the vertices in the tree adopt their distances to r as their labels. For example, the root r adopts the label 0 and vertex x adopts the label d_x. The new labels d_1 and d_2 of the vertices p_1 and p_2 are also adopted by the agents α_1 and α_2, respectively. In order to rendezvous, the agents execute Algorithm 2 designed for the half-line $L' = [0, +\infty)$ moving gradually towards the root r with the label 0.

Note that if $x = p_2$, i.e. vertex p_2 is located on the route from p_1 to r, the distance between p_1 and p_2 is $d = d_1 - d_x$, and according to Theorem 3.4 the rendezvous process will be completed in time $12d$. Otherwise, the initial distance between p_1 and p_2 in T is $d = d_1 + d_2 - 2d_x$.

Let a vertex p'_2 located on the path from p_1 to r, at distance d_2 from r be the starting position of a hypothetical agent α'_2. Note that during the execution of Algorithm 2 agent α'_2 acts the same way as α_2, and in particular the distances between r and α_2 as well as r and α'_2 are always the same. Assume also that vertex x is a starting position of another hypothetical agent α_x.

Due to the enclosure property, see Corollary 3.3, during the execution of Algorithm 2 when the agents α_x and α_1 meet they also meet α'_2. But since the moves of α_2 and α'_2 are identical and all of the agents move only towards the root r, when agents α_x and α_1 meet, they also meet α_2. Thus according to Theorem 3.4 agents α_1 and α_2 rendezvous in time $12(d_1 - d_x) < 12(d_1 + d_2 - 2d_x) = 12d$. The following theorem follows.

Theorem 3.5. *Two agents, α_1 and α_2 executing Algorithm 2 initially located at distance d on the vertices of a rooted tree T can rendezvous in $\leq 12d$ synchronous rounds.*

3.3 Rendezvous in the Higher-Dimensional Space

In this section an algorithm is presented that produces the paths of two agents, placed in δ-dimensional grid, which achieve rendezvous in optimal $\mathcal{O}(d)$ time, where d denotes the rectilinear (i.e. ℓ^1) distance between the original positions of the agents. Observe that this result may be used to achieve rendezvous for agents starting at arbitrary initial positions in the δ-dimensional space.

In order to give an efficient synchronous rendezvous algorithm recall, following [25], the concept of the sequence of *central space partitions* $\Pi = \pi_1, \pi_2, \ldots$. Each π_i is

1. A partition of δ-dimensional space into hypercubes of side length 2^i.
2. The hypercubes are aligned with the axis of the space so they form a δ-dimensional grid.
3. One of these hypercubes is the *central hypercube*, having as its centre the origin of the δ-dimensional Cartesian space.

Observe that the corners of hypercubes in π_i are points $u = (u_1, u_2, \ldots, u_\delta)$ such that $\forall r \in \{1, 2, \ldots, \delta\}$, $u_r = 2^{i-1} + k2^i$ for some integer k. Note that all the $(\delta - 1)$-dimensional hyperplanes used in all the partitions of Π are different. To assure that each π_i forms an exact partition assume that each hypercube H contains, besides its interior points, the corner v having maximum co-ordinates, as well as all open f-faces incident to v, for $f = 1, 2, \ldots, \delta - 1$.

The idea of the rendezvous algorithm is to direct each agent through a sequence of some potential *meeting points*. These meeting points are the centres of hypercubes of increasing sizes belonging to the successive partitions $\pi_1, \pi_2, \ldots$. The hypercubes are chosen in such a way that the path traversed by each agent is not too long, and that it occurs that eventually both of the agents reach the same meeting point. Since the agents will traverse, in general, different distances to reach the successive meeting points, their movements will be synchronised with the aid of some waiting periods. A couple of lemmas will serve to prove the correctness and the time complexity of this approach.

Lemma 3.6. *Any hypercube H located in partition π_i intersects the set S of 3^δ hypercubes belonging to partition π_{i-1}. A centre of any hypercube from set S is at a distance of at most $\delta \cdot 2^{i-1}$ from the centre of H.*

Proof. It can be assumed, by symmetry, that H is the central hypercube of partition π_i and s is any of its sides. Observe that the middle point of side s coincides with the centre of some hypercube of π_{i-1}. Since the side length of each hypercube of π_i equals 2^i, which is twice the side length of hypercubes of π_{i-1}, s intersects exactly three hypercubes of π_{i-1}. By induction on dimension, all hypercubes of π_{i-1} intersected by H form a hypercube G of side length $3 \cdot 2^i$, i.e. a hypercube whose volume is $3^\delta \cdot 2^{\delta i}$. Hence, G is a union of 3^δ hypercubes of the partition π_{i-1}, each one of the volume of $2^{\delta i}$.

To prove the second part of the claim, note that the centre of each hypercube of π_{i-1} which intersects H belongs to the closure of H. The shortest rectilinear path from the boundary of H to its centre is maximised when the path starts at a vertex of H. Since the length of the side of H equals 2^i, one of its vertices has all δ Cartesian co-ordinates equal to 2^{i-1}. An agent, moving along the shortest rectilinear path from such vertex to the centre of H, in each synchronous round moves from a point x to a point y, such that y has the same coordinates as x, except one coordinate which is reduced by one (with respect to this coordinate for point x). Hence $\delta \cdot 2^{i-1}$ rounds are needed in order to reach the origin (the centre of H). □

The following lemma can be derived from Lemma 3 in [25].

Lemma 3.7. *For any pair of points p_1 and p_2, initially placed at rectilinear distance d in the δ-dimensional grid, such that $d \leq 2^i$, for some $i = 1, 2, \ldots$ there exists a hypercube H of size at most $2^{i+\delta+1}$ belonging to the hierarchy of partitions Π, such that H contains both points p_1, p_2.*

Now an algorithm which achieves rendezvous in linear time of the original distance between the two agents is given.

Algorithm 3: Rendezvous in the δ-dimensional grid

```
$i \Leftarrow 0$
while $\neg$rendezvous do
  $H \Leftarrow$ hypercube of $\pi_i$ containing your initial position $p$
  move to the centre of $H$
  wait until $\delta \cdot 2^{i-1}$ rounds are completed since the start of the current iteration
  $i \Leftarrow i+1$
end while
```

The agent's trajectory produced by Algorithm 3 is in $\mathcal{O}(d)$ is now proven, when d is original distance between the agents, i.e. that Algorithm 3 is optimal (up to a multiplicative constant).

Theorem 3.8. *Suppose that two location-aware agents are placed at rectilinear distance d in the δ-dimensional grid and the agents simultaneously start their movements produced by Algorithm 3. Then the rendezvous of both agents is achieved within $d \cdot \delta \cdot 2^{\delta+2}$ synchronous rounds.*

Proof. Let i^* be an integer, such that $2^{i^*-1} < d \leq 2^{i^*}$.

Observe that, in the first iteration of the loop of Algorithm 3, the centre of the hypercube of side length 2 belonging to π_1 is reached from the initial position of the agent contained within this hypercube in at most δ rounds. By Lemma 3.6, within the i^{th} iteration of Algorithm 3, $\delta \cdot 2^{i-1}$ synchronous rounds are sufficient to reach the centre of the hypercube H. Therefore the waiting time is sufficiently long, so that at round number $\delta \cdot 2^{i-1}$ of the i^{th} iteration, both agents are mutually present at the centres of the corresponding hypercubes (despite the fact that the original distance to the centre was different for each of them). Hence if two agents aim for the centre of the same hypercube at the i^{th} iteration, they have to meet there before the completion of the iteration. By induction on the iteration number, both of the agents are synchronised so that they start and finish their movement of each subsequent iteration at the same moments of time. By Lemma 3.7, both of the agents eventually aim for the centre of the same hypercube H, whose side length equals at most $2^{i^*+\delta+1}$, and they meet there. Such hypercube H belongs to partition $\pi_{i^*+\delta+1}$, hence its centre is reached by the agents in iteration $i^* + \delta + 1$. The number of rounds spent by each agent until the end of iteration $i^* + \delta + 1$ equals

$$\textstyle\sum_{1 \leq i \leq i^*+\delta+1} \delta \cdot 2^{i-1} < \delta \cdot 2^{i^*+\delta+1} < d \cdot \delta \cdot 2^{\delta+2} \quad .$$

□

Algorithm 3 may be adapted to achieve rendezvous in δ-dimensional space. It is sufficient that the agents construct a common grid, each agent moves first to the closest grid position and then they continue their movements according to Algorithm 3. The cost of such algorithm becomes $d \cdot \delta \cdot 2^{\delta+2} + \delta$, where δ extra rounds are used first by each agent to reach the closest grid position.

Corollary 3.9. *Suppose that two location-aware agents are placed at rectilinear distance d in the δ-dimensional space. The rendezvous of the agents may be achieved within $\delta(d \cdot 2^{\delta+2} + 1)$ synchronous rounds.*

For the Euclidean distance case, when the agents do not need to walk along the axis of the δ-dimensional space, a finer grid may be constructed, so each agent may reach a grid point in a single step and the time given by Corollary 3.9 reduces to $\delta \cdot d \cdot 2^{\delta+2} + 1$. For this purpose it is sufficient to scale down the grid by at least a factor of $\frac{\sqrt{\delta}}{2}$.

3.4 Rendezvous in Arbitrary Graphs

In this section it is assumed that the agents α_1 and α_2 are located at some two vertices in a finite graph. In the same discrete model as for the earlier studied case of trees, the agents are aware of the topology of the graph and all nodes are uniquely identified. Initially it will be shown that unlike in all the cases discussed so far, there exist graph instances which require $\Omega(d)$ time to achieve rendezvous. In fact, rendezvous time of $\Omega(d\frac{\log n}{\log\log n})$ is required for graphs of extremal girth, with logarithmic average degree. In the following, all logarithms are assumed to be base 2.

Theorem 3.10. *Let ϵ be arbitrarily fixed. For any integers $N > 8$ and $d > 0$, such that $d < \min\{N^{1-\epsilon}, N/8\}$, there exists a graph of order N, such that for any rendezvous algorithm $\mathcal{A}$ there is a pair of starting locations at distance d such that rendezvous using algorithm $\mathcal{A}$ requires at least $\frac{\epsilon}{4}\frac{d\log N}{\log\log N}$ rounds, even when assuming a simultaneous start.*

Proof. From [34, Theorem III.1.1] it follows that for any integer $k > 3$, there exists a graph $G = (V, E)$ with $n = 2^k$ vertices, having $m = \frac{1}{2}nk$ edges, whose shortest cycle is of length $g > k/\log k + 1$. Let $\mathcal{A}$ be any algorithm defining the behaviour of an agent. Firstly it is shown that there exists a pair of neighbouring vertices in G such that the time required for the rendezvous of the agents starting from this pair of vertices (with $d = 1$), using algorithm $\mathcal{A}$, is at least equal to $\min\{(g-1)/2, k/2\}$. Suppose, to the contrary, that for all possible initial locations of the agents in G, the agents rendezvous within some time $t < \min\{(g-1)/2, k/2\}$. For all $v \in V$, let $A_v \subseteq G$ be the sub-graph spanned by the edges visited by an agent following algorithm $\mathcal{A}$, starting from vertex v, during the first t rounds, under the assumption that the agent does not meet the other agent. Each graph A_v has at most t edges. For any pair of adjacent vertices u, v of G, the intersection of graphs A_u and A_v must contain at least one vertex w; otherwise, no vertex will be visited by both of the agents starting at u and v, hence such agents cannot meet. Both of the graphs A_u and A_v are connected and of diameter at most $t < (g-1)/2$. Consequently, it must hold that $w \in \{u, v\}$, since otherwise G would contain a cycle of length less than g, obtained by traversing the shortest path from v to w in A_v, traversing the shortest path from w to u in A_u, and finally traversing the edge $\{u, v\}$. It follows that u is a vertex of A_v, or v is a vertex of A_u; without loss of generality, assume the former case. Then, the edge $\{u, v\}$ belongs to A_v, since otherwise the shortest path connecting u and v in A_v would form a cycle with the edge $\{u, v\}$ with a length less than g. Since u and v were arbitrarily chosen, it follows that each edge $e \in E$ belongs to some graph A_v, i.e. $e \in \bigcup_{v \in V} E(A_v)$. Consequently, $\sum_{v \in V} |E(A_v)| \geq m$, and so, there must exist a vertex v such that $|E(A_v)| \geq m/n \geq k/2$. However, this is a contradiction with $|E(A_v)| \leq t < k/2$. It follows that $t \geq \min\{(g-1)/2, k/2\} \geq k/(2\log k)$. Given a value of N, consider an input graph on N vertices consisting of graph G for $k = \lfloor \log N \rfloor$ (with $n = 2^k$), and $N - n$ additional vertices, attached with single edges to an arbitrarily chosen vertex of G. Since this modification does not affect rendezvous time, for any value of N a lower-bound of $\frac{1}{2}\frac{\lfloor \log N \rfloor}{\log \lfloor \log N \rfloor}$ is obtained on rendezvous time for agents starting from neighbouring vertices, i.e. for $d = 1$.

To prove the claim of the theorem for larger values of d, we construct a graph $G' = (V', E')$ from $G = (V, E)$ by inserting a path of $d-1$ vertices on each edge of G. Graph G' has $n' = n + m(d-1)$ vertices and $m' = md$ edges. Let $\mathcal{A}$ be a fixed algorithm for an agent which always reaches rendezvous in G' in time at most $t < d \min\{(g-1)/2, k/2\}$, and let $A'_v \subseteq G'$, for $v \in V$, be defined as the sub-graphs spanned by the edges visited by an agent following algorithm $\mathcal{A}$, starting from vertex v during the first t rounds, under the assumption that the agent does not meet the other agent. By an argument similar to that for the case of $d = 1$, we have that for all $e' \in E'$, $e' \in \bigcup_{v \in V} E(A'_v)$, and consequently, there exists a vertex $v \in V$ such that $|E(A'_v)| \geq m'/n = dm/n \geq dk/2$. This is a contradiction with $|E(A'_v)| \leq t < dk/2$. In this way we obtain a lower-bound of $dk/(2\log k)$ on rendezvous time in G'. Given a value of N, we consider an input graph on N vertices consisting of graph G' for $k = \lfloor \log(N/d) \rfloor$ (with $n' = d2^k$), and $N - n'$ additional vertices, attached with single edges to an arbitrarily chosen vertex of G'. Since this modification does not affect the rendezvous time, we obtain for any value of N a lower-bound of $\frac{1}{2}\frac{d\lfloor \log(N/d) \rfloor}{\log \lfloor \log(N/d) \rfloor} > \frac{\epsilon}{4}\frac{d \log N}{\log \log N}$ on the rendezvous time for the agents starting at distance d, where it has been taken into account that $d < N^{1-\epsilon}$. □

In conclusion, we point out that a poly-logarithmic overhead in n is sufficient to achieve rendezvous, assuming the simultaneous starting of the agents.

Theorem 3.11. *Suppose that two location-aware agents are placed at a distance of d in a known arbitrary graph $G = (V, E)$ of order n. Then the agents which start simultaneously can rendezvous within $\mathcal{O}(d \log^2 n)$ synchronous rounds.*

Theorem 3.12. *Suppose that two location-aware agents are placed at a distance of d in a known graph $G = (V, E)$ of order n. Then the rendezvous of agents with simultaneous starting is achieved within $\mathcal{O}(d)$ synchronous rounds if G is a circular-arc graph, and within $\mathcal{O}(d \log n)$ synchronous rounds if G is a chordal graph or a co-comparability graph.*

Chapter 4

Asynchronous Rendezvous With Location Information

This chapter introduces the results for performing asynchronous rendezvous when the agents have access to location information. This work was accepted to the 37th *International Colloquium on Automata, Languages and Programming*, cf. [52].

In this work efficient rendezvous of two mobile agents moving asynchronously in the Euclidean 2D space is studied. Each agent has limited visibility, permitting it to see its neighbourhood at unit range from its current location. Moreover, it is assumed that each agent knows its own initial position in the plane given by its coordinates. The agents, however, are not aware of each others position. The agents possess coherent compasses and the same unit of length, which permit them to consider their current positions within the same system of coordinates. The cost of the rendezvous algorithm is the sum of lengths of the trajectories of both agents. This cost is taken as the maximum over all possible asynchronous movements of the agents, controlled by the adversary.

An algorithm is proposed that allows the agents to meet in a local neighbourhood of diameter $\mathcal{O}(d)$, where d is the original distance between the agents. This seems rather surprising since each agent is unaware of the possible location of the other agent. In fact, the cost of this algorithm is $\mathcal{O}(d^{2+\varepsilon})$, for any constant $\varepsilon > 0$. This is almost optimal, since a lower bound of $\Omega(d^2)$ is straightforward. The only up to date paper [61] on asynchronous rendezvous of bounded-visibility agents in the plane provides the feasibility proof for rendezvous, proposing a solution exponential in the distance d and in the labels of the agents. In contrast, it is shown here that, when the identity of the agent is based solely on its original location, an almost optimal solution is possible.

An integral component of this solution is the construction of a novel type of non-simple space-filling curves that preserve locality. An infinite curve of this type visits specific grid points in the plane and provides a route that can be adopted by the mobile agents in search for one another. This new concept may also appear counter-intuitive in view of the result from [99] stating that for any simple space-filling curve, there always exists a pair of close points in the plane, such that their distance along the space-filling curve is arbitrarily large.

4.1 The Problem and the Model

A pair of identical agents are located at two points in the plane. Each of the agents have limited visibility, permitting them to see their neighbourhood at unit range from their current location. In this work it is assumed that each agent knows its own *initial position* in the plane given by its co-ordinates, i.e. it is *location aware*. However, each agent does not know the position of the other agent. It is also assumed that the agents possess coherent compasses and the same unit of length, which permit them to consider their current positions within the same system of coordinates. Therefore each agent may consider its initial location as its unique identity.

The *route* of each agent is a sequence of segments which are subsequently traversed during its movement. The entire route of the agent depends uniquely on its initial position. The actual *walk* of each agent along every segment is *asynchronous*, i.e. it is controlled by an adversary. The agents meet if they eventually get within the visibility range of each other, i.e. at some point in time the distance between their current positions will not be greater than one. Note that this work uses the continuous geometric model, and thus visibility is provided to ensure that the agents are able to meet within the discrete 2D grid that this algorithm utilises, i.e. to allow for agents to meet as they traverse into and out of a meeting point.

Next the power of the adversary is more precisely defined. The adversary initially places both agents at any two points in the plane. Given its initial location a_0, the route chosen by the agent is a sequence of segments $(e_1, e_2, \ldots)$, such that in stage i the agent traverses segment $e_i = [a_{i-1}, a_i]$, starting at a_{i-1} and ending at a_i. Stages are repeated indefinitely until rendezvous. It is assumed that each agent may start its walk at any time, but both of the agents are placed by the adversary at their respective initial positions at the same moment, and since at that time any moving agent may find the other agent, even if the other agent did not start its walk yet.

The walk f of an agent on its route, similarly as in [61]: let $R = (e_1, e_2, \ldots)$ be the route of an agent. Let $(t_1, t_2, \ldots)$, where $t_1 = 0$, be an increasing sequence of reals, chosen by the adversary, that represent points in time. Let $f_i : [t_i, t_{i+1}] \to [a_i, a_{i+1}]$ be any continuous function, chosen by the adversary, such that $f_i(t_i) = a_i$ and $f_i(t_{i+1}) = a_{i+1}$. For any $t \in [t_i, t_{i+1}]$, the following is defined, $f(t) = f_i(t)$. The interpretation of the walk f is as follows: at time t the agent is at the point $f(t)$ of its route.

The adversary may arbitrarily vary the speed of the agent, as long as the walk of the agent in each segment is continuous, eventually bringing the agent from the starting endpoint to the other endpoint of the corresponding segment of its route[1].

[1]This defines a very powerful adversary. Notice that the presented algorithm is valid even with this powerful adversary, and the lower bound argument works also for a weaker adversary that can only

The agents with routes R_1 and R_2 and with walks $f^{(1)}$ and $f^{(2)}$ meet at time t, if points $f^{(1)}(t)$ and $f^{(2)}(t)$ are identical. A rendezvous is guaranteed for routes R_1 and R_2, if the agents using these routes meet at some time t, regardless of the walks chosen by the adversary. The cost of the rendezvous algorithm is measured by the sum of the lengths of the trajectories of both agents from their starting locations until the time t of the rendezvous. Since the actual portions of these trajectories may vary depending on the adversary, the maximum of this sum can be considered, i.e. the worst-case over all possible walks chosen for both agents by the adversary. In this chapter the intention is to identify the rendezvous algorithm of the smallest possible cost with respect to the original distance d between the agents.

4.2 Efficient Construction of Space-Covering Sequences

An interesting context of this work is its relationship to *space-filling curves* extensively studied at various contexts in the literature, see, e.g. [40, 99, 137, 180]. One of the most important attributes of space-filling curves is sometimes called the *preservation of locality*, i.e. that if two points are close in the 2D space they are also closely located on the space-filling curve. In this context, however, Gotsman and Lindenbaum [99] pointed out that space-filling curves fail in preserving the locality in the worst case. They show that, for any space-filling curve, there always exist some close points in the 2D space that are arbitrarily far apart on the space-filling curve. In this work it is shown that such deficiency of space-filling curves comes from a very strong assumption that curves visit each point in a discrete 2D space exactly once. An alternative to space-filling curves is proposed that preserves locality also in the worst case. Namely, *space-covering sequences* will be introduced, that traverse points in a discrete 2D space multiple times. It can be shown that, for any $\varepsilon > 0$, there exists a space-covering sequence, such that, for any two points located at distance d in the 2D space there are well defined and efficiently computable instances of these two points in the sequence at distance $\mathcal{O}(d^{2+\varepsilon})$ apart.

The trajectories constructed in the work follow grid lines except for their initial segments by which each agent, starting from its *arbitrary* initial position, reaches its closest grid point. It is possible that the first agent traverses an arbitrarily long portion of its route, while the adversary holds the other agent on its initial segment being not visible from any grid line. To prevent this it is assumed, without loss of generality, that the 2D space is rescaled so that the value of $\frac{\sqrt{2}}{2}$ corresponds to the unit length of the 2D space. This way the considered grid is fine enough, and the agent visiting an integer grid point v will see another agent, situated in any interior point of a unit grid square with vertex v.

The fundamental concept used in the design of the rendezvous algorithm is the *space-covering sequence* on which both agents are walking until rendezvous. The space-covering sequence is infinite in both directions. It is formed of short segments with the endpoints

speed up or slow down the agent, without moving it back (corresponding to a walk function f which must be additionally monotonous). Hence these results also hold in this (perhaps more realistic) model.

located at the integer grid points. Every point of the integer grid is visited by the space-covering sequence infinitely many times. Each agent starting from its arbitrary initial position in the plane walks first to the closest grid point and then it starts making a special zig-zag movement on the sequence, each time covering more and more distance. The actual points at which the agent changes the direction of its movement are determined by the co-ordinates of the initial position of the agent, and the purpose of this algorithm is to determine them so that an efficient rendezvous will always be possible.

The construction of the space-covering sequence utilises a hierarchy of infinite grids of squares, of increasing sizes. This hierarchy is an amalgamate $\mathcal{H}_{QC}$ of two complementary hierarchies of square partitions: the *quad-tree partition hierarchy* $\mathcal{H}_Q$ and the *central square partition hierarchy* $\mathcal{H}_C$. For the clarity of presentation an argument leading to the rendezvous algorithm is demonstrated, of cost $\mathcal{O}(d^4)$ and later it is extended to obtain a $\mathcal{O}(d^{2+\varepsilon})$ cost solution.

Definition 4.1 (Quad-tree hierarchy $\mathcal{H}_Q$)**.** The first hierarchy of partitions $\mathcal{H}_Q$ has the form of an infinite *quad-tree* like structure, (for information on quad-trees see, e.g. [154]) in which the central point of the partition from each layer is aligned with the origin $(0,0)$ of the plane. The i^{th} layer L^i_Q of the hierarchy, for $i = 0,1,2,\ldots$, is formed of an infinite grid of squares of size 2^i. Hence the lowest level of the hierarchy $\mathcal{H}_Q$ corresponds to the standard integer grid. In layer L^i_Q a square $S^i_Q(x,y)$ with the corners located at points (listed in the clockwise order counting from the top-left corner) is denoted by, $(x \cdot 2^i, y \cdot 2^i)$, $((x+1)\cdot 2^i, y\cdot 2^i)$, $((x+1)\cdot 2^i, (y-1)\cdot 2^i)$ and $(x\cdot 2^i, (y-1)\cdot 2^i)$.

The overlapping squares at two neighbouring layers L^i_Q and L^{i+1}_Q are engaged in a parent-child relationship. In particular, a square $S^{i+1}_Q(x,y)$ at layer L^{i+1}_Q has four children squares $S^i_Q(2x,2y), S^i_Q(2x+1,2y), S^i_Q(2x+1,2y-1)$ and $S^i_Q(2x,2y-1)$ at the layer L^i_Q, for $i = 0,1,2,\ldots$

Definition 4.2 (Central-square hierarchy $\mathcal{H}_C$)**.** The second hierarchical partition $\mathcal{H}_C$ is formed of (infinitely many) enumerated layers, where the i^{th} layer L^i_C is an infinite grid with squares of size 2^i, for $i = 1,2,\ldots$. Each layer in $\mathcal{H}_C$ is aligned, such that, the origin $(0,0)$ of the plane is associated with the centre of one of the squares in this layer.

In particular, in layer L^i_C a square $S^i_C(x,y)$ with the corners located at points (listed in the clockwise order counting from the top-left corner) is denoted by $((x\cdot 2^i)-2^{i-1}, (y\cdot 2^i)+2^{i-1})$, $((x\cdot 2^i)+2^{i-1}, (y\cdot 2^i)+2^{i-1})$, $((x\cdot 2^i)+2^{i-1}, (y\cdot 2^i)-2^{i-1})$, and $((x\cdot 2^i)-2^{i-1}, (y\cdot 2^i)-2^{i-1})$.

Definition 4.3 (Hierarchy $\mathcal{H}_{QC}$)**.** By $\mathcal{H}_{QC}$ it can be understood that the infinite sequence of plane partitions

$$\pi_1 = L^0_Q, \pi_2 = L^1_C, \pi_3 = L^1_Q, \pi_4 = L^2_C, \pi_5 = L^2_Q, \ldots$$

Hence π_i is a grid partition of the 2D space into squares of size $2^{\lfloor i/2 \rfloor}$, with a grid point having each coordinate of the form $2^{\lfloor i-1/2 \rfloor} + k2^{\lfloor i/2 \rfloor}$, for any integer k.

To assure that each layer of $\mathcal{H}_{QC}$ forms an exact partition, it can be assumed that each square contains, besides its interior points, its right and top open sides as well as the top-right vertex.

Intuitively, the hierarchical partition $\mathcal{H}_{QC}$ provides a mechanism used to guarantee that any two points p_1 and p_2 located at distance d in the 2D space are covered by some square of size $\mathcal{O}(d)$ in either $\mathcal{H}_Q$ or in $\mathcal{H}_C$. The smallest square in the hierarchical structure with this property is referred to as the *rendezvous square* $R(p_1, p_2)$.

Less formally, the H_{QC} can be considered as two infinite trees which overlap. The first tree consists of the well known quad-tree structure. The idea of the quad tree is that for any unit point within the space, there exists a level of the tree representing this point. At higher levels of the tree, the same unit point is amalgamated with four neighbouring points of the original space to form a square. As the levels in the tree increase, the number of points represented by the branch of the tree quadruple as neighbouring squares are merged as the level increases. Secondly, the other tree is a central tree structure, this structure varies from the quad tree in that while the lowest level of the tree encapsulates the unit point of the space, the higher levels of the tree intersect nine squares from the previous level. The combination of these trees enable an agent to effectively zig-zag in its local space yet while also intersecting the local space of other agents, a concept that would not be possible with one of the trees alone. Figure 4.1 will show the tree structure in more detail later.

The following two lemmas directly continue from the definitions of $\mathcal{H}_Q$ and $\mathcal{H}_C$.

Lemma 4.4. *Any square $S^i_C(x, y)$ located in layer L^i_C, for $i = 1, 2, \ldots$, encapsulates exactly four squares $S^{i-1}_Q(2x-1, 2y+1)$, $S^{i-1}_Q(2x, 2y+1)$, $S^{i-1}_Q(2x, 2y)$, and $S^{i-1}_Q(2x-1, 2y)$, in layer L^{i-1}_Q.*

Lemma 4.5. *Any square $S^i_Q(x, y)$ located in layer L^i_Q, for $i = 1, 2, \ldots$, overlaps with exactly four squares $S^i_C(x, y)$, $S^i_C(x+1, y)$, $S^i_C(x+1, y-1)$ and $S^i_C(x, y-1)$ in layer L^i_C.*

The following tree-like structure $\mathcal{T}_{QC}$ is useful to visualise the functioning of this approach: each square of every layer of $\mathcal{H}_{QC}$ is a node of $\mathcal{T}_{QC}$. For every such square S from layer $i > 1$ of $\mathcal{H}_{QC}$ the squares from layer $i-1$ intersecting S are children of S in $\mathcal{T}_{QC}$. By lemmas 4.4 and 4.5 $\mathcal{T}_{QC}$ is a quaternary tree. $\mathcal{T}_{QC}$ is infinite (does not a have a root) and its leaves are unit squares of the integer grid.

The observation stated in the following lemma is the basis of the complexity proof of this algorithm.

Lemma 4.6. *For any two points p_1 and p_2 located at distance d in the 2D space there exists a square in $\mathcal{H}_{QC}$ of size $\mathcal{O}(d)$ that contains p_1 and p_2.*

Proof. Assume that $2^{i-1} < d \leq 2^i$. Consider five consecutive layers from $\mathcal{H}_{QC}$, L^i_Q, L^{i+1}_C, L^{i+1}_Q, L^{i+2}_C, and L^{i+2}_Q, where i meets the condition stated. Since any layer in $\mathcal{H}_{QC}$ forms a partition of the 2D space into squares, within the layer L^{i+2}_Q there exists a unique square $S_Q^{i+2}(x,y)$ containing p_1. Since four quad-tree children of $S_Q^{i+2}(x,y)$ partition it, exactly one of them, a square belonging to L_Q^{i+1} also contains p_1. Similarly, there is exactly one square from L_Q^i, one of the sixteen grandchildren of $S_Q^{i+2}(x,y)$ in the quad-tree, which contains p_1, see Figure 4.1. Let $S^* \in L_Q^i$ denote the square containing p_1. Note that, since $d \leq 2^i$, the d-neighbourhood of S^* intersects nine squares of L_Q^i - the square S^* itself and eight squares surrounding S^*. Hence p_2 must belong to one of these nine squares.

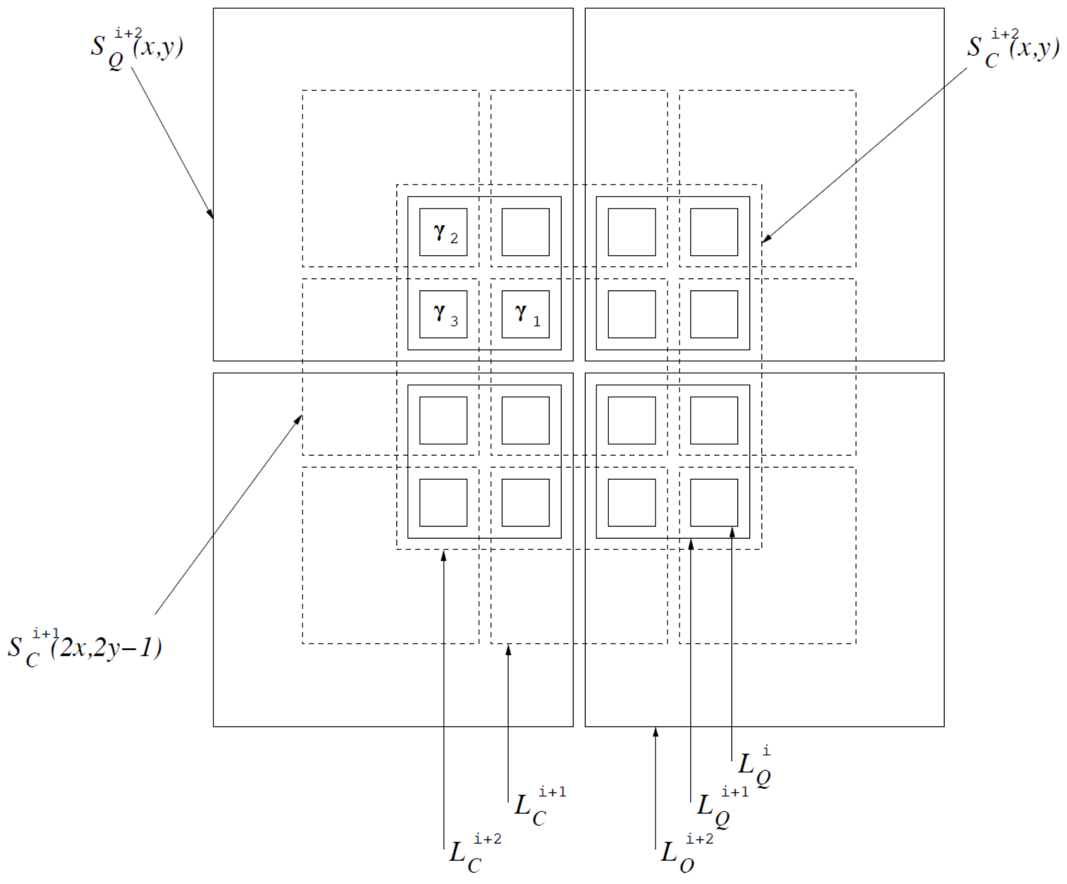

FIGURE 4.1: The symmetric structure of layers L_Q^i, L_C^{i+1}, L_Q^{i+1}, L_C^{i+2} and L_Q^{i+2}.

The cases depending which of these sixteen squares is S^*. Due to the symmetry of the five layers in $\mathcal{H}_{QC}$, without loss of generality, only one needs to be considered, e.g. the top-left quadrant of $S_C^{i+2}(x,y)$, which contains four squares at L_Q^i. One of the following three possible cases can occur.

Case 1 If S^* containing p_1 corresponds to the square depicted by γ_1, then p_2 must be located within $S_C^{i+2}(x,y)$, since $S_C^{i+2}(x,y)$ contains the entire d-neighbourhood of γ_1.

Case 2 If S^* corresponds to the square depicted by γ_2 then p_2 must be located within $S_Q^{i+2}(x,y)$.

Case 3 If S^* is one of the two remaining, symmetrically located squares within the selected quadrant depicted by γ_3, then p_2 is located either in $S_Q^{i+2}(x,y)$, $S_C^{i+2}(x,y)$ or in $S_C^{i+1}(2x,2y-1)$.

Thus in all three cases there exists a square within layers L_Q^i, L_C^{i+1}, L_Q^{i+1}, L_C^{i+2}, and L_Q^{i+2}, that contains both p_1 and p_2. Moreover, since squares at those layers are of size $\mathcal{O}(d)$, the thesis of the lemma follows. $\square$

In fact a stronger result holds too.

Corollary 4.7. *Take any fragment of $\mathcal{H}_{QC}$formed of three consecutive layers L_Q^i, L_Q^{i+1}, and L_Q^{i+2} in $\mathcal{H}_Q$ interleaved with two respective layers L_C^{i+1} and L_C^{i+2} in $\mathcal{H}_C$, such that, $d \le 2^i$. This fragment contains also a square that contains p_1 and p_2.*

Proof. The three cases from Lemma 4.6 also apply here. $\square$

Definition 4.8 (Space-covering sequences)**.** Recall, that at the lowest layer of the structure $\mathcal{H}_{QC}$ there is partition π_1 containing *unit squares* of L_Q^0. On the basis of unit squares, *atomic* space-covering sequences can be formed, constituting basic components of length $\mathcal{O}(1)$. The atomic sequence based on a point p belonging to a unit square in the 2D space is referred to as the source $s(p)$ of p. Suppose that $s(p)$ is the sequence formed of a single point, being the top left corner of the unit square containing p.

At any higher layer π_i recursively form a longer sequence associated with each square S from this layer by concatenating the sequences from layer π_{i-1}, corresponding to the children of S in the tree $\mathcal{T}_{QC}$ (i.e. for even i - squares from π_{i-1} that are covered by S^*, see Lemma 4.4, and, for odd i - squares from π_{i-1} that overlap with S^*, see Lemma 4.5).

To perform such construction *connectors* are used. Connectors link together the portions of the space-covering curve already created for the squares at the lower level. For the simplicity of presentation suppose that the portion of the space-covering curve corresponding to any square S starts and ends at the top left corner of S. Assume that the children of the same parent of $\mathcal{T}_{QC}$ are arranged in the clockwise order starting from the top left child. The connectors are the line segments joining the top left corners of the siblings. The connectors are used twice, once in the increasing order of the siblings (which, by convention, corresponds to the left-to-right traversal of the space-covering sequence) and the other time in the decreasing order. For example connectors A, B, C link, respectively, squares $S_Q^i(2x-1,2y+1)$, $S_Q^i(2x,2y+1)$, $S_Q^i(2x,2y)$ and $S_Q^i(2x-1,2y)$ in Figure 4.2(a) and squares $S_C^i(x,y)$, $S_C^i(x+1,y)$, $S_C^i(x+1,y-1)$ and $S_C^i(x,y-1)$ in Figure 4.2(b). Note that, in the former case, the obtained curve already starts and ends at the top left corner of the parent square $S_C^{i+1}(x,y)$. In the latter case, connector D is also added (cf. Figure 4.2(b)), taken twice, in order to make the constructed portion start and end at the top left corner of the parent square $S_Q^i(x,y)$.

This process can be iterated for as long as it is required. The space-covering sequence associated with the *rendezvous square* $R(p_1,p_2)$ will be used to obtain rendezvous of two participating agents located initially at points p_1 and p_2. In what follows, it will be shown how to explore the space-covering sequence efficiently during the rendezvous process.

Note that, since each layer in $\mathcal{H}_{QC}$ is a partition of the 2D space into squares, every point p in the 2D space can be associated with a unique infinite *list of squares* $S_1(p), S_2(p), \ldots$ from the consecutive layers $\pi_1(p), \pi_2(p), \ldots$ in $\mathcal{H}_{QC}$, respectively, such that, each square contains p. Note also, that the space-covering sequence associated with $S_i(p)$, for each $i \geq 1$ forms a contiguous segment of positions in the space-covering sequence associated with $S_{i+1}(p)$. Let $left(S)$ and $right(S)$ denote, respectively, the leftmost and the rightmost position on the space-covering sequence corresponding to square S. This then gives $left(S_{i+1}(p)) \leq left(S_i(p)) \leq right(S_i(p)) \leq right(S_{i+1}(p))$.

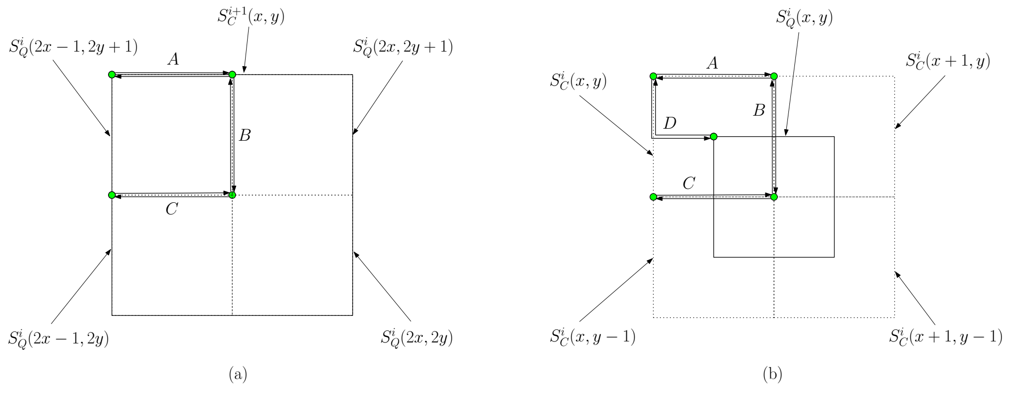

FIGURE 4.2: Connectors between siblings (dotted line squares) and parent (solid lines). In case (a) the parent comes from $\mathcal{H}_C$ family and in case (b) the parent comes from $\mathcal{H}_Q$

Lemma 4.9. *For any two points p_1 and p_2 at distance d in the 2D space, the space-covering sequence associated with the rendezvous square $R(p_1, p_2)$, is of length $\mathcal{O}(d^4)$.*

Proof. Recall that the lengths of space-covering sequences associated with squares in $\pi_1 = L_Q^0$ are $\mathcal{O}(1)$. Assume now that the sequences associated with squares of size 2^{k-1} at layer L_Q^{k-1} are of size $f(2^{k-1})$, for some positive integer function f. Note that the connectors from layer k, used for linking the endpoints of the space-covering sequences for the squares from layer $k-1$, are of length $\mathcal{O}(2^k)$. Thus the space-covering sequences associated with squares in L_C^k have length $4 \cdot f(2^{k-1}) + \mathcal{O}(2^k)$. Similarly, associate the squares in layer L_Q^k with space-covering sequences of length $4(4 \cdot f(2^{k-1}) + \mathcal{O}(2^k)) + \mathcal{O}(2^k) = 16 \cdot f(2^{k-1}) + \mathcal{O}(2^k)$. Thus the length of the space-covering sequences associated with the squares in L_Q^k (also in L_C^k) can be described by the recurrence:

$$f(2^k) = \begin{cases} \mathcal{O}(1) & \text{if } k = 0 \\ 16 \cdot f(2^{k-1}) + \mathcal{O}(2^k) & \text{if } k > 1 \end{cases}$$

Since, by Lemma 4.6, the rendezvous square $R(p_1, p_2)$ is of size $\mathcal{O}(d)$, the recurrence will be applied at most $\log d + \mathcal{O}(1)$ times. Thus the total length of the space-covering sequences for the squares from the layers L_Q^k and L_C^k is $\mathcal{O}(d^4)$. □

It is now shown that certain layers coming from $\mathcal{H}_C$ can be removed from $\mathcal{H}_{QC}$, obtaining a new hierarchy $\mathcal{H}^*_{QC}$, such that the distance separating p_1 and p_2 on the corresponding space-covering sequence can be reduced to $\mathcal{O}(d^{2+\varepsilon})$, for any constant $\varepsilon > 0$.

Definition 4.10 (Hierarchy $\mathcal{H}^*_{QC}$)**.** Fix a natural number z. $\mathcal{H}^*_{QC}$is formed of a sequence of blocks, each block containing $z+4$ layers. The i^{th} block, for $i = 0, 1, \ldots$, contains z consecutive layers of $\mathcal{H}_Q$ - $L_Q^{i(z+2)}$, $L_Q^{i(z+2)+1}, \ldots, L_Q^{i(z+2)+z-1}$, followed by four layers $L_C^{i(z+2)+z}$, $L_Q^{i(z+2)+z}$, $L_C^{i(z+2)+z+1}$, $L_Q^{i(z+2)+z+1}$. E.g. the portion of $\mathcal{H}^*_{QC}$corresponding to the first two blocks is

$$L_Q^0, L_Q^1, \ldots, L_Q^{z-1}, L_C^z, L_Q^z, L_C^{z+1}, L_Q^{z+1}, L_Q^{z+2}, \ldots, L_Q^{2z+1}, L_C^{2z+2}, L_Q^{2z+2}, L_C^{2z+3}, L_Q^{2z+3}$$

Note that, if some layer of $\mathcal{H}^*_{QC}$comes from $\mathcal{H}_C$, i.e. it is L_C^k, for some value of k, its predecessor in $\mathcal{H}^*_{QC}$is L_Q^{k-1}, hence Lemma 4.4 directly applies. On the other hand, for any layer of $\mathcal{H}^*_{QC}$coming from $\mathcal{H}_Q$, say L_Q^k, its predecessor in $\mathcal{H}^*_{QC}$is either L_C^k or L_Q^{k-1}. In the former case Lemma 4.5 directly applies. In the latter case each square from L_C^k partitions exactly into four squares of L_Q^{k-1}, hence the claim of Lemma 4.5 is true as well. Therefore the tree $\mathcal{T}^*_{QC}$ representing hierarchy $\mathcal{H}^*_{QC}$may be obtained and the space-covering sequences are constructed similarly as in the case of $\mathcal{H}_{QC}$, where in case of missing layers from $\mathcal{H}_C$ the concatenation process is performed according to the structure of squares located in the hierarchical partition $\mathcal{H}_Q$. Similarly, in $\mathcal{H}^*_{QC}$the rendezvous square $R(p_1, p_2)$ of two points p_1 and p_2 is the square on a lowest layer which contains the two points. The following lemma holds.

Lemma 4.11. *For any constant $\varepsilon > 0$, there exists $\mathcal{H}^*_{QC}$in which the space-covering sequence associated with the rendezvous square $R(p_1, p_2)$ of two arbitrary points p_1 and p_2 located at distance d in the* 2D *space is of length $\mathcal{O}(d^{2+\varepsilon})$.*

Proof. According to Corollary 4.7, $R(p_1, p_2)$ - the rendezvous square for p_1 and p_2 is present in $\mathcal{H}^*_{QC}$. Moreover, $R(p_1, p_2)$ belongs to layer k, such that $k = \log d + z + \mathcal{O}(1) = \log d + \mathcal{O}(1)$, for some constant z, meaning that the size of the rendezvous square is still $\mathcal{O}(d)$. The size of the space-covering sequence at layer k in $\mathcal{H}^*_{QC}$is then bounded by $\mathcal{O}(4^k \cdot 4^{2 \cdot \frac{k}{z+2}})$, where contribution $\mathcal{O}(4^k)$ comes from the structure of $\mathcal{H}_Q$ and $\mathcal{O}(4^{2 \cdot \frac{k}{z+2}})$ comes from $\mathcal{H}_C$. Note that the constant z can be chosen, such that, the exponent in the second term is the smallest constant that is required. Also since $k = \log d + z + \mathcal{O}(1)$ the second term translates to

$$\mathcal{O}(4^{2 \cdot \frac{\log d + z + \mathcal{O}(1)}{z+2}}) \leq \mathcal{O}(4^{2 \cdot \frac{\log d + z + \mathcal{O}(1)}{z}}) = \mathcal{O}(4^{\frac{2}{z} \cdot \log d}) = \mathcal{O}(d^{\frac{4}{z}}).$$

Thus for any $\varepsilon > 0$, an integer constant $z \geq \frac{4}{\varepsilon}$ can be found, such that, the length of the space-covering sequence in $\mathcal{H}^*_{QC}$for any two points at distance d in the 2D space is $\mathcal{O}(d^{2+\varepsilon})$. □

4.3 The Rendezvous Algorithm

This rendezvous algorithm utilises the nested structure of space-covering sequences associated with the list of squares defined for each point p in the 2D space. The agent determines the list of squares in $\mathcal{H}^*_{QC}$according to its initial location p. Then, for each square $S_i(p)$ from the list, the agent visits the leftmost and the rightmost point on the space-covering sequence, corresponding to the computed traversal of S_i, until it encounters the other agent.

Algorithm 4: Algorithm `RV` (point $p \in$ 2D space)

```
visit an integer grid point of the 2D space which the closest to $p$
$i \Leftarrow 1$
**repeat**
   $S_i(p) \Leftarrow$ square from layer $i$ of $\mathcal{H}^*_{QC}$containing $p$
   Go right on the space-covering curve until reaching $right(S_i(p))$
   Go left on the space-covering curve until reaching $left(S_i(p))$
   $i \Leftarrow i + 1$
**until** rendezvous is reached
```

Theorem 4.12. *Two location aware agents located in points p_1 and p_2 at distance d in the* 2D *space executing algorithm* `RV`*will meet after traversing asynchronously a trajectory of length $\mathcal{O}(d^{2+\varepsilon})$, for any constant $\varepsilon > 0$.*

Proof. By Corollary 4.7, there exists the square $R(p_1, p_2)$ in $\mathcal{H}^*_{QC}$containing p_1 and p_2. This square is on the list of squares considered in step 4 of the algorithm by each agent. Suppose, by symmetry, that agent α_1 is the first one to terminate at time t step 6 of the algorithm for the iteration i during which $S_i(p_1) = R(p_1, p_2)$. If agent α_2 has not yet completed its execution of step 1 of the algorithm, it must belong to $S_1(p_2)$ - a unit square included in $R(p_1, p_2)$ - and α_1 completing step 6 must meet α_2. Hence it can be assumed that, at time t agent α_2 must be traversing a portion of the space-covering sequence, corresponding to some square included in $R(p_1, p_2)$. All intermediate space-covering sequences associated with the predecessors of $R(p_1, p_2)$ in the lists of squares for p_2 form the space-covering sequences included in the interval $[left(R(p_1, p_2)), right(R(p_1, p_2))]$. Hence, while agent α_1 traverses this interval in step 6, it must meet agent α_2, which, by this assumption, is within this segment at time t. The total length of the trajectory adopted by the agents is linear in the size of the space-covering sequence due to exponential growth of intermediate sequences associated with the consecutive squares in the list of squares. □

Remark 4.13. Note that as there are $\Omega(d^2)$ integer grid points within distance d from any point in the 2D space. Therefore, since the adversary can keep one of the agents immobile, the rendezvous implies that the other agent has to explore its d-environment, adopting a route of length $\Omega(d^2)$. Thus the cost of this algorithm is almost optimal.

Chapter 5

Optimal Patrolling of Fragmented Boundaries

5.1 Introduction

This chapter introduces the results for performing network patrolling in a ring environment. This work was accepted to the 25^{th} *ACM Symposium on Parallelism in Algorithms and Architectures*, cf. [54].

A set of mobile robots is deployed on a simple curve of finite length, composed of a finite set of vital segments separated by neutral segments. The robots have to patrol the vital segments by perpetually moving on the curve, without exceeding their uniform maximum speeds. The quality of patrolling is measured by the idleness, i.e. the longest time period during which any vital point on the curve is not visited by any robot. Given a configuration of vital segments, our goal is to provide algorithms describing the movement of the robots along the curve so as to minimize the idleness.

Our main contribution is a proof that the optimal solution to the patrolling problem is attained either by the cyclic strategy, in which all the robots move in one direction around the curve, or by the partition strategy, in which the curve is partitioned into sections which are patrolled separately by individual robots. These two fundamental types of strategies were studied in the past in the robotics community in different theoretical and experimental settings. However, to our knowledge, this is the first theoretical analysis proving optimality in such a general scenario. Throughout the paper we assume that all robots have the same maximum speed. In fact, the claim is known to be invalid when this assumption does not hold.

Protecting an environment by a set of stationary or mobile point-guards has been studied before in various scenarios. The problem of patrolling a one-dimensional boundary using mobile robots has many real-world applications, and is extensively studied under the names of boundary patrolling and fence patrolling in the robotics literature [62]. In order to prevent an intruder from penetrating into a protected region, the boundary of the region must be patrolled. Some parts of the boundary may be monitored with stationary devices like sensors or cameras (or they do not need to be monitored at all), while other portions require the aid of moving robots such as walking guards, illumination rays, mobile robotic devices, etc. Since the feasibility of an intrusion likely depends on the time during which the intruder remains undiscovered, it is important to design patrolling protocols which minimize the time during which boundary points are unprotected.

Some portions of the boundary may be impenetrable at all, or they may be monitored with stationary devices like sensors or cameras. This work is devoted to the scenario in which only a finite number n of boundary segments, referred to as vital regions, need to be patrolled by a set of k mobile agents. The remaining part of the boundary, called neutral regions, do not have to be monitored by the mobile agents, but may nevertheless be traversed by an agent since this may be the way to reach one vital region from another. The problem studied is that of patrolling with the goal of minimising the idleness of points located in the vital regions, i.e. the longest time during which such a point remains unvisited by an agent. It is assumed that at any time during the traversal the speed of each agent cannot exceed a certain maximum value, identical for all agents. The goal is to define a set of functions describing the trajectories of all of the agents in time.

The most common heuristics adopted in the past to solve a variety of patrolling problems include the cyclic strategy, where agents move in one direction around the cycle covering the environment, and the partition strategy, in which the environment is partitioned into sections patrolled separately by individual agents (or subsets of agents), using the terminology introduced in [43]. However, to the best of the authors knowledge, no theoretical studies formally proving the optimality of such approaches in this setting were done in the past.

It is worth noting, that in the more heterogeneous scenario where robots have different maximum speeds, neither the cyclic strategy nor the partition strategy leads to the optimal performance. In fact, it has been shown in [62] that for the case of 3 mobile agents with different maximal speeds patrolling a cycle (forming a single vital region), neither a partition strategy nor a cyclic strategy is optimal. It turns out that a specific hybrid strategy is better than each of these two fundamental approaches. See also [109].

5.1.1 Model, Preliminaries, and Notation

$k \geq 1$ agents are considered. Each agent moves along a continuous rectifiable curve C, i.e. a curve of finite length.

Definition 5.1 (Traversal strategy)**.** A traversal strategy for a single agent is a continuous function $f : [0, +\infty) \to C$ such that $t \mapsto f(t)$, whereby $f(t)$ is the position of the agent on the rectifiable curve at time $t \geq 0$. A traversal strategy for k agents consists of k such continuous functions, one f_i for each agent $1 \leq i \leq k$.

Without loss of generality, assume that the curve C is either a segment of unit length (when C is an open curve), or a cycle of unit perimeter (when C is a closed curve). In both cases, this work will adopt the unit segment $[0, 1]$ to represent uniquely all points on the curve, noting that in the case of the cycle, points 0 and 1 are identical with each other. All agents move along C at speeds not exceeding the unit maximum value understood to be 1.

Definition 5.2 (Unit maximum speed)**.** Let $dist(p, q)$ denote the distance between any points $p, q \in C$ along the curve C. The traversal strategy for k agents respects the agents' unit maximum speed if for any $1 \leq i \leq k$ and any $t_1, t_2 \in [0, +\infty)$ then the following will hold $\frac{dist(f_i(t_1), f_i(t_2))}{|t_1 - t_2|} \leq 1$.

In what follows, this work will always consider traversal strategies that respect the maximum speed of each agent. The task of the k agents is to patrol so-called vital regions, located in the unit-length curve C (along which the agents move), so as to minimise the idleness of the points in the vital regions.

Definition 5.3 (Vital and neutral regions)**.** Each considered curve C contains n disjoint *vital regions* represented by closed intervals $\Lambda_1, \Lambda_2, \ldots, \Lambda_n$, where $\Lambda_i = [b_i, e_i]$ and $e_i < b_{i+1}$ for $1 \leq i \leq n - 1$, with $b_1 = 0$ and $e_n \leq 1$. The vital regions are separated by open intervals from N called *neutral regions*. In other word the curve $C = (\Lambda, N)$, where $[0, 1] \supseteq \Lambda = \bigcup_{i=1}^{n} \Lambda_i$, and $N = [0, 1] \setminus \Lambda$.

Definition 5.4 (Idleness)**.** Let $\mathcal{A}$ be a traversal strategy consisting of k continuous functions f_i for a system of k agents, respectively, and traversing the given curve.

1. The idleness induced by $\mathcal{A}$ at a point x of the curve, denoted by $I_x(\mathcal{A})$, is the supremum of the length of time intervals during which point x remains unvisited by any agent:

$$I_x(\mathcal{A}) = \sup_{\{0 \leq T_1 < T_2 \colon \forall i \forall t \in [T_1, T_2] \ f_i(t) \neq x\}} (T_2 - T_1).$$

2. The idleness of the system of k agents induced by $\mathcal{A}$ is defined by the supremum taken over all vital points of the curve:

$$I(\mathcal{A}) = \sup_{x \in \Lambda} I_x(\mathcal{A})$$

3. Finally, the idleness of the system of k agents is defined by $I_{opt} = \inf_{\mathcal{A}} I(\mathcal{A})$, the infimum taken over all traversal strategies $\mathcal{A}$.

With these definitions in mind the main question that will be addressed in this work can be formulated.

Question. Suppose that there are k agents traversing a given rectifiable open curve (resp., closed curve) $C = (\Lambda, N)$, represented without loss of generality as the unit-length segment (resp., the unit-perimeter cycle). What traversal strategy should the agents follow so as to minimise the idleness of the system?

Observe that this assumes a very general setting for the schedules of the agents: these are arbitrary continuous functions of the time parameter. In all cases it will be necessary to prove the optimality of the proposed schedules by showing tight upper and lower bounds on the idleness thus attained. Although, in most instances the upper bound will consist of simple algorithms describing how the agents should traverse the given curve, the lower bounds will be more challenging by proving that they are valid even for the most general settings of the class of continuous functions.

The main challenge in this study is the requirement that an algorithm with optimal idleness is being sought, for traversing all the vital intervals in the curve while at the same time the agents do not necessarily have to traverse neutral intervals.

5.1.2 Outline and Results of the Work

Lets start by recalling the *partition strategy* of patrolling in §5.2, in which each agent traverses some sub-interval of the curve back and forth. This strategy is always optimal on open curves (i.e. on the unit segment), but it need not be optimal for closed curves (i.e. for the unit cycle).

The main results concern closed curves, and are given in §5.3. This work proves that the optimal idleness for patrolling the boundary is always attained by the better of two strategies: the before-mentioned partition strategy, and the *cyclic strategy*, in which equally-spaced agents patrol the cycle, moving in the same direction. The choice of the strategy and the agents' responsibilities depends on the arrangement of the vital regions around the boundary. This approach consists in showing that finding the optimal idleness for k agents and for any set of n vital intervals may be reduced to finding the idleness for some *critical* set of $2k+1$ vital points (always resulting in the cyclic strategy) or of a critical set of $k+1$ vital points (resulting in either the cyclic or the partition strategy).

Finally, in §5.4, an $\mathcal{O}(kn \log n)$ algorithm is proposed for designing traversal strategies with optimal idleness for agents on both open and closed curves.

5.2 Optimal Patrolling Strategy for the Segment

Firstly, patrolling with k agents, is shown, of a terrain modelled by a curve $C = (\Lambda, N)$ consisting of n vital regions in $\Lambda \subseteq [0,1]$, recall Definition 5.3.

In order to describe the region patrolled by a single agent in the partition strategies this work will frequently refer to the concept of a lid which will be employed, in particular, to define a patrolling strategy.

Definition 5.5 (**(*d*,*k*)**-Lid cover)**.** A *d-lid* is a contiguous interval on the curve of length d. Curve $C = (\Lambda, N)$ has a (d,k)-*lid cover* if all of its vital regions can be covered by some set of k (not necessarily disjoint) d-lids.

A natural approach to patrolling the segment is based on the partition strategy, in which each of the agents patrols exactly one of the k lids of the lid cover of minimum lid size L. The agent moves back and forth between its endpoints at maximum speed.

```
Algorithm 5: Partition strategy (on the segment)
    Compute a $(L, k)$-lid cover of $C$, where $L$ is chosen as the minimum lid length for which $C$ admits such a lid cover.
    Let the $i^{th}$ lid, $1 \leq i \leq k$, be a segment of the form $[c_i, c_i + L]$.
    Deploy the $i^{th}$ agent so that at time $t = 2Lj + \tau$, where $j$ is a non-negative integer and $-L \leq \tau < L$, the position $f_i(t)$ of this agent on the lid is $f_i(t) = c_i + |\tau|$.
```

Note, that in this section it is Algorithm 5 that will be discussed, the algorithm that is used for computing the (L, k)-lid cover of C will be shown later in §5.4.1.

Observe that each of the points of every lid, and consequently every vital point of the segment, is visited at least once during each time interval of size $2L$. So, for this strategy there is an idleness of $I \leq 2L$. The idleness of the partition strategy is, in fact, optimal on the segment.

Theorem 5.6 (***k*** agents)**.** *The optimal idleness for k agents moving at speed at most 1 on a unit segment is given by $I = 2L$, where L is the minimum value such that the terrain admits a (L, k)-lid cover.*

To prove the theorem, the following property is shown of a greedy cover of the segment with lids.

Lemma 5.7. *Suppose that L is the minimum value, such that, a given terrain C admits a (L, k)-lid cover, and let $L' < L$. Then, $C \setminus [0, L']$ does not admit a $(L', k-1)$-lid cover.*

Proof. Indeed, if the region $C \setminus [0, L']$ admitted a $(L', k-1)$-lid cover, then one could obtain a (L', k)-lid cover of C simply by adding lid $[0, L']$ to the lid cover obtained for the region $C \setminus [0, L']$. This contradicts the minimality of L. □

Lemma 5.8. *Any patrolling strategy may be converted to a strategy, achieving the same idleness, for which the relative order of the agents on the segment is maintained throughout the traversal.*

Proof. The proof is based on the simple observation that when two agents meet while moving in opposite directions they can "exchange" roles, so that the coverage of the points on the segment by one agent is the same as coverage by the other. Since after this change of roles the set of visited nodes at any time remains the same, this does not affect the idleness of the visited nodes. □

The proof of the claim $I \geq 2L$ now proceeds by induction on the number of agents. It is clearly true for $k = 1$, since the idleness of the strategy cannot be smaller than twice the distance between the extremal vital points C, which corresponds precisely to the size of the smallest lid cover.

Suppose, by contradiction, that there exists a value of k and some terrain C such that the idleness of some patrolling strategy $\mathcal{A}$ is $I = 2L' < 2L$. By Lemma 5.8, without loss of generality it can be assumed that the agents never change places along the segment. Consider the trajectory of the leftmost agent α_1 following strategy $\mathcal{A}$. Let c be the supremum of all points along the segment reached by agent α_1. If point c is reached by agent α_1 at some time t, its last visit to point 0 must have been no later than at time $t - c$, and the next visit to point 0 will take place at time not earlier than $t + c$. From Lemma 5.8, it occurs that point 0 is never visited by any agent when it is not visited by α_1. Consequently, it must exist that $2c \leq 2L'$, and so $c \leq L'$. It follows that the region $C \setminus [0, L']$ must be patrolled solely by the set of $k-1$ agents, without the help of α_1, with idleness at most $2L'$. From the inductive assumption, this gives $C \setminus [0, L']$ admits a L'-lid cover. This is a contradiction, by Lemma 5.7. This completes the proof of Theorem 5.6.

The complexity of computing the optimal lid cover for the partition strategy is discussed in detail in §5.4.

5.3 Optimal Patrolling Strategy for the Cycle

In this section computing the optimal idleness for k agents traversing terrains represented as a unit-perimeter cycle $C = (\Lambda, N)$, is shown, with vital and neutral regions in Λ and N respectively. The class of strategies under consideration in a cycle is larger than in the case of a segment due to the ability of the agents to traverse the perimeter of the cycle. In particular, the agents on the cycle can also apply a *cyclic strategy*, performing *clockwise* (direction aligned with increasing indices of intervals in $\Lambda \subseteq [0, 1]$) rotations around the cycle with even time spacing.

Algorithm 6: Cyclic strategy (on the cycle)

```
    Deploy the $i^{th}$ agent at time 0 at position $i/k$ along the circumference of the cycle.
    Release all agents at their maximum speed to perform a clockwise traversal of the
    cycle.
```

Observation 5.9. The idleness of the cyclic strategy on the cycle is $I = 1/k$, for any (non-empty) set of vital regions.

At the same time, observe that the partition strategy introduced in the previous section is also applicable in the cycle, achieving an idleness of $I = 2L$, where L is the size of the minimum lid cover of the vital regions of the cycle with k lids. Depending on the configuration of the vital regions, one or the other of these two strategies may prove superior. In one extremal case when the cycle has no neutral regions, the cyclic strategy achieves an idleness of $1/k$, while the partition strategy has an idleness of $2/k$. At the other extreme, for vital regions consisting of k discrete points, the idleness of the cyclic strategy is still $1/k$, but the partition strategy has an idleness of 0. This leads naturally to a strategy which selects the better of the two approaches.

Algorithm 7: Combined strategy (on the cycle)

```
Let $L$ be the lid size of the minimum (with respect to lid size) lid cover of the vital
regions of the cycle with $k$ lids.
if $2L < 1/k$ then
  Apply Partition strategy.
else
  Apply Cyclic strategy.
end if
```

Observation 5.10. The idleness of the combined strategy on the cycle is $I = \min\{1/k, 2L\}$, where L is the minimum possible lid size of a (L, k)-lid cover of the cycle.

This claim gives rise to the following natural question. Does there exist any other strategy which can achieve better idleness than both the partition and cyclic approaches? Such a question admits a positive answer for the cycle in the scenario where agents have different speeds [62], even when neutral regions are not present. In this scenario, with neutral regions but for agents with equal maximal speeds, the combined strategy turns out to be optimal. The proof of this fact is surprisingly involved.

Theorem 5.11. *The idleness $I(\mathcal{A})$ of any traversal strategy $\mathcal{A}$ in a cycle with neutral regions satisfies $I \geq \min\{1/k, 2L\}$, where L is the minimum possible lid size of a (L, k)-lid cover of the cycle.*

The rest of this section is devoted to the proof of Theorem 5.11, which proceeds in three technical lemmas. First, it is shown that for any cycle with neutral regions it can be found that their are a subset of either exactly $k+1$ or exactly $2k+1$ (discrete) vital points that satisfy specific properties. Then, it is shown that the lower bound can be proved simply by considering the patrolling problem on the selected subset of points.

Lemma 5.12 (Critical point). *Let $C = (\Lambda, N)$ be a cycle with set of vital regions Λ. Let L be the minimum size of the lid cover of the vital regions of C with k lids, and let $B = \sup\{dist(b, e) : b, e \in [0, 1], [b, e] \subseteq N\}$. Then:*

(1) If $B \geq 1/(2k)$, then there exists a set of $k+1$ vital points $\{\lambda_0, \ldots, \lambda_k\} \subseteq \Lambda$, ordered clockwise, such that $\min_{0 \leq i \leq k} dist(\lambda_i, \lambda_{(i+1) \bmod (k+1)}) \geq \min\{1/(2k), L\}$.

(2) If $B < 1/(2k)$, then there exists a set of $2k+1$ vital points $\{\lambda_0, \ldots, \lambda_{2k}\} \subseteq \Lambda$, ordered clockwise, such that $\min_{0 \leq i \leq 2k} dist(\lambda_i, \lambda_{(i+2) \bmod (2k+1)}) > 1/(2k)$.

Proof. To prove clause (1), let λ_0 be the first vital point located at the clockwise endpoint of a neutral region of length B. Fix $\epsilon > 0$ and consider the set of points chosen iteratively as follows: let λ_{i+1} be the first vital point located at arc distance not less than $L - \epsilon$ from λ_i, moving in the clockwise direction. Point λ_k is reached before completing one full rotation around the cycle, starting from λ_0. Indeed, if this were not the case, then there would exist a set of k lids: $[\lambda_0, \lambda_0 + L - \epsilon], \ldots, [\lambda_{k-1}, \lambda_{k-1} + L - \epsilon]$, covering the whole of Λ, a contradiction with the minimality of lid cover size L. Finally, note that the distance between points λ_0 and λ_k is at least $B \geq 1/(2k)$. Parametrising each of the points λ_i as $\lambda_i(\epsilon)$, it follows that for any $\epsilon > 0$, their can be found a set of $k+1$ points $(\lambda_0(\epsilon), \lambda_1(\epsilon), \dots, \lambda_k(\epsilon))$ such that $\min_{0 \leq i \leq k} dist(\lambda_i(\epsilon), \lambda_{(i+1) \bmod (k+1)}(\epsilon)) \geq \min\{1/(2k), L\} - \epsilon$. By taking into account that the set of vital points Λ is a closed set and $\lambda_i(\epsilon)$ is non-decreasing and bounded (with regards to shifts in the clockwise direction) for any sequence $\epsilon \searrow 0$, converge to a sequence of vital points $(\lambda_0(0), \dots, \lambda_k(0))$ satisfying clause (1).

To prove clause (2), a slightly stronger version "(2')" of this clause in which the assumption "$B < 1/(2k)$" is replaced by "$B \leq 1/(2k)$". Suppose that terrain (Λ, N) is a counterexample to the claim of (2'), such that for any other terrain (Λ', N') which violates clause (2') it holds that $\Lambda' \subsetneq \Lambda$. (Such an inclusion-wise minimal counterexample always exists, since the set of vital points is by assumption a closed set.) By the minimality of Λ, the set of its vital points must be discrete, say, $\Lambda = \{u_0, \dots, u_{n-1}\}$.

Assume that for some $0 \leq i \leq n$, $dist(u_i, u_{(i+2) \bmod (n)}) \leq 1/(2k)$. The terrain $(\Lambda \setminus \{u_{i+1}\}, N \cup \{u_{i+1}\})$ has no neutral intervals of length greater than $1/(2k)$, and thus is a smaller counter-example to this claim, a contradiction. It will now follow that $\min_{0 \leq i < n} dist(u_i, u_{(i+2) \bmod (n)}) > 1/(2k)$. Further to this, since for all $0 \leq i < n$, $dist(u_i, u_{(i+1) \bmod (n)}) < 1/(2k)$, there must be $n \geq 2k+1$. So, choosing points $\{\lambda_0, \dots, \lambda_{2k}\}$ as $\lambda_i = u_i$, for all $0 \leq i \leq 2k$, it can be found in Λ the subset of vital points satisfying clause (2'). So, Λ cannot be a counter-example to the claim. □

Lemma 5.13 (**$k+1$** points)**.** *Let $(\lambda_0, \lambda_1, \dots, \lambda_k)$ be a set of $k+1$ points chosen from vital regions of the terrain, arranged in the clockwise order. The idleness $I(\mathcal{A})$ of any traversal strategy $\mathcal{A}$ for k agents in this terrain satisfies $I(\mathcal{A}) \geq \min\left\{\frac{1}{k}, 2s\right\}$, where: $s = \min_{0 \leq i \neq j \leq k} dist(\lambda_i, \lambda_j)$.*

Proof. Let $(\lambda_0, \lambda_1, \dots, \lambda_k)$ be $k+1$ vital points chosen so that $dist(\lambda_i, \lambda_{(i+1)}) \geq s$, for all $0 \leq i \leq k$. Throughout the proof, indices of points and agents are understood modulo $k+1$, i.e. for all integers i, j assume $\lambda_i \equiv \lambda_{i \bmod (k+1)}$ and $\alpha_j \equiv \alpha_{j \bmod (k+1)}$. It can be shown that the claim holds even if $\{\lambda_0, \lambda_1, \dots, \lambda_k\}$ are the only vital points of the cycle.

If the idle time of any strategy is at least equal to $2s$, the claim holds. Now, consider any (sufficiently small) $\epsilon > 0$ such that there exists a strategy $\mathcal{A}$ with $I(\mathcal{A}) < 2s - \frac{\epsilon}{2}$. There exists a point λ_i, $0 \leq i \leq k$, such that the time between some two consecutive visits of an agent to point λ_i is greater than $\tau = \frac{1}{k} - \epsilon$ when following strategy $\mathcal{A}$. (This completes the proof of the Lemma, since there is for any $\epsilon > 0$, $I(\mathcal{A}) < 2s - \frac{\epsilon}{2} \longrightarrow I(\mathcal{A}) > \frac{1}{k} - \epsilon$.)

Without loss of generality, by modifying the trajectories of the agents, a strategy $\mathcal{A}$ can be converted into another strategy $\mathcal{A}'$ so that the following properties are satisfied by $\mathcal{A}'$:

(i) if an agent following $\mathcal{A}'$ leaves some vital point λ_i, then it does not re-enter this vital point before reaching some other vital point first (namely, λ_{i-1} or λ_{i+1}),

(ii) no two agents following $\mathcal{A}'$ ever meet,

(iii) if a vital point is visited by an agent following strategy $\mathcal{A}$ at time t, then it is visited by an agent following strategy $\mathcal{A}'$ within the interval $[t - \frac{\epsilon}{4}, t + \frac{\epsilon}{4}]$.

For completeness, an outline the technical steps which are required to perform the above conversion follows. First, property (i) is achieved by modifying the trajectories of the agents in neutral regions, only. Next, properties (ii) and (iii) can be ensured by first converting the strategy to one which preserves the ordering of the agents as in Lemma 5.7, and then delaying the movements of some of the agents to avoid meetings, without changing the time intervals during which a vital point is occupied by more than $\frac{\epsilon}{4}$.

By property (iii), if a point is unvisited by $\mathcal{A}'$ in time interval $[t_1, t_2]$, then it is unvisited by $\mathcal{A}$ in the time interval $[t_1 + \frac{\epsilon}{4}, t_2 - \frac{\epsilon}{4}]$. It now suffices to show that the time between some two consecutive visits of an agent following strategy $\mathcal{A}'$ to point λ_i is greater than $\tau + \frac{\epsilon}{2} = \frac{1}{k} - \frac{\epsilon}{2}$. Moreover, $I(\mathcal{A}') \leq I(A) + \frac{\epsilon}{2} < 2s$. From now on lets consider agents following $\mathcal{A}'$, only.

Since no two agents following $\mathcal{A}'$ ever meet by (ii), an arbitrarily chosen agent can be denoted by α_1, and the other agents by $\alpha_2, \ldots, \alpha_k$ in clockwise order; this order never changes throughout the traversal.

Suppose that at some time t, an agent α_j leaves point λ_i on the arc towards point λ_{i+1}. By (i), the next vital point it reaches has to be point λ_{i+1}. Therefore, agent α_j cannot re-enter point λ_i before time $t + 2dist(\lambda_i, \lambda_{i+1}) \geq t + 2s > t + I(\mathcal{A}')$. So, some other agent must visit point λ_i in between the two visits by agent α_j. Since the agents never meet, it follows that within the time interval $[t, t+I(\mathcal{A}')]$, agent α_{j-1} entered node λ_i. Before this visit, the previous vital point visited by α_{j-1} must have been λ_{i-1}. It follows that to each traversal of the arc $(\lambda_i, \lambda_{i+1})$ by agent α_j that starts at some time t, a distinct traversal can be assigned of the arc $(\lambda_{i-1}, \lambda_i)$ by agent α_{j-1} that ends within the time interval $[t, t+I(\mathcal{A}')]$. Fix two values of time T_1 and T_2, $0 \leq T_1 < T_2$. From now on, this work will apply certain counting arguments within the time interval $[T_1, T_2]$. Let us denote by $C_j(i, i+1)$ the number of traversals of arc $(\lambda_i, \lambda_{i+1})$ by agent α_j starting in the time interval $[T_1, T_2]$. Since only the first and last traversals of $(\lambda_i, \lambda_{i+1})$ by agent α_j within this time interval may be unmatched by corresponding traversals of $(\lambda_{i-1}, \lambda_i)$ by agent α_{j-1} within the same time interval, thus providing:

$$C_j(i, i+1) - C_{j-1}(i-1, i) \leq 2.$$

Let $C(i, i+1) = \sum_{j=1}^{k} C_j(i, i+1)$ be the total number of traversals of the arc $(\lambda_i, \lambda_{i+1})$ by all agents starting within the time interval $[T_1, T_2]$. Summing the above inequalities, gives:

$$C(i, i+1) - C(i-1, i) \leq 2k.$$

An analogous analysis can be performed for the counter-clockwise direction, i.e. considering values of the form $C(i+1,i)$, corresponding to traversal of the arc from λ_{i+1} to λ_i. This obtains:

$$C(i,i-1) - C(i+1,i) \leq 2k.$$

In general, by iterating the above around the cycle, for any two vital points λ_{i_1} and λ_{i_2} this obtains:

$$C(i_1,i_1+1) - C(i_2,i_2+1) \leq 2k^2.$$

Denoting by $C^{cw} = \min_{0\leq i\leq k} C(i,i+1)$, provides for any i:

$$C(i,i+1) \leq C^{cw} + 2k^2.$$

An analogous analysis can be performed for the counter-clockwise direction, i.e. considering values of the form $C(i+1,i)$, corresponding to traversal of the arc from λ_{i+1} to λ_i. Consequently, denoting $C^{cc} = \min_{0\leq i\leq k} C(i+1,i)$, there is for any i:

$$C(i+1,i) \leq C^{cc} + 2k^2.$$

Now, denote by $W_j(i) \geq 0$ the total time spent by agent α_j at point λ_i within the time interval $[T_1,T_2]$, and let $W(i) = \sum_{j=1}^{k} W_j(i)$. Without loss of generality, let λ_0 be a vital point with the minimal total waiting time, i.e. $W(0) = \min_{0\leq i\leq k} W(i)$.

With respect to point λ_0, the trajectory of each agent α_j within the time interval $[T_1,T_2]$ can be described by an ordered sequence of time moments $(e_j^1, l_j^1, e_j^2, l_j^2, \ldots, l_j^{n_j})$, where e_j^p is the time at which agent α_j enters point λ_0 for the p^{th} time, whereas l_j^p is the time at which agent α_j leaves point λ_0 for the p^{th} time. It can be assumed that $T_1 \leq e_j^1 \leq l_j^1 < e_j^2 \leq l_j^2 < \ldots < e_j^{n_j} \leq l_j^{n_j} \leq T_2$, where $e_j^1 = T_1$ is put if agent α_j was located at node λ_0 at time T_1, and $l_j^{n_j} = T_2$ if agent α_j was located at node λ_0 at time T_2. For the sake of notation, let $l_j^0 = T_1$ and $e_j^{n_j+1} = T_2$.

During the time interval $[T_1,T_2]$, point λ_0 is covered by an agent during the set of moments X given as:

$$X = \bigcup_{j=1}^{k} \bigcup_{p=1}^{n_j} [e_j^p, l_j^p],$$

such that $|X| = W(0)$. (Here, $|X|$ denotes the measure of X, i.e. the sum of lengths of time intervals contained in X. Note that all of the time intervals $[e_j^p, l_j^p]$ in the above union are disjoint, since no two agents following strategy $\mathcal{A}'$ ever meet by the definition of the strategy.) During the remaining time, i.e. $\overline{X} = [T_1,T_2] \setminus X$, no agent is located at λ_0. Observe that $\overline{X}$ is a union of at most $1 + \sum_{j=1}^{k} n_j$ intervals. Hence, there exists some time interval of length τ:

$$\tau \geq \frac{|\overline{X}|}{1+\sum_{j=1}^{k} n_j} = \frac{(T_2-T_1)-|X|}{1+\sum_{j=1}^{k} n_j} = \frac{(T_2-T_1)-W(0)}{1+\sum_{j=1}^{k} n_j} \tag{5.1}$$

during which λ_0 remains unvisited.

Notice that each agent α_j leaves point λ_0 at least $n_j - 1$ times in the time interval $[T_1, T_2]$, going towards either point λ_1 or point λ_k. Thus, their is:

$$C(0,1) + C(0,k) \geq \sum_{j=1}^{k}(n_j - 1) = \sum_{j=1}^{k} n_j - k.$$

Taking into account that $C(0,1) \leq C^{cw} + 2k^2$ and $C(0,k) \leq C^{cc} + 2k^2$, provides:

$$\sum_{j=1}^{k} n_j \leq C^{cw} + C^{cc} + 4k^2 + k. \tag{5.2}$$

Moreover, since each arc of the cycle is traversed in either direction a total of at least $C^{cw} + C^{cc}$ times, the total distance covered by all the agents is at least $C^{cw} + C^{cc}$. Thus, the total time of the movement for all k agents within the time interval $[T_1, T_2]$ is at least $C^{cw} + C^{cc}$, and the following inequality is obtained:

$$C^{cw} + C^{cc} + \sum_{i=0}^{k} W(i) \leq k(T_2 - T_1)$$

$$C^{cw} + C^{cc} \leq k(T_2 - T_1) - (k+1)W(0). \tag{5.3}$$

Combining inequalities (5.1), (5.2), and (5.3), results in:

$$\tau \geq \frac{(T_2-T_1)-W(0)}{k(T_2-T_1)-(k+1)W(0)+4k^2+k+1} \geq$$

$$\geq \frac{1}{k + \frac{4k^2+k+1-W(0)}{(T_2-T_1)-W(0)}} \geq \frac{1}{k + \frac{4k^2+k+1}{(T_2-T_1)-(4k^2+k+1)}} >$$

$$> \frac{1}{k} - \frac{4k^2+k+1}{(T_2-T_1)-(4k^2+k+1)}.$$

In the above, is is assumed that $(T_2 - T_1) - W(0) > 0$, i.e. there cannot be an agent covering λ_0 throughout the time interval $[T_1, T_2]$. This is true, since otherwise, taking into account that $W(i) \geq W(0)$ for all $1 \leq i \leq k$, all $k+1$ points would have to be covered by an agent throughout $[T_1, T_2]$, and there are only k agents, a contradiction.

Now, suppose that T_2 is chosen to be sufficiently large so that $\frac{4k^2+k+1}{(T_2-T_1)-(4k^2+k+1)} < \frac{\epsilon}{2}$. This gives $\tau \geq \frac{1}{k} - \frac{\epsilon}{2}$, and so there exists a vital point on the cycle such that the time between some two successive visits of agents following $\mathcal{A}'$ to this point is greater than $\frac{1}{k} - \frac{\epsilon}{2}$. This completes the proof of Lemma 5.13. □

Lemma 5.14 ($\boldsymbol{2k+1}$ points). *Let $(\lambda_0, \lambda_1, \ldots, \lambda_{2k})$ be a set of $2k+1$ points chosen from the vital regions of the terrain, which are listed in the clockwise order, such that, $dist(\lambda_i, \lambda_{(i+2) \bmod (2k+1)}) > \frac{1}{2k}$. The idleness $I(\mathcal{A})$ of any traversal strategy $\mathcal{A}$ for k agents in this terrain satisfies $I(\mathcal{A}) \geq \frac{1}{k}$.*

Proof. Let $(\lambda_0, \lambda_1, \ldots, \lambda_{2k})$ be $2k+1$ vital points chosen in accordance with the assumptions of the lemma. For the proof of the lower bound, the concept is introduced of a *shadow agent*, which can be seen as an auxiliary agent which temporarily appears in the system and assists agents in their patrolling task. More precisely, given a strategy $\mathcal{A}$, consider the trajectory of an agent α_j. Suppose that the agent leaves a vital point λ_i at some time t_a, moves to an adjacent vital point $\lambda_{i_1} \in \{\lambda_{i-1}, \lambda_{i+1}\}$ and then returns to point λ_i at time t_b, without encountering any other vital points within the interval $[t_a, t_b]$. A shadow agent α_j^{i*} is created at time t_a at point λ_i, waits at λ_i protecting it until time t_b, and then disappears. The addition of such a shadow agent, obviously, cannot increase the idleness of the strategy.

Observe that one agent can create at most two shadow agents at a time: when α_j is located anywhere within a closed arc $[\lambda_i, \lambda_{i+1}]$, then it may only have the shadow agents α_j^{i*} and $\alpha_j^{(i+1)*}$. Agent α_j and its shadow agents can wait at not more than two vital points simultaneously.

Will show that the claim holds even if $\{\lambda_0, \lambda_1, \ldots, \lambda_{2k}\}$ are the only vital points of the cycle. The rest of the proof proceeds analogously to the proof of Lemma 5.13, subject to the inclusion of shadow agents in the team of agents patrolling the terrain. Once again, for a fixed $\epsilon > 0$, modify the trajectories of the agents, converting any strategy $\mathcal{A}$ into another strategy $\mathcal{A}'$ fulfilling the following properties:

(i) if an agent following $\mathcal{A}'$ leaves some vital point λ_i, then it does not re-enter this vital point before reaching some other vital point first (namely, λ_{i-1} or λ_{i+1}),

(ii) no two agents following $\mathcal{A}'$ ever meet each other or the shadow agents of other agents,

(iii) if a point P is visited by an agent following strategy $\mathcal{A}$ at time t, then it is visited by an agent or shadow agent following strategy $\mathcal{A}'$ within the interval $[t - \frac{\epsilon}{4}, t + \frac{\epsilon}{4}]$.

From now on consider agents $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ and their shadow agents following $\mathcal{A}'$, only, and proceed to perform a modification of the proof of Lemma 5.13 which takes shadow agents into account.

Suppose that $I(\mathcal{A}') < 1/k - \epsilon$. A traversal of the directed arc $(\lambda_i, \lambda_{i+1})$ by agent α_j will be referred to as *shadowless* if after arriving at λ_{i+1}, the next vital point visited by agent α_j is λ_{i+2} (not λ_i). Equivalently, a traversal of arc $(\lambda_i, \lambda_{i+1})$ is shadowless if α_j does not leave its shadow agent α_j^{i*} at λ_i during this traversal.

Fix a time interval $[T_1, T_2]$. For all $0 \leq i \leq 2k$, denote by $C_j(i, i+2)$ the number of shadowless traversals of the directed arc $(\lambda_i, \lambda_{i+1})$ by agent α_j starting in this time interval. Suppose that agent α_j initiates a shadowless traversal at some time t. Since

$dist(\lambda_i, \lambda_{i+2}) > \frac{1}{2k}$ by assumption, it is now known that the next visit of this agent to λ_i takes place after time $t + 1/k > t + I(\mathcal{A}')$. Since λ_i is not occupied by a shadow agent, the agent α_{j-1} must arrive at point λ_i at some time $t' \in (t, t+1/k)$. The previous vital point occupied by agent α_{j-1} before t' must have been λ_{i-1}. Before that, the agent cannot have occupied vital point λ_i, since then, during its traversal from λ_i to λ_{i-1} and back to λ_i, the shadow agent α_{j-1}^{i*} would have existed at λ_i. This shadow agent must have met agent α_i at point λ_i at time t, which contradicts the assumption that agents and shadow agents do not meet. It follows that before arriving at λ_{i-1} agent α_{j-1} must have been located at λ_{i-2}. Thus, agent α_{j-1} was performing a shadowless traversal of $(\lambda_{i-2}, \lambda_{i-1})$. This traversal counts towards $C_{j-1}(i-2, i)$ if agent α_j left λ_{i-2} within the interval $[T_1, T_2]$. Following the reasoning from Lemma 5.13, this obtains the following bound:

$$C_j(i, i+2) - C_{j-1}(i-2, i) \leq 2.$$

Summing the above inequalities over all agents, and performing analysis for the counter-clockwise direction gives:

$$C(i, i+2) - C(i-2, i) \leq 2k.$$

$$C(i, i-2) - C(i+2, i) \leq 2k.$$

Since the number of points $2k+1$ is odd, by iterating the above around the cycle at most $2k$ times in one direction, for any two vital points λ_{i_1} and λ_{i_2} obtains:

$$C(i_1, i_1+2) - C(i_2, i_2+2) \leq 2k^2.$$

$$C(i_1, i_1-2) - C(i_2, i_2-2) \leq 2k^2.$$

Denoting $C^{cw} = \min_{0 \leq i \leq 2k} C(i, i+2)$ and similarly $C^{cc} = \min_{0 \leq i \leq 2k} C(i+2, i)$, provides for any i:

$$C(i, i+2) \leq C^{cw} + 2k^2.$$

$$C(i+2, i) \leq C^{cc} + 2k^2.$$

Now, denote by $W_j(i) \geq 0$ the total time spent by agent α_j at point λ_i and by its shadow α_j^{i*} within the time interval $[T_1, T_2]$, and let $W(i) = \sum_{j=1}^{k} W_j(i)$. Without loss of generality, let λ_0 be a vital point with the minimal total waiting time, i.e. $W(0) = \min_{0 \leq i \leq 2k} W(i)$.

With respect to point λ_0, the trajectory of each agent α_j within the time interval $[T_1, T_2]$ can be described by an ordered sequence of time moments $(e_j^1, l_j^1, e_j^2, l_j^2, \ldots, l_j^{n_j})$, where e_j^p is the time at which agent α_j enters point λ_0 for the p^{th} time after a shadowless traversal (of arc $(\lambda_{2k-1}, \lambda_2k)$ or (λ_2, λ_1)), whereas l_j^p is the time at which agent α_j leaves point λ_0 starting its p^{th} shadowless traversal (of arc (λ_0, λ_1) or $(\lambda_0, \lambda_{2k})$). Assume that $T_1 \le e_j^1 \le l_j^1 < e_j^2 \le l_j^2 < \ldots < e_j^{n_j} \le l_j^{n_j} \le T_2$, and make the same boundary assumptions as in the proof of Lemma 5.13.

During the time interval $[T_1, T_2]$, point λ_0 is covered by some agent or some shadow agent during the set of moments X given as:

$$X = \bigcup_{j=1}^{k} \bigcup_{p=1}^{n_j} [e_j^p, l_j^p],$$

such that $|X| = W(0)$. During the remaining time, i.e. $\overline{X} = [T_1, T_2] \setminus X$, no agent and no shadow agent is located at λ_0. Observe that $\overline{X}$ is a union of at most $1 + \sum_{j=1}^{k} n_j$ intervals. Hence, there exists some time interval of length τ during which point λ_0 remains unvisited, lower-bounded by an inequality of the same form as (5.1):

$$\tau \ge \frac{|\overline{X}|}{1 + \sum_{j=1}^{k} n_j} = \frac{(T_2 - T_1) - |X|}{1 + \sum_{j=1}^{k} n_j} = \frac{(T_2 - T_1) - W(0)}{1 + \sum_{j=1}^{k} n_j} \tag{5.4}$$

Notice that each agent α_j leaves point λ_0 at least $n_j - 1$ times in the time interval $[T_1, T_2]$, embarking on a shadowless traversal of the arc either to point λ_1 (and then to λ_2) or to point λ_{2k} (and then to λ_{2k-1}). Thus, this gives:

$$C(0,2) + C(0, 2k-1) \ge \sum_{j=1}^{k} (n_j - 1) = \sum_{j=1}^{k} n_j - k.$$

Taking into account that $C(0,2) \le C^{cw} + 2k^2$ and $C(0, 2k-1) \le C^{cc} + 2k^2$, this results in:

$$\sum_{j=1}^{k} n_j \le C^{cw} + C^{cc} + 4k^2 + k. \tag{5.5}$$

For any agent α_j, its trajectory can be traced within the time interval $[T_1, T_2]$, looking at the number of shadow agents in time. At any time, α_j and its shadow agents may be waiting at at most two vital points in total. Moreover, suppose that α_j embarks on a shadowless traversal of some arc $(\lambda_i, \lambda_{i+1})$, leaving λ_i at some moment of time t, arriving at λ_{i+1} not earlier than at time $t + dist(\lambda_i, \lambda_{i+1})$. Then, throughout the time interval $[t, t + dist(\lambda_i, \lambda_{i+1})]$, agent α_j can have at most one shadow located at λ_{i+1}. Suppose this shadow agent $\alpha_j^{(i+1)*}$ exists. Then, the last traversal of α_j preceding time t must have been one of the arc $(\lambda_{i+1}, \lambda_i)$, and not shadowless. Tracing back in time the zig-zags of agent α_j between points λ_i, λ_{i+1}, during which it had shadows at both λ_i and λ_{i+1}, leads back to the earliest traversal of arc $(\lambda_i, \lambda_{i+1})$ (or possibly arc $(\lambda_{i+1}, \lambda_i)$, directly after the agent's arrival from λ_{i-1} (respectively, from λ_{i+2}). During this traversal, of duration at least $dist(\lambda_i, \lambda_{i+1})$, agent α_j had precisely one shadow located at λ_i (respectively, at λ_{i+1}). In summary, is has been shown that during every shadowless traversal of arc $(\lambda_i, \lambda_{i+1})$, agent α_j either has no shadow, or it has exactly one shadow and it can be associated with this traversal another time period of length $dist(\lambda_i, \lambda_{i+1})$ during which it has exactly one shadow (with no overlap of time periods). The same argument applies for the counter-clockwise direction. Thus, a bound has been obtained on the total waiting time of agent α_j and its shadows:

$$\sum_{i=0}^{2k} W_j(i) \le 2(T_2 - T_1)$$

$$-2\sum_{i=0}^{2k}(C_j(i,i+2) + C_j(i+1,i-1) - 2)dist(\lambda_i, \lambda_{i+1}),$$

where the constant is subtracted from C_j to account for boundary conditions around times T_1 and T_2. Summing over all k agents obtains:

$$\sum_{i=0}^{2k} W(i) \le 2k(T_2 - T_1)$$

$$-2\sum_{i=0}^{2k}(C(i,i+2) + C(i+1,i-1) - 2)dist(\lambda_i, \lambda_{i+1}).$$

Taking into account that the circumference of the cycle is 1 and that $W(0)$ is the minimum of all $W(i)$, allows:

$$W(0) \le \frac{1}{2k+1}\sum_{i=0}^{2k} W(i) \le$$

$$\le \frac{2k(T_2 - T_1) - 2(C^{cc} + C^{cw}) + 4}{2k+1}.$$

and finally:

$$C^{cc} + C^{cw} < k(T_2 - T_1) - (k + \tfrac{1}{2})W(0) + 2. \tag{5.6}$$

Combining inequalities (5.4), (5.5), and (5.6), obtains:

$$\tau \ge \frac{(T_2 - T_1) - W(0)}{k(T_2 - T_1) - (k + \frac{1}{2})W(0) + 4k^2 + k + 3} \ge$$

$$\ge \frac{1}{k + \frac{1}{2}\frac{2(4k^2+k+3) - W(0)}{(T_2 - T_1) - W(0)}} \ge \frac{1}{k + \frac{4k^2+k+3}{(T_2-T_1) - 2(4k^2+k+3)}} >$$

$$> \frac{1}{k} - \frac{4k^2 + k + 3}{(T_2 - T_1) - 2(4k^2 + k + 3)}.$$

In the above, it is assumed that $(T_2 - T_1) - W(0) > 0$, i.e. there cannot be an agent covering λ_0 throughout the time interval $[T_1, T_2]$. This is true, since otherwise, taking into account that $W(i) \geq W(0)$ for all $1 \leq i \leq k$, all $2k+1$ points would have to be covered by an agent or its shadow agent throughout $[T_1, T_2]$, and there are at most $2k$ agents and shadow agents in total at any time, a contradiction.

Now, suppose that T_2 is chosen to be sufficiently large so that $\frac{4k^2+k+3}{(T_2-T_1)-2(4k^2+k+3)} < \frac{\epsilon}{2}$. This then gives $\tau \geq \frac{1}{k} - \frac{\epsilon}{2}$, and so there exists a vital point on the cycle such that the time between some two successive visits of agents following $\mathcal{A}'$ to this point is greater than $\frac{1}{k} - \frac{\epsilon}{2}$. This completes the proof of the lemma. □

To complete the proof of Theorem 5.11, consider an arbitrary terrain $C = (\Lambda, N)$. Let B be the length of the longest neutral interval of C, as defined in Lemma 5.11. There are two cases to consider.

- If $B \geq 1/(2k)$, then by clause (1) of Lemma 5.12, there exists a subset of $k+1$ vital points $\{\lambda_0, \ldots, \lambda_k\} \subseteq \Lambda$ such that for these points, in Lemma 5.13 gives $s = \min\{1/(2k), L\}$. Now, by Lemma 5.13 it can be obtained that for any strategy $\mathcal{A}$, the idleness is lower bounded by $I(\mathcal{A}) \geq \min\{1/k, 2s\} = \min\{1/k, 2L\}$.
- If $B < 1/(2k)$, then by clause (2) of Lemma 5.12, there exists a subset of $2k+1$ vital points $\{\lambda_0, \ldots, \lambda_{2k}\} \subseteq \Lambda$ that satisfies the assumptions of Lemma 5.14. Thus, by Lemma 5.14 it can be obtained that for any strategy $\mathcal{A}$, the idleness is lower bounded by $I(\mathcal{A}) \geq 1/k$.

In either case, this obtains that the idleness of any strategy patrolling C is at least $\min\{1/k, 2L\}$, which proves the claim of the Theorem.

5.4 Computing Optimal Agent Trajectories

Let $C = (\Lambda, N)$ be the unit segment $[0,1]$ with vital and neutral regions. Assume without loss of generality that the vital intervals in C are arranged in a data structure from left to right as $\Lambda_i = [b_i, e_i]$, for $i = 1, 2, \ldots, n$ where $b_1 = 0$, $b_i \leq e_i < b_{i+1}$. Assume that arithmetic operations involving these values can be performed in unit time.

Recall that in this case the solution is based on the use of lids, where with each lid a different agent is associated. Firstly, it can be shown that one can test in time $\mathcal{O}(\min\{n, k \log n\})$ whether for a collection of k lids each of length d can cover all vital points in $[0,1]$.

A recursive procedure $TestLidSize(k,d,p)$ is proposed that operates on sub-intervals of the form $[p,1]$ of C, where k stands for the number of available lids and $d \leq 1$ refers to the uniform length of the lids. The procedure returns value *true* if all vital points in C can be covered by the collection of k lids. Otherwise the returned value is $false$.

Lemma 5.15. *For any positive integer k, $d > 0$, and $p \in C = (\Lambda, N)$, such that p is vital, procedure $TestLidSize(k,d,p)$ verifies in time $\mathcal{O}(\min\{n, k \log n\})$ whether all vital points in $[p, e_n]$ can be covered by k lids of length d.*

Algorithm 8: TestLidSize(k, d, p): {true,false};

```
Use the next lid to cover segment $[p, p + d]$
if $(p + d) \geq e_n$ then
    return $true$ {all vital points are covered}
end if
$p^* = \inf\{p' \in \Lambda : p' > p + d\}$ {$p^*$ exists since $p + d < e_n$}
if $(k > 1)$ then
    $returnTestLidSize(k - 1, d, p^*)$
else
    return $false$
end if
```

Proof. Firstly it is shown that this recursive procedure performs verification correctly. In the proof induction is used on k. More precisely, it is assumed inductively that for any $1 \leq l < k$ and $q \in C$ the call $TestLidSize(l, d, q)$ verifies whether one can cover all valid points in the interval $[q, e_n]$ using l lids of length d.

Consider the call $TestLidSize(k, d, p)$ in which the first lid is chosen to cover all vital points in $[p, p+d]$. A further recursive call $TestLidSize(k-1, d, p^*)$ verifies whether the remaining $k - 1$ lids suffice to cover all valid points in $[p^*, e_n]$, where $p^* = \inf\{\text{vital } p' \in C = [p + d, e_n] : p' > p + d\}$. By the inductive assumption on k, it is known that this call provides the correct answer. And if this answer is positive, i.e. the vital points in $[p^*, e_n]$ can be covered by $k - 1$ lids it can be concluded that all vital points in $[p, e_n]$ (formed of vital points in $[p, p + d]$ and $[p^*, e_n]$) can be covered by k lids. Alternatively, if $k - 1$ lids are insufficient to cover vital points $[p^*, e_n]$ the extra lid that covers vital points in $[p, p + d]$ is of no use for valid points in $[p^*, e_n]$ since the left endpoint of this lid must be aligned with p. Thus in this case, as expected, the answer computed by $TestLidSize(k, d, p)$ is also negative.

The time complexity $\mathcal{O}(\min\{n, k \log n\})$ is dominated by computation of p^* at most k times, see line 3. If $p+d$ is vital and $(p+d) \neq e_j$, for any $1 \leq j < n$, p^* can be computed in constant time. Otherwise, one must either use binary search on points $b_1, \ldots, b_n$ to find p^* imposing complexity $\mathcal{O}(k \log n)$ or one can search through this list of points in time $\mathcal{O}(n)$. □

To show how to efficiently compute the optimal (minimal) size of the lid, following lemma is needed.

Lemma 5.16. *If L is the optimal (minimal) size of lids, there must be some integer $1 \leq l \leq k$, such that, $L = \frac{e_j - b_i}{l}$, for some $1 \leq i \leq j \leq n$.*

Proof. Consider any cover based on lids with the minimal size. In such a cover one can arrange the lids so that they touch but do not overlap with each other. If such an arrangement is not possible, one could decrease the length of the lids, contradicting the minimality of their length. Thus, it can be assumed that in the cover all of the lids are partitioned into maximal sequences, such that in each sequence the lids are placed tightly one after another, but different sequences do not share their endpoints. Consider any such sequence based on m lids. The left endpoint of the leftmost lid in this sequence must coincide with some b_i. Otherwise, this would not be the leftmost lid in the sequence. If the right endpoint of the rightmost (m^{th}) lid in this sequence coincides with some e_j, the claim of the lemma follows. Assume, to the contrary, that this is not the case for any maximal sequence of lids. This means that the last lid in each maximal sequence overlaps with some neutral region, and consequently, that the length of the lids could be decreased. □

5.4.1 Optimal Lids

The algorithm that computes the optimal size of lids is now presented. Using Lemma 5.16, one can observe that testing at most $\mathcal{O}(kn^2)$ values is needed in search for the optimal size of lids. These values can be sorted in time $\mathcal{O}(kn^2 \log n)$ and later use binary search to find the optimal value. The number of tests during the binary search is $\mathcal{O}(\log n)$ and the cost of each test is $\mathcal{O}(\min\{n, k \log n\})$, see Lemma 5.15. Thus the total complexity is dominated by sorting performed in time $\mathcal{O}(kn^2 \log n)$.

Observation 5.17. The optimal size of lids can be computed in time $\mathcal{O}(kn^2 \log n)$.

The complexity of this algorithm can be further improved if an implicit representation is used of $\mathcal{O}(kn^2)$ candidates based on values $\frac{e_j - b_i}{l}$, for $1 \le i \le j \le n$ and $1 \le l \le k$, and perform search for the optimal size of lids in a more sophisticated fashion. Searches among are performed on values based on each l separately.

Let M_i^l represent implicitly the list of values $(\frac{e_i - b_i}{l}, \frac{e_{i+1} - b_i}{l}, \ldots, \frac{e_n - b_i}{l})$, for $1 \le i \le n$. Each list M_i^l contains at most n values. Any value from this list can be calculated on the basis of the sequence $e_i, \ldots, e_n$, where values e_1 through e_n are stored in an array of length n. In particular, using this representation one can calculate the value of any requested element in M_i^l in constant time.

The search algorithm operates in rounds on all M_i^ls simultaneously. At the beginning all entries in M_i^l are potential candidates for being the optimal length of the lids. During each round the list of candidates in half of the M_i^ls is reduced by half but the remaining candidates in M_i^l always form a sublist of consecutive elements in M_i^l. The reduction process in each round is performed as follows.

Note that the cost of each round, in which there are $a \le kn$ non-empty lists M_i^l, can be bounded by $\mathcal{O}(a + \min\{n, k \log n\})$. The bound on the total number of rounds can now be computed. At the start associate with each list M_i^l a potential of $\log n$. This means that the combined potential of all lists is $kn \log n$. During each round the potential of half of the lists is reduced by 1. Eventually some lists M_i^l become empty which is reflected in the null potential. Note that until at least $\frac{kn}{2}$ lists are non-empty the combined potential is reduced by $\frac{kn}{4}$ during each round. Thus the number of rounds with at least $\frac{kn}{2}$ non-empty lists is limited by $(kn \log n)/(\frac{kn}{4}) = 4 \log n$, and the total duration of these rounds is $\mathcal{O}(\log n \cdot (kn + \min\{n, k \log n\}))$. Furthermore, reduce the number of non-empty lists from $\frac{kn}{2}$ to $\frac{nk}{4}$ also in at most $(\frac{kn}{2} \log n)/(\frac{kn}{8}) = 4 \log n$ rounds,

Algorithm 9: FastLidSearch(l, C): $\{true, false\}$;

```
repeat
  Find the medians $m_i^l$ among the remaining candidates in each $M_i^l$
  Find the median $m^*$ among $m_i^l$, for all $1 \leq i \leq n$
  Use procedure $TestLidSize$ to test whether $m^*$ is long enough
  if $true$ then
    Reduce by half the content of lists $M_i^l$ with $m_i^l > m^*$
  else
    Reduce by half the content of lists $M_i^l$ with $m_i^l < m^*$
  end if
until only one candidate value is left
```

and the total duration of these rounds is $\mathcal{O}(\log n \cdot (\frac{kn}{2} + \min\{n, k \log n\}))$. Thus, if one continues this process until only one element in one list is left, the total time of execution is bounded by:

$$\mathcal{O}\left(\sum_{j=0}^{\log(kn)} \left(\log n \cdot (\tfrac{kn}{2^j} + \min\{n, k\log n\})\right)\right) =$$

$$= \mathcal{O}(kn \log n + \log n (\log n + \log k) \min\{n, k \log n\}) =$$

$$= \mathcal{O}(kn \log n + k \log^3 n + n \log k \log n) = \mathcal{O}(kn \log n).$$

Corollary 5.18. *The optimal size of lids can be computed in time $\mathcal{O}(kn \log n)$.*

This approach is also applicable to the combined strategy on the cycle, since, in fact, the optimal lid size only needs to be computed in the case when the cycle contains some neutral region N_i of length at least $1/(2k)$. Then, the problem on the cycle C reduces to that on the closed segment $C \setminus N_i$. This provides the following:

Theorem 5.19. *Consider k agents patrolling a boundary cycle (resp., segment) with n vital regions. The agent trajectories which result in minimal idleness can be described using the combined strategy (resp., the partition strategy). Such a description can be computed using an $\mathcal{O}(kn \log n)$ algorithm.*

Chapter 6

Other Work

In this chapter some of the other work that has been conducted in relation to the main body of this project is discussed. Specifically, other models and aspects are looked at in relation to distributed computing. The first section looks at *Parasitic Computation* a method for performing distributed computation tasks using existing web technologies. The second section discusses the *Basic Walk* concept and the experimental results obtained through using this method on various graph structures. Finally the third section shows a summary of the work completed in *Graph Visualisation and Analysis* during the production of *"GraphDraw"*.

Unlike previous chapters, no work in this chapter has yet been published. The bulk of this work was executed as a side interest by the author and for that reason has been completed predominately by the author, however, a number of contributions were made by third parties. Firstly, the formula discovered in §6.1 was discovered during the authors internship at "*Institut national de recherche en informatique et en automatique*" ("National Institute for Research in Computer Science and Control"), which took place at the *"Laboratoire Bordelais de Recherche en Informatique"* ("Bordeaux Laboratory for Research in Computer science") with the assistance of Dr. Ralf Klasing, and Dr. Evangelos Bampas. Further, in §6.2.2, the results in relation to logarithmic cycle length were discovered with the assistance of Prof. Leszek Gąsieniec, Prof. Evangelos Kranakis, and Dr. Russell Martin.

Secondly, it should also be noted that while the third section on graph visualisation and analysis was initiated by the author due to it being a by-product of the *Basic Walk* work, this work is no longer continued alone as a number of undergraduate and postgraduate students have since become involved in contributing new algorithms to the software. The work completed by the students, so far, is available in [102, 179]. The algorithms utilised in this project are cited throughout the section itself to their respective authors.

6.1 Parasitic Computation

In this chapter the work that was performed on parasitic computing is shown. Firstly, before the model and methodology is shown for this work, the terminology that is used must be defined.

Definition 6.1 (Parasite - *Biology*)**.** An organism which lives in or on another organism (its host) and benefits by deriving nutrients at the other's expense [161].

A parasite is defined as an entity which exploits the resources upon which the host has available to it. A point that should be made is that the parasite is not defined as intentionally trying to harm the host, though this may happen, but instead simply trying to gain something for free at their expense. In contrast the definition of a computer virus should be looked at.

Definition 6.2 (Virus - *Computing*)**.** A program or piece of code which when executed causes itself to be copied into other locations, and which is therefore capable of propagating itself within the memory of a computer or across a network, usually with deleterious results [162].

In this definition the difference between the two terms can be seen, where as a parasite simply exploits the resources of a host, a virus not only exploits the host but also tries to spread itself without care for the integrity of the host. This leads to the requirement of a new definition, a computational parasite, which is defined as a process that exploits the resources of a host but does not attempt to spread itself or harm the host. Specifically:

Definition 6.3 (Parasitic Computing)**.** The concept of exploiting existing protocols to unwittingly force a host machine to execute known protocols in a specific manner so as to inadvertently cause the host to perform computation for a third party with no direct benefit to the host machine.

6.1.1 Background

In a previous work by the author [51], the Bailey-Borwein-Plouffe Algorithm (BBP) [22] was implemented in the Java programming language to generate the binary expansions of the number π for the purpose of experimenting with irrational numbers as a PRNG. It is within this work that the following observations were made.

Observation 6.4 (Parallelism of n digits). The BBP algorithm is classed as a spigot algorithm, an algorithm which does not generate a sequence in its entirety or to a given n but instead only generates the value at the n^{th} position. Therefore n digits of π can be generated by n processors independently of each other.

Observation 6.5 (Parallelism of the nth digit). To generate a single digit of π requires the computation of a summation, however, each iteration of the summation is independent of all other iterations in the summation thus allowing for a single digit to be split across multiple processors.

Through utilising Observation 6.4, it is easy to see that the BBP algorithm can be used in a distributed setting where by one would like to compute n digits by p processors. Each processor within p is assigned a digit from $\{0, \ldots, n\}$ to compute until all digits up to n are assigned and computed. However, assigning work using this method is not necessarily suitable as the difficulty of computing the n^{th} digit grows in the order of $\mathcal{O}(n \log n)$. Due to this there will exist a threshold such that the cost of calculating the n^{th} digit will become greater than the amount of time that can be feasibly assigned to a given processor.

To counter this issue, Observation 6.5 becomes of interest. Through utilising Observation 6.5 a single processor will be able to share its workload with another (or many) processors until the workload returns to a manageable size. More so these two observations can be applied into a MapReduce model [19, 68].

Observation 6.4 is of course trivial and in no way surprising, and further Observation 6.5 is also well known and has been previously exploited by the PiHex project [149], a distributed attempt that found the quadrillionth hexadecimal digit of π, which at the time was unknown.

The third observation which was made is in how the algorithm is constructed.

Observation 6.6 (Arithmetic of BBP). All operations within BBP are comprised of simple mathematical operations, using only the operators; addition, subtraction, multiplication, division, modulus, and exponentiation within $\mathbb{R}$.

Through this observation it becomes possible to realise that the BBP algorithm may be implemented in most programming languages with relative ease.

As it is now clear that this algorithm is easily parallisable in many programming languages, it allows for the realisation that this algorithm may map quite easily to a distributed project. For instance in reviewing some of the most noteworthy projects[1] such as the Great Internet Mersenne Prime Search (GIMPS) [135], Folding@Home [186], and SETI@Home [176], a common issue existing in all of these is that software is required to be be installed on the user's system. Something that is not always possible due to security, general acceptance (due to trust), or possibly many other issues. This leads to the question, can a distributed project be constructed that does not require the user to install any third party software and only utilises what is generally assumed to be available to most Internet connected personal computers.

6.1.2 This work

Due to this an alternative distributed model was considered that would allow for the BBP algorithm to be implemented in a distributed manner without the requirement of a third party software installation using the concept of parasitic computation.

[1]Noteworthy for their mainstream acceptance and consequently high volume of accessible computing power.

One such option to meet this target is to use a language such as Javascript. Javascript is a scripting language used in web browsers to allow a website to add interactive actions to its pages. Interestingly, Javascript contains all of the necessary functionality, and if implemented correctly could mean that a distributed project could be run entirely through a web browser with no additional software installation. Not only is Javascript able to perform the necessary arithmetic operations, it is also capable of remotely retrieving or storing data from an external resource using a technique known as Asynchronous Javascript and XML (AJAX). This would mean that it would be possible to use Javascript as a *client* to remotely request a *job* from a *server*. Specifically a Javascript process is able to request a job which would contain the n^{th} digit, calculate it, and then return the calculated n^{th} digit back to the server for storage, as per the traditional *client-server model*. Finally, as Javascript is generally executed without user interaction, this would allow for the possibility of a user to casually access a web page and unwittingly generate a digit of π as per the principle of parasitic computation.

To test this model a prototype was built that would allow a user visiting a web page with any standards compliant web browser, which supports Javascript, to generate a digit of π. For the model to work, a user must be able to generate an arbitrary digit selected by the server, be able to return it for storage, and must not be forced to use any extensions, plugins or such forth. In the prototype it was selected that the prototype would generate continuously further values of n starting from the 0^{th} digit. This would give sufficient data to test the model and also experimentally assist in identifying strengths and weakness of the model. Further it would take a considerably large value of n before a user would be unable to process a single digit in a reasonable amount of time thus preventing the need to cater for Observation 6.5. For ethical reasons the only modification from the previously discussed model that was made was that the user would be required to interact with the web page to voluntarily choose to contribute towards the project.

Since development of the prototype nearly $8,000,000$ of the first digits of π have been generated from approximately 300 unique users from across industry and academia[2]. While firstly these results show that the model in itself is feasible, it secondly has lead to a number of observations.

Observation 6.7 (BBP allows trust). Generating the n^{th} digit of π does not only generate the n^{th} digit but also the next k digits where k is related to the length of the data type that is used.

As a consequence of Observation 6.7 if n digits are generated where $n > k$ then it is possible for the $n-k$ to $n-1$ digits preceding n to predict the expected value of the n^{th} digit, thus allowing for a level of trust to be given to the processor of the n^{th} digit. When floating point data types are used, some of the digits will be erroneous due to a loss of precision. Further a weaker form of trust can be given by looking at the $n+k$ digits, if the n^{th} digit predicted these values correctly then it is possible to assume that the n^{th}

[2]IP address logs have shown digits were generated from within a number of university and from high profile industrial interest such as Intel Corporation.

was from a trustworthy source and also correct. For instance see Figure 6.1 which shows the digits generated for $0 \leq n \leq 9$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	**2**	4	3	f	6	a	8	8	8	5									
1		**4**	3	f	6	a	8	8	8	5	a								
2			**3**	f	6	a	8	8	8	5	a	3							
3				**f**	6	a	8	8	8	5	a	3	0						
4					**6**	a	8	8	8	5	a	3	0	8					
5						**a**	8	8	8	5	a	3	0	8	d				
6							**8**	8	8	5	a	3	0	8	d	3			
7								**8**	8	5	a	3	0	8	d	3	1		
8									**8**	5	a	3	0	8	d	3	1	3	
9										**5**	a	3	0	8	d	3	1	3	1

TABLE 6.1: The result of generating the first 10 hexadecimal digits of π. Table shows that with each digit, an additional $k = 9$ digits are also generated.

A result of this observation is that this algorithm can be safely distributed in an environment with malicious and or corrupted processors, as long as the allocation of work is done safely and securely (i.e. no single processor will generate a pair of digits, n and m where $|n - m| \leq k$) so as to prevent a processor gradually manipulating a sequence of length k. In a technique such as this, trust is a significant issue since all aspects are open due to the nature of Javascript being that of an interpreted language. Not only is the source code of the algorithm public (though obfuscation is possible it is not recommended [155]), all aspects of data transmission are also visible since it runs over known protocols (Hyper-Text Transport Protocol (HTTP), etc.). The result of this is that all work must be easily verifiable by the server otherwise any algorithm run through this method has a serious attack vector which can be exploited. In the BBP algorithm, it is feasible to verify a task by comparing a processor's results to that of another processor. However, if a different algorithm is used to solve a different problem then a new method of verification is required.

Observation 6.8 (Scaling to NP). Parasitic computing is feasible for many problems in NP that are suited for the Javascript language.

This leads to the next matter of interest, the application of this technique. While generating the expansions of π is interesting, it is perhaps not of general interest, and further more there are faster algorithms [37, 47] that have been implemented that have long since discovered more digits than this method will be able to handle [30, 114, 115] in the near future. This leads to the question, can the method be used for other problems. From the experimental work conducted it becomes feasible to identify potential suitable problems. Particularly this method is well suited for problems where the cost required to identify an answer the problem is high, while the cost to verify the correctness of an answer is low. Specifically, if a large workload can be passed to a processor to compute and the server is only responsible for verifying that the answer is correct then the requirement of trust is bypassed as a server can simply reject answers which are incorrect, and then reassign the same job to a different processor.

Finally as the model can now be abstracted to solve any problem that is in NP (though Observation 6.6 must hold), motivation is now needed as to why a processor may want to join *the grid* and donate its computational power. Due to the nature of this model already running completely on the web and being transparent (as per its parasitic nature) then an application is perhaps easy to find. Consider for instance the well known, and unappreciated, but widely used Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) that is used to keep robots away from certain resources [139]. The purpose of a CAPTCHA is to identify a robot from a human by presenting the user with a question that a computer cannot answer. While many CAPTCHA's are successful in this objective, most will eventually become defeated by a computer as Optical Character Recognition (OCR) algorithms improve and more computational power becomes available. For the sake of simplicity one could utilise the technique outlined so far to create a "*Computational CAPTCHA*"[3] in that the human and robot are not tested, but are instead required to *work* for access to a resource, similar to the concept of *pricing functions* as proposed by Dwork and Naor [75] for e-mail. While both the user and the robot can pass this test, one may notice it while the other may not, if each access to a resource requires 30 seconds of computation time, then this would be unnoticeable to a human as the computation could be completed in the time it takes to read or write text (e.g. a message board post, registration form, etc.), or further this could be combined with existing CAPTCHA techniques. However, to a robot this would require 30 seconds of resources to complete a task that otherwise would take a trivial amount of time, the result is that a resource, and further, a financial cost is attached to accessing a resource. Further, as computation improves, the problem difficulty can increase. The problem that is used could be that of academic or commercial importance such as what is performed in existing distributed projects, or it could be something such as BitCoin mining[4] [138] to provide a means of requiring work to access a resource and a financial pay off to the resource owner as an alternative to existing advertisement based revenue models.

6.2 Basic Walk

As a part of the investigation into the *Basic Walk* in networks with randomly arranged ports, one of the main goals was to develop a simple usable software solution that would provide a platform for the generation of various 2D square grids (e.g. tori) firstly to gain familiarity with the concept of the basic walk, and secondly to gain more intuition concerning the structure of cycles formed through the execution of the basic walk concept. The early direction of the work was to take a more thorough look at [39] to ascertain the correctness of Brunell's results and to provide a starting point for this work. However,

[3]Clearly the naming is inappropriate but this name may emphasise what is being proposed.

[4]The author has not investigated the feasibility of this specific application, however, it serves as an example.

[39] is by no means itself an in-depth study of the basic walk as both its software solution and the quality of the observations are limited.

In this work, the length of each cycle and the length of the longest cycle are investigated.

6.2.1 Average cycle length

The initial experiments for the n^2 2D square grid produced results consistent with those results shown in [39], that were first observed by Kosowski et al. [116], particularly that the average length of all cycles would be approximately 78. However, through the experiments performed in this work it will be shown that this value converges to $78.6 \pm \epsilon$.

To identify this result, the following methodology was performed. For each n where $n = \{2, \dots, 2000\}$, 100 randomised n^2 grids were generated, and the basic walk was performed on each. The purpose of the increasing values of n was to try to identify whether the average cycle length was the result of a very slow growing function (such as $\log \log$) or was in fact a constant value as originally suggested. Secondly, the purpose of the 100 iterations per n were performed to dampen noise that could occur in the results due to the random nature of the model.

Through recording the average number of vertices and edges in all experiments, Figures 6.1 and 6.1 can be produced. Through these results, not only is the existing observation made on the average cycle length further refined but a new and previously unknown constant is discovered.

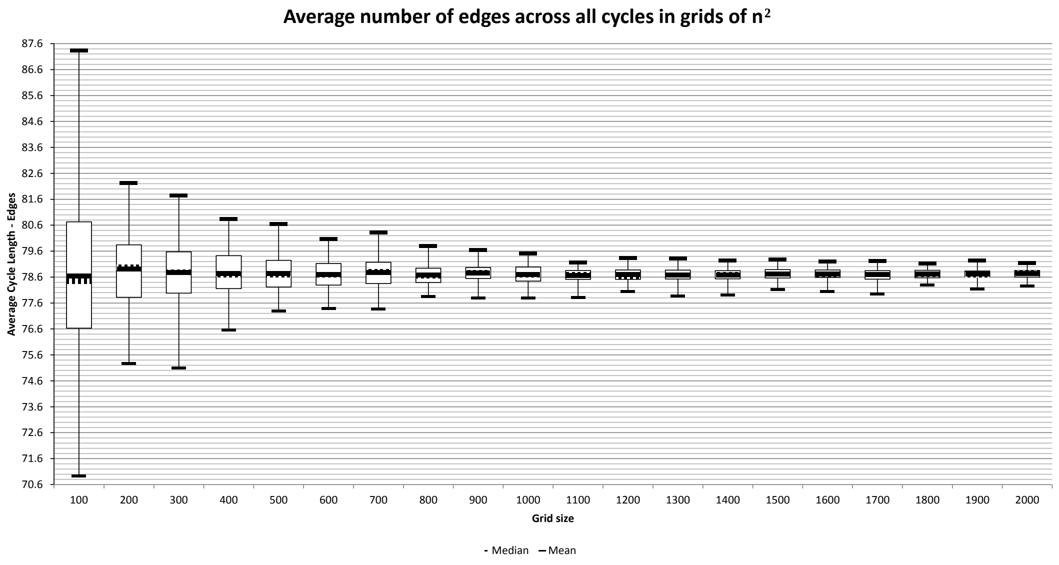

FIGURE 6.1: Average number of edges across all cycles in grids of size n^2 (100 to 2000)

Observation 6.9 (Average number of edges in a cycle). For any grid n^2 where $n > 100$, the average number of edges within a cycle is $78.6 \pm \epsilon$.

Observation 6.10 (Average number of vertices in a cycle). For any grid n^2 where $n > 100$, the average number of vertices within a cycle is $41 \pm \epsilon$.

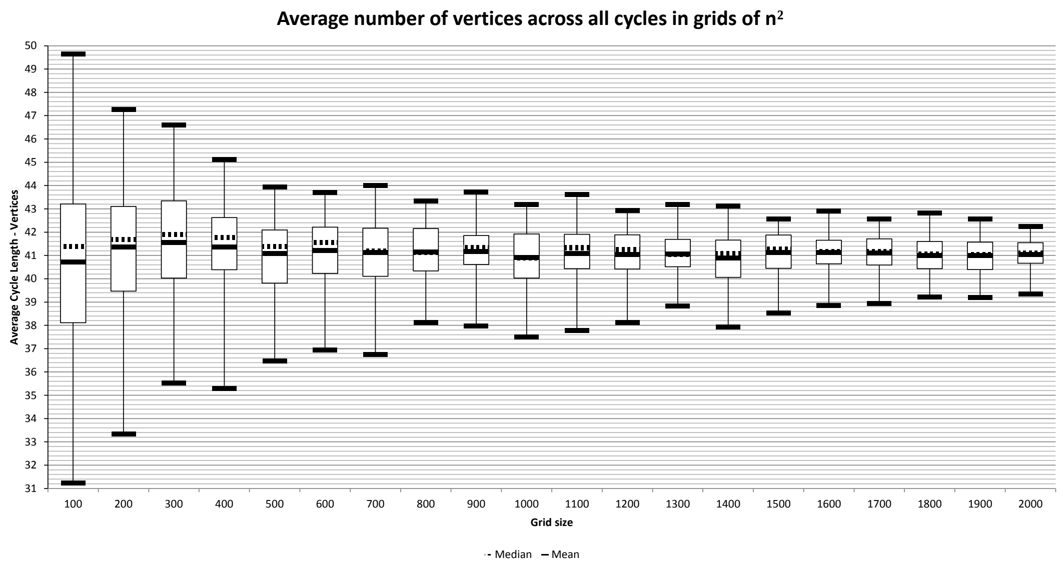

FIGURE 6.2: Average number of vertices across all cycles in grids of size n^2 (100 to 2000)

As can be seen in Figures 6.1 and 6.1, neither of the results appear to show any indication that the value grows as n grows, thus indicating that either the rate of growth is incredibly slow as to be unnoticeable or that in fact the values observed are constants. The reasoning for these constants is as of yet, *unknown.*

In addition to the previous two identified constants, similar constants were also observed in both 2D triangular grids and 2D hexagonal grids using the same methodology. Table 6.2 shows the constants that were identified.

	Edges	**Vertices**
Hexagonal	102	62
Square	78.6	41
Triangular	78	32

TABLE 6.2: Average number of edges and vertices in 2D regular grids. All values are approximations

6.2.2 Longest cycle length

Through investigating the average cycle length it was observed that for nearly all graphs their is always a cycle that contains significantly more edges than all other cycles. This cycle is referred to as the *longest cycle.*

This cycle is interesting from the perspective of creating new network topologies as it is hoped its presence will allow data or agents to traverse this large cycle and simply hop off when they reach their destination. Similarly as to how one would hop onto a motorway and then hop off when one reaches their destination (or local area of). Particularly this investigation focuses on the properties in the 2D square grid, however, the longest cycle has been observed in various real world graphs, sensor networks and various other grids.

n^2 2D Square Grids

Through experimental analysis of n^2 2D square grids the following observations were made.

Observation 6.11. For any n^2 grid of adequate size, there will always exist a longest cycle that contains approximately $\frac{1}{2} \cdot |V|$ of vertices within the graph, and further will also contain approximately $\frac{1}{3} \cdot |E|$ of all edges in the graph.

The empirical results to support Observation 6.11 are shown in Figures 6.3 and 6.4.

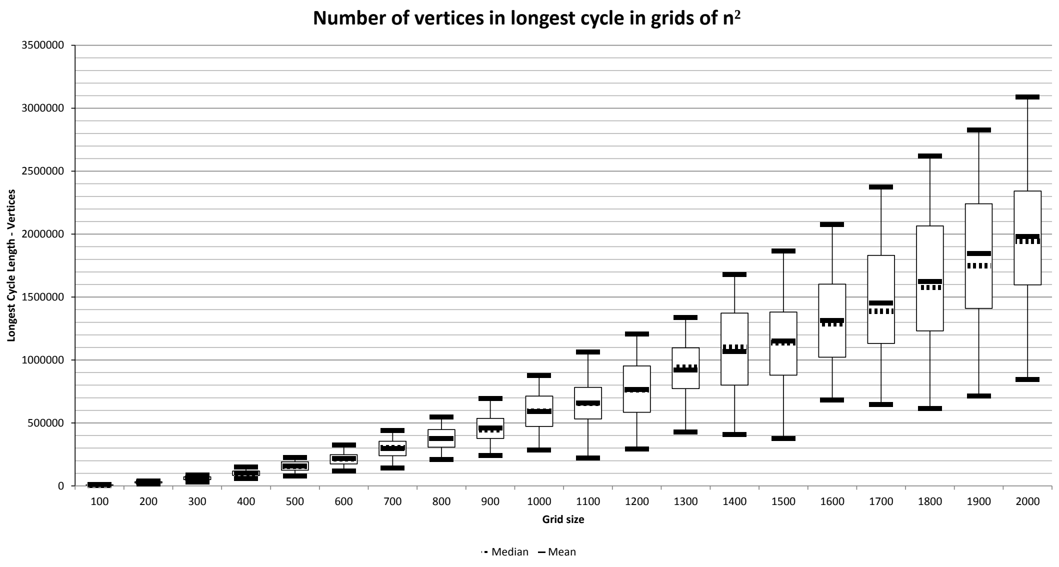

FIGURE 6.3: Number of vertices in the longest cycle in n^2 grids

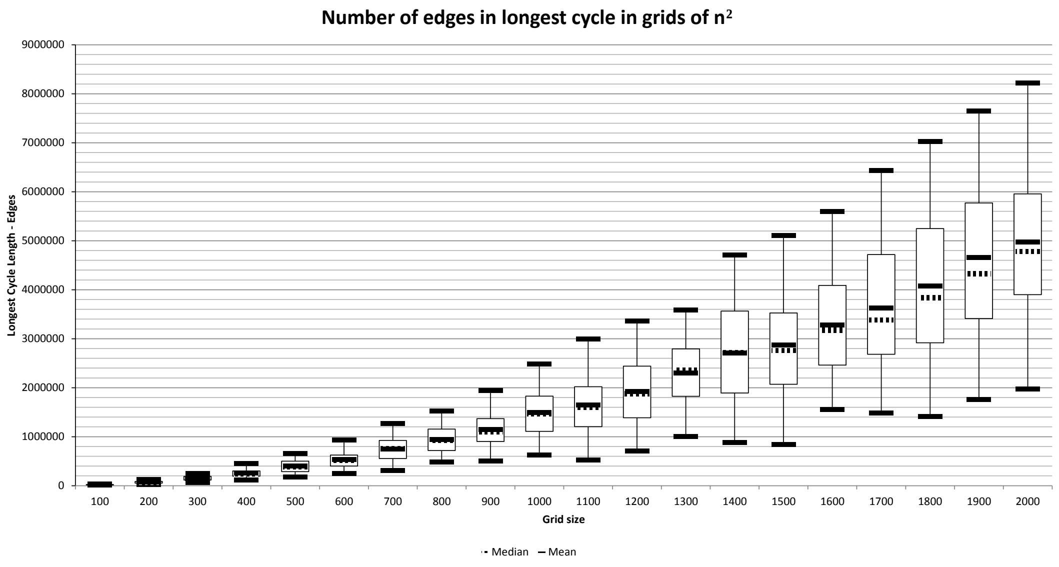

FIGURE 6.4: Number of edges in the longest cycle in n^2 grids

The significance of this observation is that it allows for the consideration of this model as a means of routing data through a network, while there is wastage within the route, this can be offset by providing more resources to the route, such as for instance how a motorway network may not be the shortest path between point A and point B, however, this is offset by the greatly increased capacity and throughput available to the path.

Figure 6.5 shows an example of a grid with a longest cycle which covers most of the space and any area not covered is a relatively short distance, further reinforcing the previous suggested application.

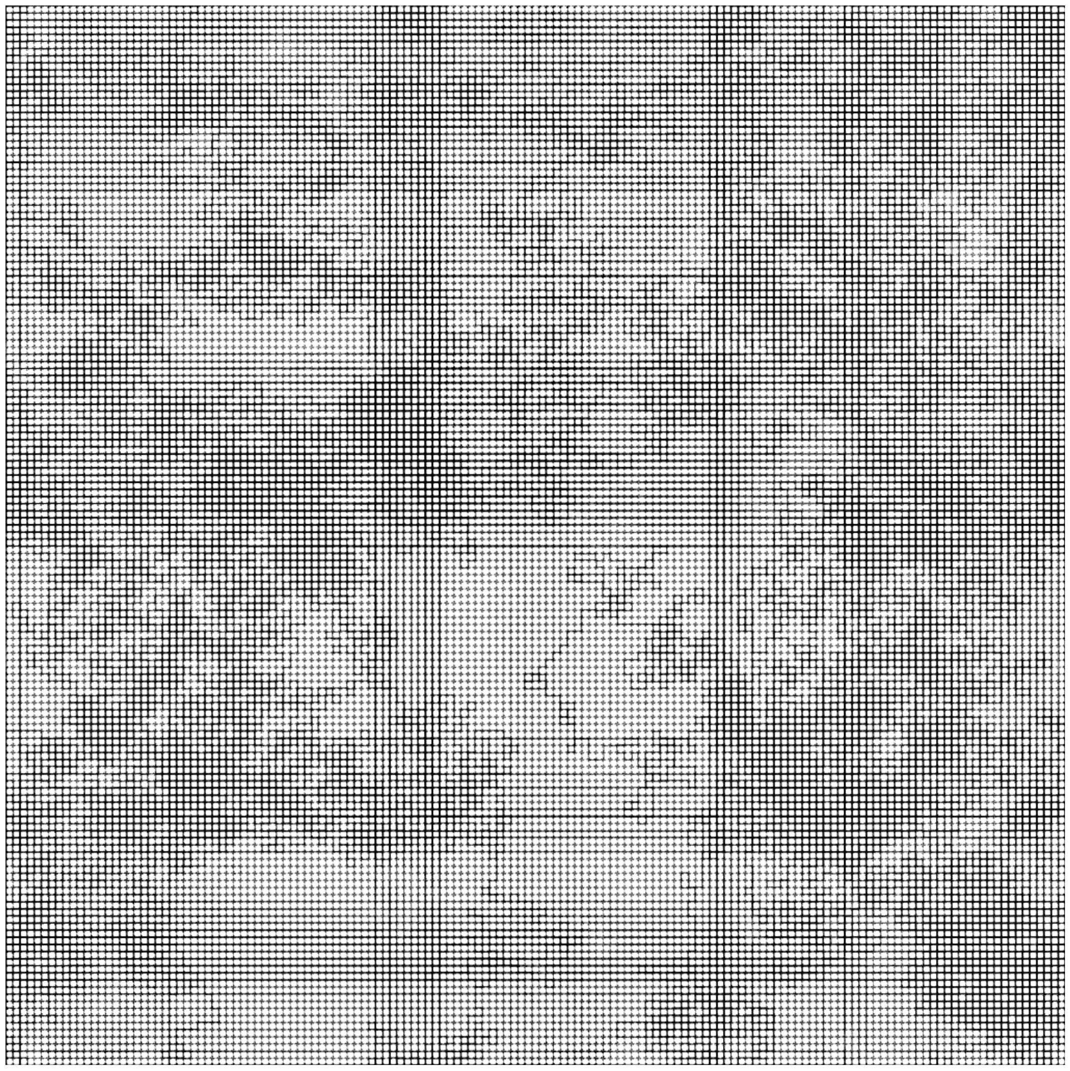

FIGURE 6.5: The basic walk performed on a 150^2 2D Square grid. Longest cycle is the black (or darkest coloured) cycle.

$k \times n$ 2D Square Grids

However, this observation does not appear for *any* grid size, it only occurs (that has so far been observed) in n^2 grids. If a grid of a size $k \times n$ where k is a fixed size the longest cycle appears to be logarithmic in length as can be seen in Figure 6.7. Specifically,

Observation 6.12 (Logarithmic run in $k \times n$ girds). For any grid $k \times n$ where $k \neq n$ the longest cycle will be logarithmic in length.

Certainly in the case where $k = 2$, it appears that their may be a good reason for this to occur, consider that in the $2 \times n$ grid there are 3! possible port arrangements, disregarding rotations this gives 2 possible port arrangements for any given vertex (excluding corner vertices). For the sake of simplicity these vertex arrangements can be labelled arbitrarily as A or B. Further, it can be stated that should their ever exist two neighbouring vertices with the same port labellings (i.e. $A \rightarrow A$ or $B \rightarrow B$) then these two vertices will contain the same cycle as it will be connected through those two vertices (Figure 6.6). Finally, in a sequence of length n, what is longest sub-sequence (or longest run) that contains solely A or B?

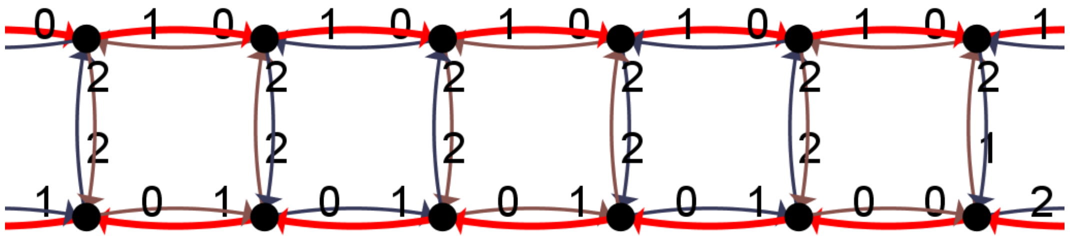

FIGURE 6.6: An example $2 \times n$ grid, with neighbouring vertices sharing a common port arrangement

Schilling [157]'s award winning article provides a brief overview as to what could be expected in a similar situation, the longest run of heads in a series of coin flips. In [157], the author refers to the "$\log n$ law", in that the longest sub-sequence will be logarithmic to the number of coins. It is this $\log n$ law that is occurring within the $k \times n$ grids that have been tested. However, as of yet it has not been identified at what point the longest cycle ceases to be logarithmic and becomes linear, or as to why it eventually becomes linear. Further to this, it has not yet been identified whether any of these properties exist in similar grids (hexagonal or triangular). What has been observed is that most graphs do seem to have the property of a longest cycle.

Distance to longest cycle

With the property shown that their exists a longest cycle and it covers a large proportion of the grid, the next logical question to ask is, from any other cycle, what is the distance (in terms of number of hops) to get to the longest cycle (i.e. what is the maximum eccentricity of the largest cycle). In addition to this, can this be applied to any, real world, graph. To test this graphs from [126] were investigated.

Figure 6.8 shows the longest eccentricity in all[5] of the SNAP graphs. The results were obtained by taking each graph, and firstly mapping each edge to that of an undirected digraph before secondly reducing them to the largest connected component. After this 100 iterations of assigning random ports to each edge with the basic walk being performed on each iteration. The average of the longest eccentricities are shown in the chart.

[5]This excludes the "Memetracker" and "Twitter" graphs as these networks as simply too large for any resources that are available to the author. Further, it should be noted that the archive of SNAP graphs may have expanded since this work was completed.

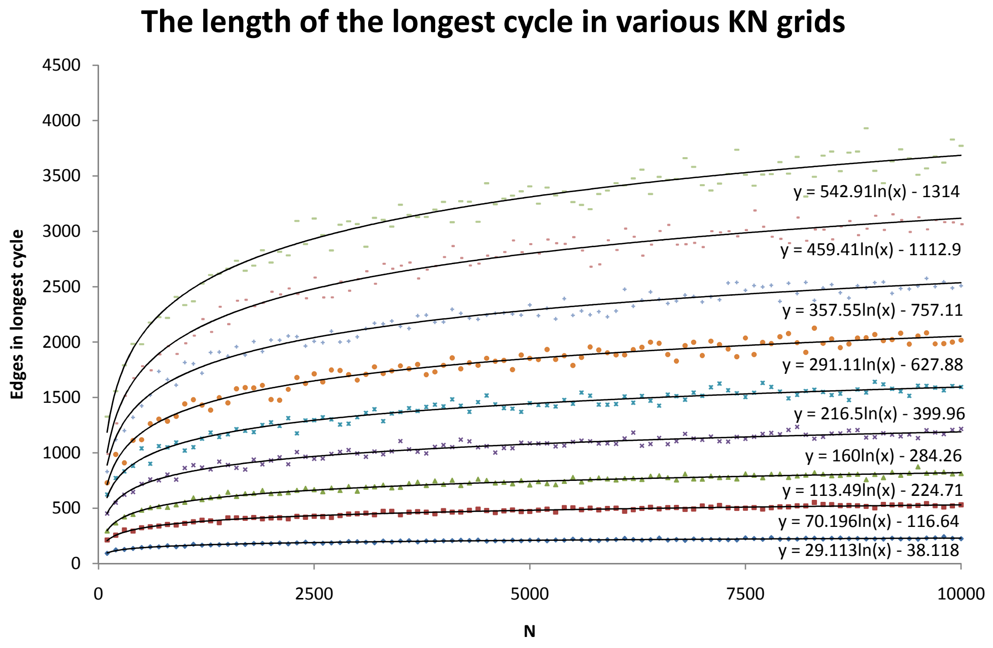

FIGURE 6.7: Number of edges in the longest cycle in $k \times n$ grids

In Figure 6.8 the reader may notice that certain graph types (*roadNet-X* and *web-X*) seem to perform poorly in comparison to the other graphs. While these graphs have been too large to visualise to see exactly why the eccentricities are so large in comparison to the others, it is possible to hypothesis a possible reason. The road networks are for an entire state, and presumably this means there are various small communities (villages, towns, cities) which are quite dense within the graph, all of which are connected by a few sparser connections (i.e. motorways, small roads). The side effect of this is the basic walk may have difficulty escaping a community resulting in several large cycles forming for each of the clustered areas. Similarly, it is possible a related issue occurs with web graphs as their is most likely various strongly connected areas within a university website (e.g. faculties, departments, or research groups), and then only sparse connections between them. However, this property is interesting in itself as it could suggest that the basic walk could be used as a method of identifying communities such as how random walks are currently already used [13, 41, 107].

With exception of the graph types mentioned it seems that in general the eccentricity of the longest cycle to all other cycles is very low, generally ≤ 3, with even in the exceptional cases it is possible for the eccentricity to be low as shown by *web-NotreDame* and *web-Google*. This further reinforces the possibility that this methodology could be used as a routing mechanism.

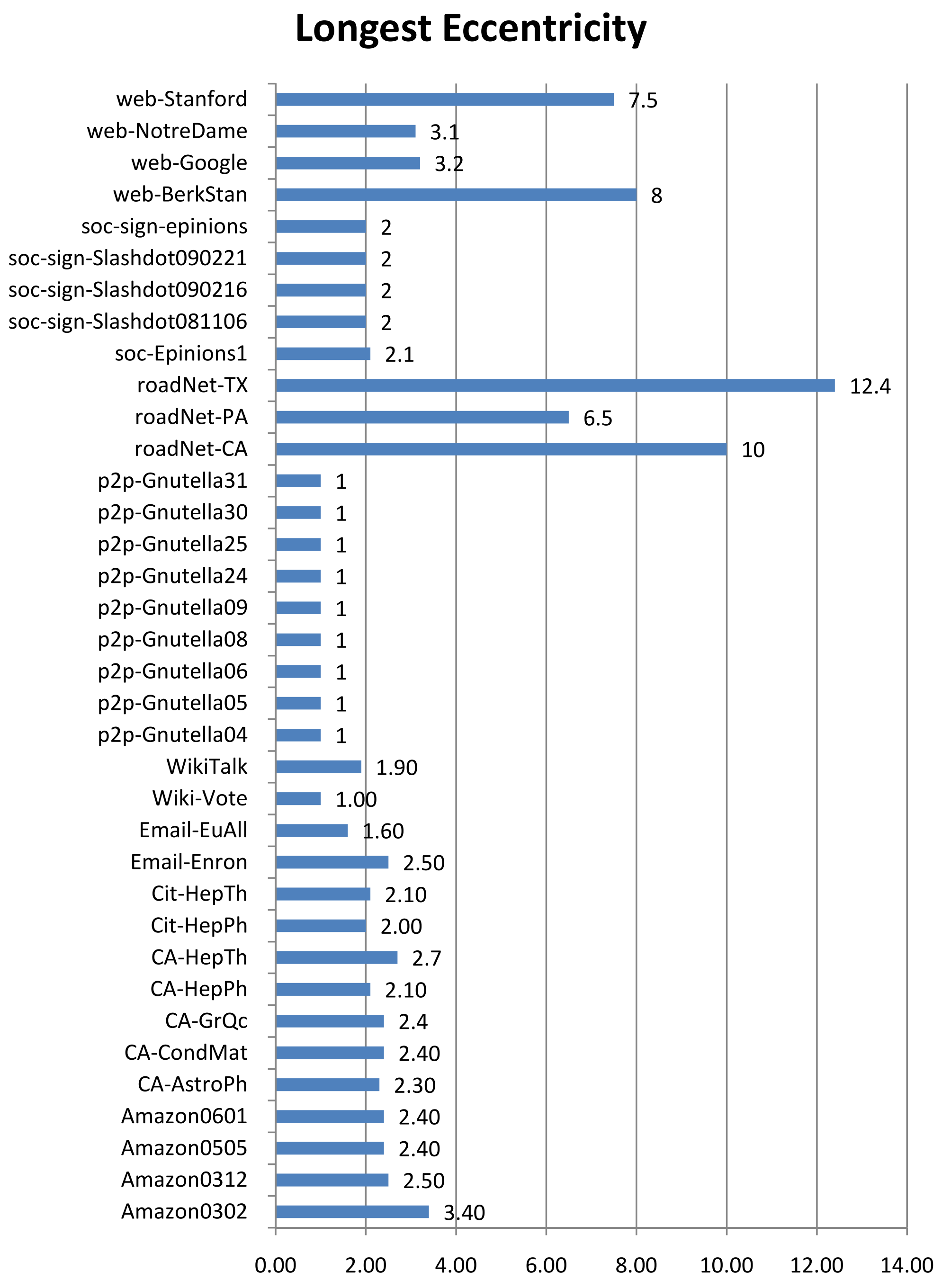

FIGURE 6.8: The average maximum eccentricity of the longest cycle in various SNAP graphs

6.2.3 Probability of a cycle of given length appearing

To investigate both the longest cycle and the average cycle length, the structure of all cycles was considered. Specifically, for any n^2 grid, how many cycles of length m will approximately appear. If C contains the set of all possible cycles of length m then the following would answer this:

$$Pr = \sum_{c \in C} \frac{2^{c^s}}{6^{c^t}} \tag{6.1}$$

Which when multiplied with the size of the grid will provide an approximation of the number of cycles of length m.

$$\sharp m = n^2 \cdot Pr \tag{6.2}$$

Recall that each vertex has 6 possible permutations and that further once any vertex is utilised once, there is only 2 remaining possible permutations for the vertex to be assigned. Which further leads to c^s is defined as the number of vertices which are visited singularly and c^t is defined as the total number of vertices in cycle c where $s \leq t$.

However, what should be noted in this formula is that it depends on prior knowledge of the entire set C for any given m which so far there is no known formula for calculating these values. Once m becomes greater than 10 it becomes infeasible to calculate these numbers manually as already for $m = 10$ their are 120 possible cycles. For this reason a brute force cycle generator was developed to identify the number of cycles for larger values. However, despite heavy optimisation the algorithm still had a theoretical cost of $\mathcal{O}(m^3)$ thus limiting the results to $m \leq 34$. This bound is related to the fact that the algorithm discovers cycles through the method of entering a vertex and has three possible exits to explore, thus producing a ternary tree. The full results generated are available in Appendix A and can be used to feed into Equation 6.1. Table 6.3 contains the precomputed Pr values, though for better accuracy in large n, re-computing the values with higher precision may be necessary.

To check the accuracy of the formula, a series of experiments were performed to provide empirical results as to the the actual distribution of cycles. Table 6.4 shows the actual number of cycles of length $4 \leq m \leq 34$ from every n^2 grid $100 \leq n \leq 1000$ at intervals of 100. For each experiment, 100 iterations were generated and the averages number of cycles is shown in the table. While, as expected, the formula does not produce the exact count, however, the margin of error can be seen to be very small providing some level confidence in the correctness of the formula.

Table 6.5 shows a summary of the types of cycles that were identified. In this table the cycles are categorised into two types of single, a self-avoiding cycle and a self-intersecting cycle. A self-avoiding cycle is defined as a cycle that only visits a cycle once, i.e. there is no point in the cycle when the walk intersects. A self-intersecting cycle is the counter of this definition, in that it is defined as the set of cycles which contain vertices that are visited more than once.

Most interesting, yet perhaps not surprising, is that one of these sequences is well known. While the the total number of cycles appears to match no know sequence, one of the component parts does. What is referred to here as a self-avoiding cycle is well known

m	Pr Value
4	0.024691358024691
6	0.0054869684499314
8	0.0039628105471727
10	0.0029805754542837
12	0.0019512984508158
14	0.0014551631005762
16	0.0011655010842754
18	0.00090980297250102
20	0.0007474755606491
22	0.00062426369979494
24	0.00052549119764244
26	0.00045000966789279
28	0.00039062296525159
30	0.0003411190501216
32	0.00030079203258124
34	0.00026736342673897

TABLE 6.3: The Pr values computed when calculating the chance of a cycle of a given length m occurring

in the physics and chemistry literature as a Self Avoiding Polygon (SAP) and has been investigated at least since the 1950's [172–174] with earlier works in a slightly different context by Wakefield [171] and Orr [146], in 1951 and 1947 respectively. While the early works were limited due to the lack of access to modern day computing, the work is still of interest today and recent works have greatly enhanced the earlier works to identify higher values of m [49, 108]. However, even still Clisby and Jensen [49] have so far only managed to reach $m = 130$ after using 25,000 CPU hours using 1,000 processors and 2.5TB of memory. The results of Clisby and Jensen can be found in [164].

Despite so far over 60 years of research in what seems to be a trivial problem, these values are still only being identified through the means of brute forcing them and future values can only be predicted using an asymptotic formula. More aptly the opening paragraph in the preface of the book by Guttmann [101] states the situation most clearly,

> The problem of counting the number of self-avoiding polygons on a square grid, either by their perimeter or their enclosed area, is a problem that is so easy to state that, at first sight, it seems surprising that it hasn't been solved. It is however perhaps the simplest member of a large class of such problems that have resisted all attempts at their exact solution. These are all problems that are easy to state and look as if they should be solvable. They include percolation, in its various forms, the Ising model of ferromagnetism, polyomino enumeration, Potts models and many others. These models are of intrinsic interest to mathematicians and mathematical physicists, but can also be applied to many other areas, including economics, the social sciences,

m	A/E	100	200	300	400	500	600	700	800	900	1000
4	A	242.52	982.39	2,207.99	3,908.50	6,101.62	8,807.17	11,992.94	15,656.31	19,812.84	24,447.41
	E	246.91	987.65	2,222.22	3,950.62	6,172.84	8,888.89	12,098.77	15,802.47	20,000.00	24,691.36
6	A	54.57	216.59	491.47	873.71	1,354.65	1,949.62	2,664.65	3,471.15	4,396.02	5,437.47
	E	54.87	219.48	493.83	877.91	1,371.74	1,975.31	2,688.61	3,511.66	4,444.44	5,486.97
8	A	39.59	158.42	353.63	629.77	983.79	1,409.39	1,919.26	2,506.16	3,177.84	3,924.30
	E	39.63	158.51	356.65	634.05	990.70	1,426.61	1,941.78	2,536.20	3,209.88	3,962.81
10	A	29.06	119.01	268.93	471.13	739.52	1,068.26	1,443.16	1,892.08	2,393.43	2,969.38
	E	29.81	119.22	268.25	476.89	745.14	1,073.01	1,460.48	1,907.57	2,414.27	2,980.58
12	A	19.47	77.99	174.31	307.67	481.00	695.21	943.87	1,238.70	1,563.59	1,930.96
	E	19.51	78.05	175.62	312.21	487.82	702.47	956.14	1,248.83	1,580.55	1,951.30
14	A	14.70	58.03	130.02	231.99	358.25	518.15	706.40	928.63	1,169.08	1,439.50
	E	14.55	58.21	130.96	232.83	363.79	523.86	713.03	931.30	1,178.68	1,455.16
16	A	11.97	45.50	104.38	185.67	288.83	421.91	566.41	741.10	939.49	1,152.48
	E	11.66	46.62	104.90	186.48	291.38	419.58	571.10	745.92	944.06	1,165.50
18	A	8.79	35.36	81.36	142.77	226.35	324.84	440.02	577.60	727.79	897.03
	E	9.10	36.39	81.88	145.57	227.45	327.53	445.80	582.27	736.94	909.80
20	A	7.32	29.67	66.08	117.13	184.57	268.64	366.73	473.71	600.37	739.74
	E	7.47	29.90	67.27	119.60	186.87	269.09	366.26	478.38	605.46	747.48
22	A	5.89	24.29	55.43	98.18	154.06	220.50	302.93	397.98	497.97	616.13
	E	6.24	24.97	56.18	99.88	156.07	224.73	305.89	399.53	505.65	624.26
24	A	5.37	21.35	46.59	83.13	130.69	188.57	257.59	332.54	421.16	520.66
	E	5.25	21.02	47.29	84.08	131.37	189.18	257.49	336.31	425.65	525.49
26	A	4.48	18.46	39.93	72.89	111.70	160.12	220.20	289.07	362.45	445.75
	E	4.50	18.00	40.50	72.00	112.50	162.00	220.50	288.01	364.51	450.01
28	A	3.83	16.17	34.66	62.91	98.01	139.98	188.66	247.94	314.91	385.07
	E	3.91	15.62	35.16	62.50	97.66	140.62	191.41	250.00	316.40	390.62
30	A	3.75	13.46	31.18	54.20	84.54	122.32	165.51	215.03	274.48	338.57
	E	3.41	13.64	30.70	54.58	85.28	122.80	167.15	218.32	276.31	341.12
32	A	3.32	12.36	27.54	47.44	74.59	106.87	146.77	189.48	240.67	296.21
	E	3.01	12.03	27.07	48.13	75.20	108.29	147.39	192.51	243.64	300.79
34	A	3.00	11.17	24.48	43.16	66.04	98.14	132.69	169.61	215.17	263.24
	E	2.67	10.69	24.06	42.78	66.84	96.25	131.01	171.11	216.56	267.36

TABLE 6.4: Table showing the exact number of cycles of length m seen empirically (A), and the number of cycles of length m expected (E) to be seen. Actual data is averaged over 100 experiments.

m	Total Cycles	Self-Avoiding Cycles	Self-Intersecting Cycles
4	2	4	0
6	4	4	0
8	22	14	8
10	120	56	64
12	624	248	376
14	3,600	1,176	2,424
16	21,388	5,876	15,512
18	129,284	30,536	98,748
20	803,296	163,652	639,644
22	5,075,292	899,144	4,176,148
24	32,542,624	5,042,540	27,500,084
26	211,437,956	28,770,752	182,667,204
28	1,389,206,920	166,580,848	1,222,626,072
30	9,217,403,992	976,769,056	8,240,634,936
32	61,693,656,876	5,790,865,320	55,902,791,556
34	416,145,092,064	34,665,748,728	381,479,343,336

TABLE 6.5: Number of self avoiding cycle and self intersecting cycles of size m where $4 \leq m \leq 34$

the biological sciences and even to traffic models. It is the widespread applicability of these models to interesting phenomena that makes them so deserving of our attention. [101, pg. vii]

While the results so far do not show a formula cannot exist, it merely and rather optimistically has not been found yet. Further, should a formula be found how does this affect the problem in this work, such as could it through some manipulation allow for counting the self-intersecting cycles of length m and thus, the combination of the two, the counting of all possible cycles of length m. Also, vice versa, could a formula that counts all possible cycles help in solving the problem of calculating all possible self-avoiding cycles and thus solve the problem for SAP's that have great importance to other fields.

6.3 Graph Visualisation and Analysis

Another direction that has been looked at through the duration of this project is that of graph visualisation. Through the process of investigating the basic walk the problem of visualising graphs was considered to assist in identifying the correctness of the implemented algorithm and the analysing of the results. Due to the nature of the work with the basic walk exploring large graphs, the visualiser had to be capable of handling graphs of at least one million vertices.

The visualiser was built in Java with the intention of visualising specific graphs such as the 2D grid. However, through peer feedback and from general interest the visualiser was quickly expanded to allow for drawing of various graph topologies (i.e. 2D triangular grid, complete graph, wheel graph, and others.) as well as allowing for graphs to be manually drawn through the GUI to allow for a user to draw any graph that they had interest in. Further to this to allow for experimentation with real datasets a number of common data formats were added to allow for importing from the SNAP data set repository [126], Matrix Market [33], as well as more general formats such as Trivial Graph Format [185] and Graph Modelling Language [105].

With the addition of the data import formats it started to become more common to test real data which lead to the importance of identifying methods of visualising the data when no location information was present. In §6.3.1 some of the methods used for visualisation are discussed. Further once a good visualisation was possible it became necessary to analyse the graphs through the use of algorithms, a selection of algorithms used for this are discussed in §6.3.2. Through developing a good implementation of these two methods the software attracted industrial interest from Dollywagon Media Services Ltd which utilised these tools in the "Efficient Harvesting of Social Network Data"; a joint partnership between industry and the University of Liverpool. Further software has also attracted interest from Biological research and is currently central to the project, "Efficient Biological Networks Discovery and Analysis" [85], which is a Natural Environment Research Council funded project between the Department of Computer Science and the Institute of Integrative Biology, both of the University of Liverpool.

In the following sections discussions on the steps taken to reach the functionality that was of interest to Dollywagon and its clientèle are presented, and discussions on the experimental work conducted. Specifically as per the project title, interest branched over to looking specifically at social network data.

6.3.1 Graph Visualisation

While analysis of a graph can be performed without ever having a visual representation of it, it can be beneficial to see exactly what the results are indicating just as how a chart can reveal more than a table alone can.

One of the key interests in graph visualisation is in identifying methods of distributing the vertices within the viewing space such that the vertices do not overlap or obscure each other and also the number of edges which cross over each other are minimised. For instance, simply positioning vertices through the use of a PRNG could be tweaked to allow for the property of non-overlapping vertices to be met, however, it would be incredibly difficult and time consuming to position the vertices randomly with minimal edge crossings. However, deterministic algorithms can be the most suitable in some scenarios where the structure is easily defined, such as for example in a bipartite graphs, complete graphs, or tree structures. Though even still, for any arbitrary graph structural information alone may not provide any intuition as to how to position the data.

For instance, in data such as social network data it is common to find that there exists distinct groupings of vertices (cliques) that are separated by various bridging vertices, for example as can be seen in Figure 6.9. This leads to the issue that while a simple deterministic method is unsuitable, and further still randomised methods are also completely unsuited. This leads to the need for a method that will group the cluster but

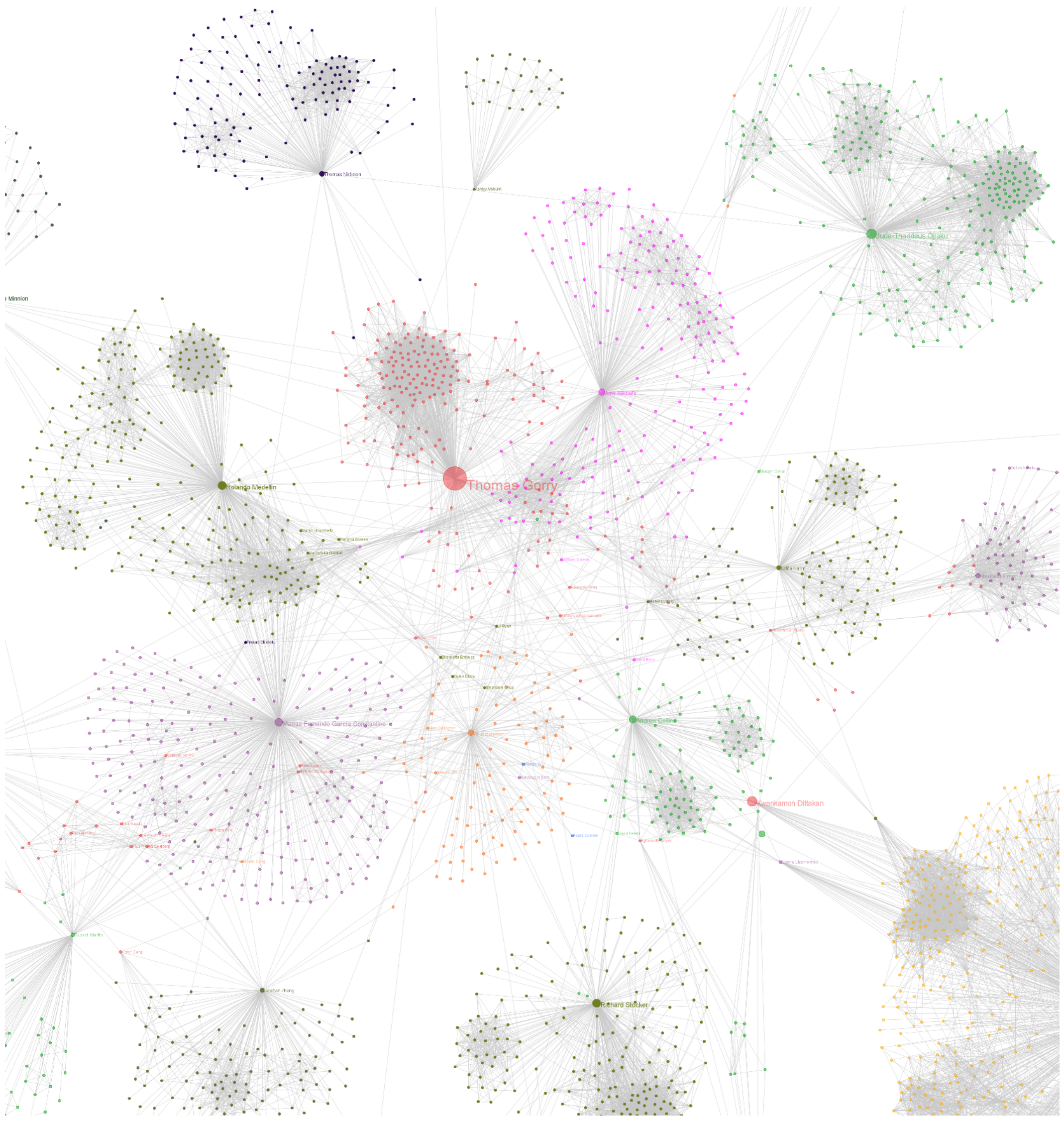

FIGURE 6.9: The combined Facebook friend relation data of several PhD students from the University of Liverpool Computer Science department.

separate each cluster from each other while still preserving locality. A common solution which was looked at is that of force directed layouts. This type of algorithm first came to prominence thanks to the work of Eades [76] though was later adapted and improved by Fruchterman and Reingold [93]. Since then there have been a number of papers on the topic providing a number of improvements in the complexity through different methods, for example: Holten and van Wijk [106] discusses the concept of edge bundling where by groupings of vertices are identified and then collapsed to reduce edge calculations and paths, Quigley [151, 152] considers grouping vertices by their location and then performing force direction on the groups to reduce the complexity dramatically so that it depends less on the number of vertices and more on the number groupings identified using a method such as quad trees.

However, due to time constraints this work adopted the simple, yet very effective Fruchterman and Reingold Algorithm, which is reproduced in Algorithm 10. Though while this algorithm is considerably simplistic in comparison to its successors, it provides a very good starting point for understanding how these algorithms work and why they are so interesting. The objective of these algorithms is to separate all vertices by some distance using the principle of Coulomb's inverse-square law to repulse each vertex from each other vertex so that no pair of vertices will be located near any other vertex. Secondly the algorithm combines this with the use of Hooke's Law to control the attraction of vertices. That being that any pair of connected vertices are attracted to each other at a force proportional to their distance. The resulting combination of these two techniques is that connected vertices will be located close to each other, and that unconnected vertices should be located at some distance, with the fortunate property that edge crossings will become minimised. In a social network the resulting graph should clearly show communities due to their high connectivity while at the same time ensuring that each community is placed in a position that reflects the connectivity between clusters, as can be seen in Figure 6.9.

Algorithm 10: ForceDirectedLayout(V, E)

```
for all $v \in V$ do
  for all $u \in V$ do
    $v = v - u$
  end for
end for
for all $v \in V$ do
  for all $u \in \Delta\ v$ do
    $v = v + u$
  end for
end for
for all $v \in V$ do
  $v = displacement_v$
end for
```

This algorithm proved to be highly effective for the needs of this work and also for the third parties that had taken interest in the software. While faster algorithms will of course permit for large datasets to be rendered, this algorithm, and specifically the slightly modified version of it which reduced the run-time by only performing V^2 repulsion loop when vertices were within close distance, permitted the rendering of graphs with up to 10,000 vertices to be rendered in a reasonable amount of time. For most of the use cases where a rendering tends to be required, a minimal amount of vertices that can be quickly looked at and understood is critical so, of which 10,000 exceeds this minimal limit by an order of magnitude. While inevitably there will be use cases where higher numbers of vertices would be interesting to see, and this algorithm was shown to successfully handle greater numbers, the time required to generate makes most cases infeasible and better algorithms would need to be adopted.

Finally what also should be mentioned in terms of visualisation, is that through good programming practice, real time interactivity with large graphs is possible with

GraphDraw. Specifically the software makes extensive use of high quality algorithms that require linear or logarithmic time complexity to minimise processing of graph elements that are not currently in the users view ports and makes extensive use of indexing to enable for interactivity with single elements. The result of this is that not only can a user visualise a reasonably or very large graph in a way that is appealing and intuitive, but also quickly interactive with the viewing area and focus on single elements without a noticeable delay.

6.3.2 Graph Analysis

The high quality of the algorithms that are core to GraphDraw's ability to render, index and search the loaded GraphDraw permit the user to analyse the graph with ease. In addition to visualising the graph, analysing the graph was also of interest to try to identify points of interest that should be looked at more thoroughly. To do this methods of analysing a graph using algorithms was considered. Specifically, GraphDraw currently implements the Page Rank algorithm [38], Betweenness Centrality algorithm [36, 91], Closeness Centrality [36, 153], Clustering Coefficient [175] and various other less common algorithms.

In Figure 6.9, one of the aforementioned algorithms, betweenness centrality, has been calculated for the graph, and then applied as the scaling for the vertices. Through having the visual component and also being able to calculate these metrics on the graph it is possible to easily see which vertices within the graph are significant for some reason. With combination with the work completed in the prior section this allows an end user to immediately focus on a significant vertex and discover its properties or the properties of its neighbours to identify why a given vertex might be significant. This has become greatly important to the users of GraphDraw as it has allowed the industrial partners to use the software as a means of discovering significant users or organisations that are in a social network, and in the biological context it has allowed the biologists to identify genes, proteins, or similar, to be identified that seem to be significant for *some reason* to the remainder of the network.

What has been learned from working with and applying these algorithms to hundreds of networks through the duration of this project is that there is no 'one size fits all' algorithm, and still, little is known about what an algorithm is actually discovering. While it is fully understood what each algorithm is discovering in a theoretical sense, how this applies to specific contexts is a completely different matter. For example, two algorithms can produce vastly different results, however, which one is more interesting and why. In Figure 6.10 and Figure 6.11 are two different algorithms that have been executed on a graph, and then applied as the vertex scaling. This data was collected during the retirement of Pope Benedict XVI and election of Pope Francis from the Twitter social network. What this graph shows is the interactivity between different Twitter users, each vertex is a Twitter user and any pair of vertices has an edge between them if the two users have interacted in some way such as through responding to the other persons message, retweeting it (forwarding it) or through simply messaging that user in one of their own messages. Firstly, when using the betweenness centrality algorithm some interesting users become highlighted such as the “Catholic News Svc”, “Opinionated Catholic”, and “A Radical Catholic”, as well as a number of perhaps unclear users such as “Rachel Zoll”, “James Toups”, “lukecoppen”, and “Joshua McElwee”. Through further investigation of these users, it can be learned that they are all journalists and religious correspondents or editors for media organisations, such the Associated Press, The Catholic Herald and the National Catholic Reporter. The only exception to this is James Toups who is neither and simply a prominent catholic follower who has a very large following.

Now contrast this with the Page Rank algorithm which seems to discover more familiar names such as “Pope Francis” himself, “The Assosicated Press”, “Catholic News Svc”, “Reuters”, “Huffington Post” and many others. Immediately it is quite clear to see that this algorithm is discovering a different set of users as being significant and in particular is discovering the “big media” agencies that millions of users follow and regularly communicate back with.

What this seems to show is that while the Betweenness Centrality algorithm seems to be discovering the information creators, they do not have enough direct influence or popularity to be directly noticed by the general network. However, it is these very people who are the content producers for the big media groups that Page Rank discovers. Both of these groups are interesting for different reasons and this shows different algorithms must be utilised depending on the specific goal of the analysis. If the content producers and the bottle necks of the network are required, then in this context it appears to be Betweenness Centrality that should be used, however, if it is the users that are actually influencing and interacting with the bulk of the network, then Page Rank should be utilised.

Finally, in closing to this section, it should be noted that none of the algorithms implemented and deployed have ever realistically been designed or considered for the uses in which they are now being used for and for that reason it is perhaps still too early to fully understand exactly what they are discovering in the new and unexpected types of data that is being experimented with. However, what can be commented on by the author is that similar results for these algorithms were seen in various different data sets and this could perhaps indicate that there is a need to fully experiment with these algorithms in a supervised manner and try to fully understand exactly what type of significant data these algorithms are discovering.

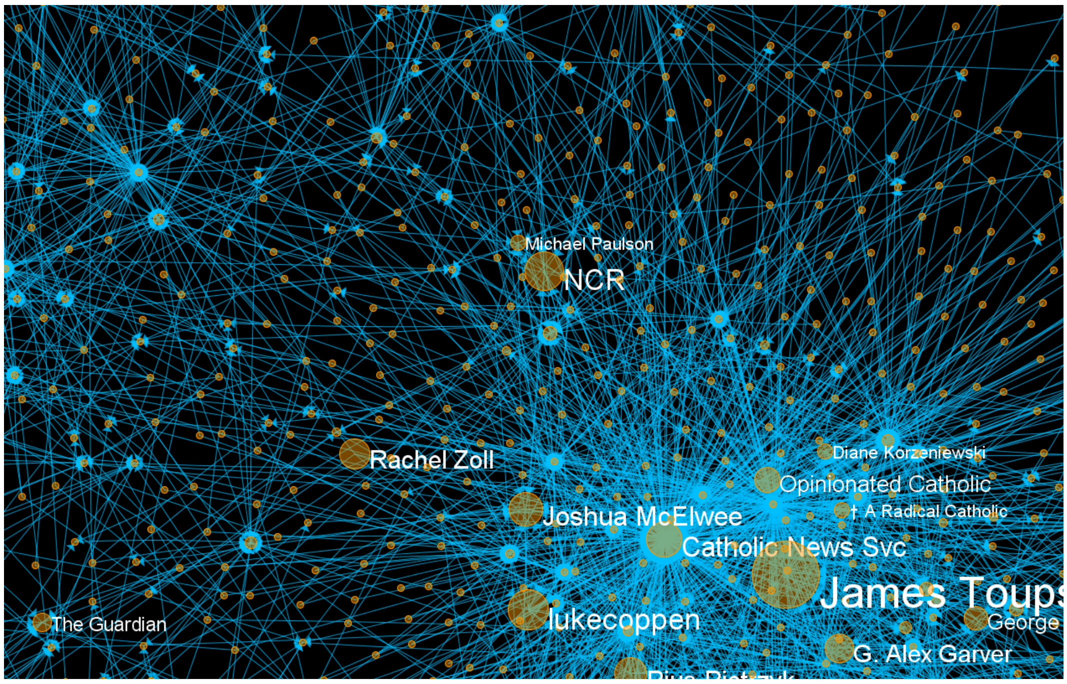

FIGURE 6.10: A network comprising of interactions between Twitter users interested in the Papal elections. Vertices are scaled by Betweenness Centrality.

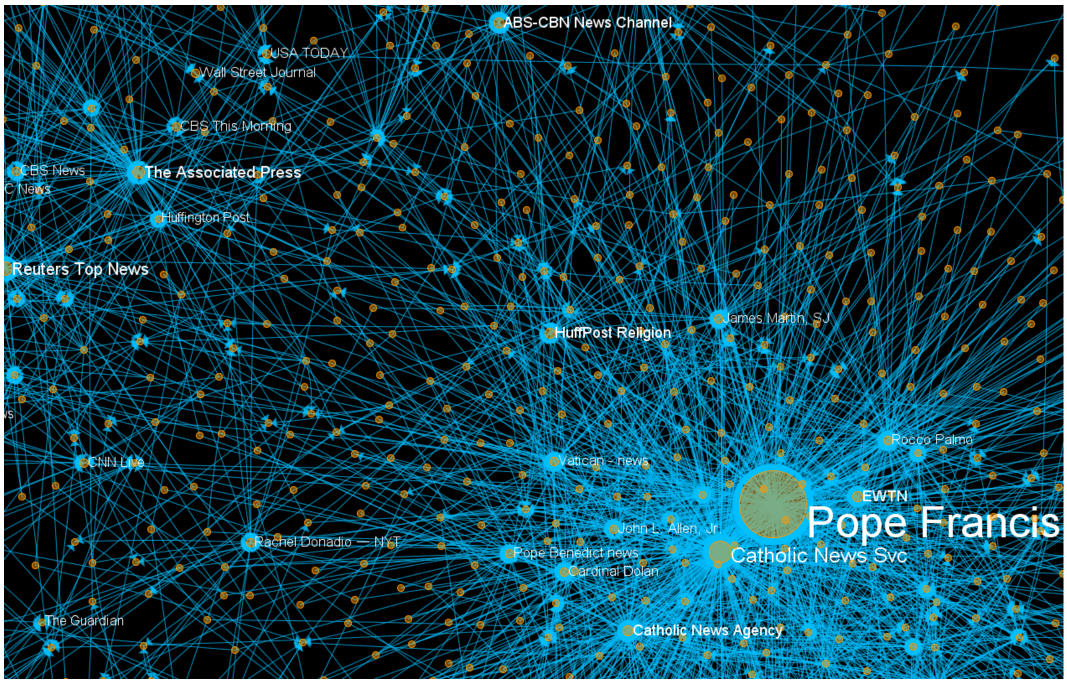

FIGURE 6.11: A network comprising of interactions between Twitter users interested in the Papal elections. Vertices are scaled by Page Rank.

Chapter 7

Conclusion

7.1 Conclusion and Further Work

In this thesis a number of solutions were presented for problems within the general area of the Rendezvous Problem and Network Patrolling. Specifically the algorithms present new time efficient solutions that are either the current, as of writing, state of the art, or have been superseded during the period of the production of this work by newer work that builds on the results in this thesis.

In Chapter 3, synchronous rendezvous was discussed in various environments. Firstly an algorithm was shown that would allow rendezvous to occur for two agents in the infinite line in linear time. Later the problem of rendezvous in trees was shown, to solve this firstly an algorithm for permitting rendezvous in $\mathcal{O}(d)$ was shown for the half-line, where one side of the line is fixed at the origin and the other side extends towards ∞. In this structure the agents meet by traversing the structure in one direction, towards the origin. The algorithm shown to achieve this is then further extended to allow for rendezvous to be performed in linear time in tree structures. Finally Chapter 3 discusses higher dimensional space and arbitrary graphs; it is shown that in this environment rendezvous can be achieved in linear time through the partitioning of the space and establishing a route between meeting points within the partitions.

In future work in synchronous rendezvous there are a number of potential directions that can be taken. Firstly, the algorithms that were identified, while linear and near optimal, were affected by some small constant. It would be interesting to see if more efficient algorithms can be produced to reduce these constants further. Further, while a number of structures are considered, other specific structures could be considered.

In Chapter 4, asynchronous rendezvous was considered and an algorithm was shown that would permit rendezvous to occur in a near optimal time for 2D spaces. This algorithm showcased the concept of space covering sequences in regards to the rendezvous problem.

Since completed, this work was superseded by [25] which managed to apply the same techniques developed in the results produced for this thesis but further reduce the overall cost of rendezvous to $\mathcal{O}(d^2 \text{ polylog } d)$ by producing a more efficient space covering sequence through pruning of the hierarchical tree structure . However, while this work does provide a near optimal result, further improvements could possibly still be made. Also, both of these works focused solely on the 2D space which further leads to the future direction in applying these techniques (or altered techniques) to other environments in hope of also finding similar optimal solutions.

In Chapter 5, a slightly different problem was introduced though still in a similar direction as the previous chapters, and that is the topic of network patrolling. In this chapter, an algorithm that will discover a traversal sequence for k agents that minimises the amount of time in which a vital part of the ring is idle. In this work an algorithm was discovered that requires the time complexity of $\mathcal{O}(kn \log n)$ to discover minimal idleness trajectories, of which this cost is optimal. As per the prior chapters, once again in this chapter, as a direction for possible future work the methodologies developed in this work should be considered for other environments to see if this or similar algorithms can be applied. For instance, one possible avenue is that of the infinite line, while for a finite k agents it would be impossible for an infinite line to be solvable (as an agent beginning a walk would never return, and thus never *patrol*), however, fixing some aspects of the infinite line may allow for some solutions to be developed.

Overall, in Chapter 3, 4, and 5 the algorithms shown are presented primarily with rendezvous for two agents in mind, however, it is reasonable to see that some of these algorithms may extend to three or more agents with either no or minor modifications. As no discussion, or proofs are provided in this work for more than two agents, it would be the responsibility of a future work to develop on this idea and see if the proofs in this thesis can be extended or whether completely new algorithms are required to be found.

Finally Chapter 6, three topics were discussed by the author in relation to experimental and exploratory works.

Firstly the topic of parasitic computing was discussed. This topic showcased a model and opened discussions for possible applications that could be developed using this model, such as large scale computation of data through, revenue generation and security for online resources. An application of this model was submitted and accepted by the Biotechnology and Biological Sciences Research Council special funding call "Crowd sourcing for the biological sciences". Specifically this project will allow for the construction of a system that will provide a potentially massive amount of computational power for the Human Proteome Project.

The second topic discussed was that of investigations into the basic walk method. This work confirmed previous weaker works and discovered new phenomena within the model. Further this work lead to the discovery of a new formula for calculating the probable number of cycles within a grid and also discovered all possible cycles that can be constructed in n^2 grids up to a length of 34, the highest as of writing. Also, in this work experimental results hinted possible models that could be built using this concept due to the low eccentricity of cycles. In future work, there are number of possible directions such as proving the reasoning for the discovered phenomena of longest and average cycle length, discovering better algorithms for discovering all possible cycles of a given length and further testing whether the model can be used in the application of network routing and in the application of boundary discovery.

Finally, the software developed during this project “GraphDraw” was showcased. The work in this project has built on many established algorithms and made them all available in one place. Since GraphDraw’s initial development it has gained academic and industrial interest due to its ability to process large data and due to the quality of its implementation. This project has great potential and is already receiving both financial support (through the Natural Environment Research Council and through industry) as well additional development support with a number of bachelor, master and doctoral level projects being launched that will continue to add more algorithms for visualising data, analysing data and integrating the software with other tools. While at the time of writing, no algorithms have been developed specifically by the author or those involved in the project but if support continues at the current rate it would come as no surprise to the author that this project will directly lead to assisting in discovering new algorithms for computer scientists and provide a means in which other fields discover new results within their own data.

Appendix A

Summary of all cycles up to length 34

A.1 Definitions

This chapter contains all of the results generated for cycles $n = \{4, \ldots, 34\}$.

To read the following tables, each section contains the result for a given n. Each row defines a cycle classification based on the number of vertices repeated within it's cycle. Specifically, for any cycle c the vertices within it can be identified as v_x where x is the number of times that vertex is visited by the same cycle. A vertex that is contained within a cycle can visited a minimum of one to a maximum of four when the basic walk is ran on the 2D Grid.

To read a row, the frequency column states how many cycles of that classification exists, the total vertices column defines how many vertices were in that given cycle and then c_x shows the the frequency of cycles with that type. For example $n = 8$ has a cycle with 7 total vertices of which one of the vertices is crossed once, there are eight cycles of this type. The Pr value is calculated from performing the summation in Equation 6.1 using the data below.

Finally, as of writing, these tables are believed to be the only attempt at generating these values, and consequently so far contain the largest n value.

A.2 Cycle length 4

Frequency	Total Vertices	v_1	v_2	v_3	v_4
2	4	4	0	0	0

TABLE A.1: Cycle length 4. 2 discovered. $Pr = 0.024691358024691$

A.3 Cycle length 6

Frequency	Total Vertices	v_1	v_2	v_3	v_4
4	6	6	0	0	0

TABLE A.2: Cycle length 6. 4 discovered. $Pr = 0.0054869684499314$

A.4 Cycle length 8

Frequency	Total Vertices	v_1	v_2	v_3	v_4
8	7	6	1	0	0
14	8	8	0	0	0

TABLE A.3: Cycle length 8. 22 discovered. $Pr = 0.0039628105471727$

A.5 Cycle length 10

Frequency	Total Vertices	v_1	v_2	v_3	v_4
32	8	6	2	0	0
32	9	8	1	0	0
56	10	10	0	0	0

TABLE A.4: Cycle length 10. 120 discovered. $Pr = 0.0029805754542837$

A.6 Cycle length 12

Frequency	Total Vertices	v_1	v_2	v_3	v_4
8	8	5	2	1	0
208	10	8	2	0	0
160	11	10	1	0	0
248	12	12	0	0	0

TABLE A.5: Cycle length 12. 624 discovered. $Pr = 0.0019512984508158$

A.7 Cycle length 14

Frequency	Total Vertices	v_1	v_2	v_3	v_4
64	10	6	4	0	0
48	10	7	2	1	0
352	11	8	3	0	0
1128	12	10	2	0	0
832	13	12	1	0	0
1176	14	14	0	0	0

TABLE A.6: Cycle length 14. 3,600 discovered. $Pr = 0.0014551631005762$

A.8 Cycle length 16

Frequency	Total Vertices	v_1	v_2	v_3	v_4
32	10	5	4	1	0
64	11	6	5	0	0
144	11	7	3	1	0
1104	12	8	4	0	0
432	12	9	2	1	0
3008	13	10	3	0	0
6248	14	12	2	0	0
4480	15	14	1	0	0
5876	16	16	0	0	0

TABLE A.7: Cycle length 16. 21,388 discovered. $Pr = 0.0011655010842754$

A.9 Cycle length 18

Frequency	Total Vertices	v_1	v_2	v_3	v_4
32	11	5	5	1	0
20	12	6	6	0	0
640	12	7	4	1	0
128	12	8	2	2	0
1472	13	8	5	0	0
1152	13	9	3	1	0
11600	14	10	4	0	0
2480	14	11	2	1	0
20928	15	12	3	0	0
35496	16	14	2	0	0
24800	17	16	1	0	0
30536	18	18	0	0	0

TABLE A.8: Cycle length 18. 129,284 discovered. $Pr = 0.00090980297250102$

A.10 Cycle length 20

Frequency	Total Vertices	v_1	v_2	v_3	v_4
48	11	5	4	1	1
64	12	5	6	1	0

80	12	6	4	2	0
48	12	7	2	3	0
12	12	8	0	4	0
512	13	6	7	0	0
864	13	7	5	1	0
128	13	8	3	2	0
4104	14	8	6	0	0
7888	14	9	4	1	0
896	14	10	2	2	0
25160	15	10	5	0	0
10208	15	11	3	1	0
91600	16	12	4	0	0
13728	16	13	2	1	0
137376	17	14	3	0	0
205984	18	16	2	0	0
140944	19	18	1	0	0
163652	20	20	0	0	0

TABLE A.9: Cycle length 20. 803,296 discovered. $Pr = 0.0007474755606491$

A.11 Cycle length 22

Frequency	Total Vertices	v_1	v_2	v_3	v_4
16	12	4	6	2	0
448	13	6	5	2	0
384	13	7	4	1	1
608	14	6	8	0	0
3760	14	7	6	1	0
2920	14	8	4	2	0
544	14	9	2	3	0
48	14	10	0	4	0
8992	15	8	7	0	0
20256	15	9	5	1	0
3456	15	10	3	2	0
72364	16	10	6	0	0
69920	16	11	4	1	0
5696	16	12	2	2	0
260784	17	12	5	0	0
70464	17	13	3	1	0
657232	18	14	4	0	0
75552	18	15	2	1	0
887008	19	16	3	0	0

1217040	20	18	2	0	0
818656	21	20	1	0	0
899144	22	22	0	0	0

TABLE A.10: Cycle length 22. 5,075,292 discovered. $Pr = 0.00062426369979494$

A.12 Cycle length 24

Frequency	Total Vertices	v_1	v_2	v_3	v_4
96	13	5	6	1	1
256	14	5	8	1	0
648	14	6	6	2	0
192	14	6	7	0	1
832	14	7	4	3	0
480	14	7	5	1	1
96	14	8	2	4	0
128	15	6	9	0	0
4488	15	7	7	1	0
9448	15	8	5	2	0
352	15	8	6	0	1
1824	15	9	3	3	0
2064	15	9	4	1	1
192	15	10	1	4	0
20896	16	8	8	0	0
67312	16	9	6	1	0
35788	16	10	4	2	0
1216	16	10	5	0	1
4848	16	11	2	3	0
216	16	12	0	4	0
159776	17	10	7	0	0
229664	17	11	5	1	0
29696	17	12	3	2	0
208	17	12	4	0	1
870272	18	12	6	0	0
538288	18	13	4	1	0
33856	18	14	2	2	0
2232408	19	14	5	0	0
457296	19	15	3	1	0
4528464	20	16	4	0	0
420128	20	17	2	1	0
5698144	21	18	3	0	0
7308944	22	20	2	0	0

4841568	23	22	1	0	0
5042540	24	24	0	0	0

Table A.11: Cycle length 24. 32,542,624 discovered. $Pr = 0.00052549119764244$

A.13 Cycle length 26

Frequency	Total Vertices	v_1	v_2	v_3	v_4
368	14	5	6	3	0
192	14	5	7	1	1
48	14	6	4	4	0
160	15	5	9	1	0
1248	15	6	7	2	0
48	15	6	8	0	1
2048	15	7	5	3	0
2976	15	7	6	1	1
320	15	8	3	4	0
1536	15	8	4	2	1
4048	16	6	10	0	0
15600	16	7	8	1	0
31088	16	8	6	2	0
2688	16	8	7	0	1
16704	16	9	4	3	0
7808	16	9	5	1	1
3200	16	10	2	4	0
42368	17	8	9	0	0
160496	17	9	7	1	0
126784	17	10	5	2	0
6176	17	10	6	0	1
19776	17	11	3	3	0
11344	17	11	4	1	1
1152	17	12	1	4	0
446472	18	10	8	0	0
941952	18	11	6	1	0
319856	18	12	4	2	0
9888	18	12	5	0	1
31808	18	13	2	3	0
1056	18	14	0	4	0
2276928	19	12	7	0	0
2129664	19	13	5	1	0
216256	19	14	3	2	0
1984	19	14	4	0	1

8272892	20	14	6	0	0
3765808	20	15	4	1	0
198416	20	16	2	2	0
17417168	21	16	5	0	0
2887488	21	17	3	1	0
30702016	22	18	4	0	0
2373600	22	19	2	1	0
36632128	23	20	3	0	0
44516352	24	22	2	0	0
29067296	25	24	1	0	0
28770752	26	26	0	0	0

TABLE A.12: Cycle length 26. 211,437,956 discovered. $Pr = 0.00045000966789279$

A.14 Cycle length 28

Frequency	Total Vertices	v_1	v_2	v_3	v_4
288	14	5	6	1	2
336	14	6	4	2	2
8	15	4	9	2	0
208	15	5	7	3	0
192	15	5	8	1	1
352	15	6	6	2	1
864	15	7	4	3	1
576	15	8	2	4	1
172	16	4	12	0	0
432	16	5	10	1	0
6656	16	6	8	2	0
224	16	6	9	0	1
13800	16	7	6	3	0
4672	16	7	7	1	1
4720	16	8	4	4	0
2816	16	8	5	2	1
352	16	9	2	5	0
2752	17	6	11	0	0
27072	17	7	9	1	0
85728	17	8	7	2	0
3600	17	8	8	0	1
66368	17	9	5	3	0
40928	17	9	6	1	1
6528	17	10	3	4	0
14912	17	10	4	2	1

120336	18	8	10	0	0
487248	18	9	8	1	0
636720	18	10	6	2	0
29248	18	10	7	0	1
210144	18	11	4	3	0
62496	18	11	5	1	1
30176	18	12	2	4	0
1041264	19	10	9	0	0
2722616	19	11	7	1	0
1393600	19	12	5	2	0
72080	19	12	6	0	1
172416	19	13	3	3	0
65328	19	13	4	1	1
7456	19	14	1	4	0
6650456	20	12	8	0	0
9747248	20	13	6	1	0
2418280	20	14	4	2	0
71520	20	14	5	0	1
189072	20	15	2	3	0
5524	20	16	0	4	0
25439608	21	14	7	0	0
17331520	21	15	5	1	0
1459776	21	16	3	2	0
16528	21	16	4	0	1
70257616	22	16	6	0	0
25244848	22	17	4	1	0
1165824	22	18	2	2	0
129371272	23	18	5	0	0
18040832	23	19	3	1	0
206663728	24	20	4	0	0
13639104	24	21	2	1	0
236392128	25	22	3	0	0
274448352	26	24	2	0	0
176737152	27	26	1	0	0
166580848	28	28	0	0	0

Table A.13: Cycle length 28. 1,389,206,920 discovered. $Pr = 0.00039062296525159$

A.15 Cycle length 30

Frequency	Total Vertices	v_1	v_2	v_3	v_4

96	15	4	8	2	1
240	16	4	10	2	0
448	16	5	8	3	0
176	16	5	9	1	1
2080	16	6	6	4	0
2688	16	6	7	2	1
80	16	6	8	0	2
640	16	7	4	5	0
2432	16	7	5	3	1
2880	16	7	6	1	2
2688	16	8	4	2	2
960	17	5	11	1	0
9280	17	6	9	2	0
960	17	6	10	0	1
24448	17	7	7	3	0
13232	17	7	8	1	1
17504	17	8	5	4	0
24048	17	8	6	2	1
1792	17	9	3	5	0
8992	17	9	4	3	1
3456	17	10	2	4	1
7972	18	6	12	0	0
58624	18	7	10	1	0
274912	18	8	8	2	0
18880	18	8	9	0	1
325040	18	9	6	3	0
103648	18	9	7	1	1
118864	18	10	4	4	0
61504	18	10	5	2	1
2304	18	10	6	0	2
5952	18	11	2	5	0
768	18	12	0	6	0
203136	19	8	11	0	0
1036128	19	9	9	1	0
1894560	19	10	7	2	0
96160	19	10	8	0	1
959008	19	11	5	3	0
394496	19	11	6	1	1
109184	19	12	3	4	0
94528	19	12	4	2	1
2745964	20	10	10	0	0

8846944	20	11	8	1	0
7612352	20	12	6	2	0
362784	20	12	7	0	1
1904736	20	13	4	3	0
442848	20	13	5	1	1
205376	20	14	2	4	0
17734832	21	12	9	0	0
33581008	21	13	7	1	0
12444416	21	14	5	2	0
577728	21	14	6	0	1
1265568	21	15	3	3	0
383632	21	15	4	1	1
48512	21	16	1	4	0
81268048	22	14	8	0	0
87056368	22	15	6	1	0
16914616	22	16	4	2	0
475168	22	16	5	0	1
1094912	22	17	2	3	0
31024	22	18	0	4	0
242996208	23	16	7	0	0
130570784	23	17	5	1	0
9517184	23	18	3	2	0
127744	23	18	4	0	1
559207432	24	18	6	0	0
165901504	24	19	4	1	0
6915008	24	20	2	2	0
935509168	25	20	5	0	0
112514336	25	21	3	1	0
1387758112	26	22	4	0	0
79630368	26	23	2	1	0
1532949952	27	24	3	0	0
1709896120	28	26	2	0	0
1086259392	29	28	1	0	0
976769056	30	30	0	0	0

TABLE A.14: Cycle length 30. 9,217,403,992 discovered. $Pr = 0.0003411190501216$

A.16 Cycle length 32

Frequency	Total Vertices	v_1	v_2	v_3	v_4
232	16	4	8	4	0
128	16	4	9	2	1

576	16	5	8	1	2
448	16	6	6	2	2
1056	17	5	9	3	0
960	17	5	10	1	1
6080	17	6	7	4	0
5200	17	6	8	2	1
1152	17	6	9	0	2
320	17	7	5	5	0
11200	17	7	6	3	1
3488	17	7	7	1	2
8704	17	8	4	4	1
4032	17	8	5	2	2
3984	18	5	12	1	0
25632	18	6	10	2	0
800	18	6	11	0	1
93664	18	7	8	3	0
25328	18	7	9	1	1
121456	18	8	6	4	0
84896	18	8	7	2	1
1936	18	8	8	0	2
32992	18	9	4	5	0
64704	18	9	5	3	1
18016	18	9	6	1	2
2432	18	10	2	6	0
9216	18	10	3	4	1
14112	18	10	4	2	2
21984	19	6	13	0	0
158400	19	7	11	1	0
624704	19	8	9	2	0
39296	19	8	10	0	1
970448	19	9	7	3	0
351216	19	9	8	1	1
442416	19	10	5	4	0
454976	19	10	6	2	1
4928	19	10	7	0	2
44784	19	11	3	5	0
106272	19	11	4	3	1
16704	19	12	2	4	1
504380	20	8	12	0	0
2949392	20	9	10	1	0
6743168	20	10	8	2	0

371456	20	10	9	0	1
5062616	20	11	6	3	0
1488384	20	11	7	1	1
1354240	20	12	4	4	0
645120	20	12	5	2	1
20896	20	12	6	0	2
67584	20	13	2	5	0
13568	20	13	3	3	1
5376	20	14	0	6	0
6546560	21	10	11	0	0
24123760	21	11	9	1	0
28109280	21	12	7	2	0
1340168	21	12	8	0	1
9810384	21	13	5	3	0
3144512	21	13	6	1	1
1061056	21	14	3	4	0
569600	21	14	4	2	1
4864	21	15	2	3	1
49338456	22	12	10	0	0
121448992	22	13	8	1	0
74202920	22	14	6	2	0
3394240	22	14	7	0	1
14662976	22	15	4	3	0
2947456	22	15	5	1	1
1278912	22	16	2	4	0
238120800	23	14	9	0	0
341308616	23	15	7	1	0
99028240	23	16	5	2	0
4173712	23	16	6	0	1
8525680	23	17	3	3	0
2265792	23	17	4	1	1
304352	23	18	1	4	0
848282144	24	16	8	0	0
708530944	24	17	6	1	0
113894688	24	18	4	2	0
3076384	24	18	5	0	1
6331232	24	19	2	3	0
181928	24	20	0	4	0
2119101632	25	18	7	0	0
940999680	25	19	5	1	0
61114368	25	20	3	2	0

939680	25	20	4	0	1
4280983120	26	20	6	0	0
1081015360	26	21	4	1	0
41482336	26	22	2	2	0
6660111240	27	22	5	0	0
703747360	27	23	3	1	0
9320678144	28	24	4	0	0
471664464	28	25	2	1	0
9992462976	29	26	3	0	0
10751026008	30	28	2	0	0
6738451488	31	30	1	0	0
5790865320	32	32	0	0	0

Table A.15: Cycle length 32. 61,693,656,876 discovered. $Pr = 0.00030079203258124$

A.17 Cycle length 34

Frequency	Total Vertices	v_1	v_2	v_3	v_4
224	16	4	8	2	2
1488	17	5	8	3	1
576	17	5	9	1	2
3072	17	6	6	4	1
3264	17	6	7	2	2
800	17	7	4	5	1
112	18	4	12	2	0
6336	18	5	10	3	0
1408	18	5	11	1	1
20720	18	6	8	4	0
13696	18	6	9	2	1
464	18	6	10	0	2
17792	18	7	6	5	0
27904	18	7	7	3	1
21088	18	7	8	1	2
7680	18	8	4	6	0
23040	18	8	5	4	1
37760	18	8	6	2	2
416	18	9	2	7	0
1152	18	9	3	5	1
10752	18	9	4	3	2
4608	19	5	13	1	0
42096	19	6	11	2	0
480	19	6	12	0	1

229600	19	7	9	3	0
75808	19	7	10	1	1
355904	19	8	7	4	0
235136	19	8	8	2	1
16832	19	8	9	0	2
138560	19	9	5	5	0
367360	19	9	6	3	1
62400	19	9	7	1	2
7296	19	10	3	6	0
149344	19	10	4	4	1
45696	19	10	5	2	2
8064	19	11	2	5	1
23728	20	6	14	0	0
392352	20	7	12	1	0
1862456	20	8	10	2	0
70608	20	8	11	0	1
3860160	20	9	8	3	0
889696	20	9	9	1	1
2871552	20	10	6	4	0
1649984	20	10	7	2	1
37728	20	10	8	0	2
596000	20	11	4	5	0
760832	20	11	5	3	1
106576	20	11	6	1	2
50432	20	12	2	6	0
112704	20	12	3	4	1
68096	20	12	4	2	2
1210784	21	8	13	0	0
6949360	21	9	11	1	0
19519616	21	10	9	2	0
966528	21	10	10	0	1
19655904	21	11	7	3	0
5486176	21	11	8	1	1
6493376	21	12	5	4	0
4922480	21	12	6	2	1
98368	21	12	7	0	2
483680	21	13	3	5	0
875296	21	13	4	3	1
13824	21	14	1	6	0
83520	21	14	2	4	1
16027868	22	10	12	0	0

71535232	22	11	10	1	0
111816416	22	12	8	2	0
5356800	22	12	9	0	1
59391200	22	13	6	3	0
14787968	22	13	7	1	1
12170352	22	14	4	4	0
4871264	22	14	5	2	1
149552	22	14	6	0	2
521184	22	15	2	5	0
138496	22	15	3	3	1
29184	22	16	0	6	0
129814592	23	12	11	0	0
378956896	23	13	9	1	0
316767776	23	14	7	2	0
15279032	23	14	8	0	1
85578048	23	15	5	3	0
23160624	23	15	6	1	1
8235520	23	16	3	4	0
3418432	23	16	4	2	1
62528	23	17	2	3	1
714529488	24	14	10	0	0
1358051568	24	15	8	1	0
635597648	24	16	6	2	0
27283104	24	16	7	0	1
103726368	24	17	4	3	0
19123648	24	17	5	1	1
7799360	24	18	2	4	0
2729329920	25	16	9	0	0
3083885312	25	17	7	1	0
736455264	25	18	5	2	0
28697600	25	18	6	0	1
55211392	25	19	3	3	0
13453648	25	19	4	1	1
1892096	25	20	1	4	0
7970539488	26	18	8	0	0
5446132992	26	19	6	1	0
752461240	26	20	4	2	0
19786400	26	20	5	0	1
36864928	26	21	2	3	0
1088144	26	22	0	4	0
17447219520	27	20	7	0	0

6608062592	27	21	5	1	0
390536960	27	22	3	2	0
6681440	27	22	4	0	1
31985403668	28	22	6	0	0
7026159568	28	23	4	1	0
251616064	28	24	2	2	0
46986191920	29	24	5	0	0
4424108096	29	25	3	1	0
62697759616	30	26	4	0	0
2829942864	30	27	2	1	0
65466410112	31	28	3	0	0
68139566944	32	30	2	0	0
42137654656	33	32	1	0	0
34665748728	34	34	0	0	0

TABLE A.16: Cycle length 34. 416,145,092,064 discovered. $Pr = 0.00026736342673897$

Bibliography

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: a locality aware location service for mobile ad hoc networks. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, DIALM-POMC 2004, pages 75–84, New York, NY, USA, 2004. ACM. ISBN 1-58113-921-7. doi: 10.1145/1022630.1022643.

[2] A. Aggarwal. *The Art Gallery Theorem and Algorithm.* PhD thesis, Johns Hopkins University, 1984.

[3] N. Agmon, S. Kraus, and G. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, pages 2339 –2345, may 2008. doi: 10.1109/ROBOT.2008.4543563.

[4] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In A. L. C. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence - SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, pages 474–483. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23237-7. doi: 10.1007/978-3-540-28645-5_48.

[5] S. Alpern. Hide and seek games. Institut für Höhere Studien, July 1976. Seminar.

[6] S. Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3): 673–683, May 1995. ISSN 0363-0129. doi: 10.1137/S0363012993249195.

[7] S. Alpern. Rendezvous search: A personal perspective. *Operations Research*, 50(5):772–795, 2002. ISSN 0030364X.

[8] S. Alpern. Bilateral street searching in Manhattan (line-of-sight rendezvous on a planar lattice). Technical Report LSE-CDAM-2004-09, Centre for Discrete and Applicable Mathematics, London School of Economics, 2004.

[9] S. Alpern and S. Gal. Rendezvous search on the line with distinguishable players. *SIAM Journal on Control and Optimization*, 33(4):1270–1276, 1995. ISSN 0363-0129. doi: 10.1137/S0363012993260288.

[10] S. Alpern and S. Gal. *The Theory of Search Games and Rendezvous*, volume 55 of *International Series in Operations Research & Management Science.* Springer, 2003. ISBN 978-0-7923-7468-8.

[11] S. Alpern, V. J. Baston, and S. Essegaier. Rendezvous search on a graph. *Journal of Applied Probability*, 36(1):pp. 223–231, 1999. ISSN 00219002.

[12] S. Alpern, R. Fokkink, L. Gąsieniec, R. Lindelauf, and V. S. Subrahmanian, editors. *Search Theory: A Game Theoretic Perspective.* Springer New York, 2013. ISBN 978-1-4614-6824-0. doi: 10.1007/978-1-4614-6825-7.

[13] A. Alshukri, F. Coenen, and M. Zito. Web-site boundary detection using incremental random walk clustering. In M. Bramer, M. Petridis, and L. Nolle, editors, *Research and Development in Intelligent Systems XXVIII*, pages 255–268. Springer London, 2011. ISBN 978-1-4471-2317-0. doi: 10.1007/978-1-4471-2318-7_20.

[14] E. J. Anderson and S. Essegaier. Rendezvous search on the line with indistinguishable players. *SIAM Journal on Control and Optimization*, 33(6):1637–1642, Nov. 1995. ISSN 0363-0129. doi: 10.1137/S0363012993260707.

[15] E. J. Anderson and S. P. Fekete. Asymmetric rendezvous on the plane. In *Proceedings of the fourteenth annual symposium on Computational geometry*, 14th Annual ACM Symposium on Computational Geometry, pages 365–373, New York, NY, USA, 1998. ACM. ISBN 0-89791-973-4. doi: 10.1145/276884.276925.

[16] E. J. Anderson and S. P. Fekete. Two dimensional rendezvous search. *Operations Research*, 49(1): 107–118, 2001. ISSN 0030364X.

[17] E. J. Anderson and R. R. Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability*, 28:839–851, 1990.

[18] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, oct 1999. ISSN 1042-296X. doi: 10.1109/70.795787.

[19] Apache Software Foundation. Apache hadoop, 2013. URL `http://hadoop.apache.org`.

[20] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Minimum-link watchman tours. *Information Processing Letters*, 86(4):203–207, May 2003. ISSN 0020-0190. doi: 10.1016/S0020-0190(02) 00502-1.

[21] D. Auber and P. Mary. Tulip. http://tulip.labri.fr, 2013.

[22] D. H. Bailey, P. Borwein, and S. Plouffe. On the rapid computation of various polylogarithmic constants. *Mathematics of Computation*, 66(218):903–913, 1997. ISSN 0025-5718. doi: http://dx.doi.org/10.1090/S0025-5718-97-00856-9.

[23] E. Bampas, L. Gąsieniec, N. Hanusse, D. Ilcinkas, R. Klasing, and A. Kosowski. Euler tour lock-in problem in the rotor-router model. In I. Keidar, editor, *DISC*, volume 5805 of *Lecture Notes in Computer Science*, pages 423–435. Springer, 2009. ISBN 978-3-642-04354-3. doi: 10.1007/978-3-642-04355-0_44.

[24] E. Bampas, L. Gąsieniec, R. Klasing, A. Kosowski, and T. Radzik. Robustness of the rotor-router mechanism. In *OPODIS*, pages 345–358, 2009. doi: 10.1007/978-3-642-10877-8_27.

[25] E. Bampas, J. Czyzowicz, L. Gąsieniec, D. Ilcinkas, and A. Labourel. Almost optimal asynchronous rendezvous in infinite multidimensional grids. In *Proceedings of the 24th international conference on Distributed computing*, 24th international conference on Distributed computing, pages 297–311, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15762-9, 978-3-642-15762-2.

[26] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro. Rendezvous and election of mobile agents: Impact of sense of direction. *Theory of Computing Systems*, 40(2):143–162, 2007. ISSN 1432-4350. doi: 10.1007/s00224-005-1223-5.

[27] V. Baston. Note: Two rendezvous search problems on the line. *Naval Research Logistics*, 46(3): 335–340, 1999. ISSN 1520-6750. doi: 10.1002/(SICI)1520-6750(199904)46:3<335::AID-NAV6>3.0.CO;2-Q.

[28] V. Baston and S. Gal. Rendezvous on the line when the players' initial distance is given by an unknown probability distribution. *SIAM Journal on Control and Optimization*, 36(6):1880–1889, 1998. ISSN 0363-0129. doi: 10.1137/S0363012996314130.

[29] V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Research Logistics*, 48(8):722–731, 2001. ISSN 1520-6750. doi: 10.1002/nav.1044.

[30] F. Bellard. Computation of 2700 billion decimal digits of pi using a desktop computer, 2010. URL `http://bellard.org/pi/pi2700e9/pipcrecord.pdf`.

[31] J. Blom, G. Chittaranjan, and D. Gatica-Perez. Mining large-scale smartphone data for personality studies. *Personal and Ubiquitous Computing*, 17(3):433–450, 2013. ISSN 1617-4909. doi: 10.1007/s00779-011-0490-1.

[32] F. Boekhorst, H. Kamerman, and R. Zane. From big bang to big data: ASTRON and IBM collaborate to explore origins of the universe, 2012. URL `http://www.ibm.com/press/us/en/pressrelease/37361.wss`.

[33] R. F. Boisvert, R. Pozo, and K. A. Remington. The matrix market exchange formats: Initial design. Technical Report NISTIR 5935, National Institute of Standards and Technology, 2013.

[34] B. Bollobás. *Extremal Graph Theory.* Dover Books on Mathematics Series. Dover, 1978. ISBN 9780486435961.

[35] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7:609–616, 2001. ISSN 1022-0038. doi: 10.1023/A:1012319418150.

[36] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[37] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23:242–251, 1976.

[38] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Seventh International World-Wide Web Conference*, pages 107–117, Amsterdam, The Netherlands, 1998. Elsevier Science Publishers B. V.

[39] I. F. Brunell. Cycles of tours in a directed graph. Master's dissertation, University of Liverpool, 2008.

[40] K. Buchin. Constructing delaunay triangulations along space-filling curves. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 119–130. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04127-3. doi: 10.1007/978-3-642-04128-0_11.

[41] B. Cai, H. Wang, H. Zheng, and H. Wang. An improved random walk based clustering algorithm for community detection in complex networks. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 2162–2167, 2011. doi: 10.1109/ICSMC.2011.6083997.

[42] S. Carlsson, B. J. Nilsson, and S. C. Ntafos. Optimum guard covers and m-watchmen routes for restricted polygons. *International Journal of Computational Geometry and Applications*, 03(01):85–105, 1993. doi: 10.1142/S0218195993000063.

[43] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, pages 302–308, 2004. doi: 10.1109/IAT.2004.1342959.

[44] W.-P. Chin and S. Ntafos. Optimum watchman routes. In *Proceedings of the second annual symposium on Computational geometry*, SCG '86, pages 24–33, New York, NY, USA, 1986. ACM. ISBN 0-89791-194-6. doi: 10.1145/10515.10518. URL `http://doi.acm.org/10.1145/10515.10518`.

[45] W.-P. Chin and S. Ntafos. Shortest watchman routes in simple polygons. *Discrete and Computational Geometry*, 6:9–31, 1991. ISSN 0179-5376. doi: 10.1007/BF02574671.

[46] W.-P. Chin and S. Ntafos. The zookeeper route problem. *Information Sciences*, 63(3):245–259, Sept. 1992. ISSN 0020-0255. doi: 10.1016/0020-0255(92)90072-G.

[47] D. V. Chudnovsky and G. V. Chudnovsky. The computation of classical constants. *Proceedings of the National Academy of Sciences*, 86(21):8178–8182, 1989.

[48] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. In *Proceedings of the 30th international conference on Automata, languages and programming*, ICALP'03, pages 1181–1196, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40493-7.

[49] N. Clisby and I. Jensen. A new transfer-matrix algorithm for exact enumerations: self-avoiding polygons on the square lattice. *Journal of Physics A: Mathematical and Theoretical*, 45(11):115202, 2012.

[50] R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, June 2005. ISSN 0097-5397. doi: 10.1137/S0097539704446475.

[51] A. Collins. 2d graph exploration using irrational numbers. Master's dissertation, University of Liverpool, 2009.

[52] A. Collins, J. Czyzowicz, L. Gąsieniec, and A. Labourel. Tell me where I am so I can meet you sooner. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 502–514. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14161-4. doi: 10.1007/978-3-642-14162-1_42.

[53] A. Collins, J. Czyzowicz, L. Gąsieniec, A. Kosowski, and R. Martin. Synchronous rendezvous for location-aware agents. In D. Peleg, editor, *Distributed Computing*, volume 6950 of *Lecture Notes in Computer Science*, pages 447–459. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24099-7. doi: 10.1007/978-3-642-24100-0_42.

[54] A. Collins, J. Czyzowicz, L. Gąsieniec, A. Kosowski, E. Kranakis, D. Krizanc, R. Martin, and O. Ponce. Optimal patrolling of fragmented boundaries. In *Proceedings of the 25th ACM symposium on Parallelism in algorithms and architectures*. ACM, 2013.

[55] C. Cooper, A. Frieze, and T. Radzik. Multiple random walks and interacting particle systems. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 399–410. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02929-5. doi: 10.1007/978-3-642-02930-1_33.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2009.

[57] Cytoscape Consortium. Cytoscape. http://www.cytoscape.org, 2013.

[58] J. Czyzowicz, P. Egyed, H. Everett, D. Rappaport, T. Shermer, D. Souvaine, G. Toussaint, and J. Urrutia. The aquarium keeper's problem. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, SODA '91, pages 459–464, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics. ISBN 0-89791-376-0.

[59] J. Czyzowicz, S. Dobrev, L. Gąsieniec, D. Ilcinkas, J. Jansson, R. Klasing, I. Lignos, R. Martin, K. Sadakane, and W.-K. Sung. More efficient periodic traversal in anonymous undirected graphs. In *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO 2009)*, pages 174–188, 2009.

[60] J. Czyzowicz, D. Ilcinkas, A. Labourel, and A. Pelc. Asynchronous deterministic rendezvous in bounded terrains. In B. Patt-Shamir and T. Ekim, editors, *Structural Information and Communication Complexity*, volume 6058 of *Lecture Notes in Computer Science*, pages 72–85. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13283-4. doi: 10.1007/978-3-642-13284-1_7.

[61] J. Czyzowicz, A. Labourel, and A. Pelc. How to meet asynchronously (almost) everywhere. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 22–30, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-98-6.

[62] J. Czyzowicz, L. Gąsieniec, A. Kosowski, and E. Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In C. Demetrescu and M. M. Halldórsson, editors, *Algorithms - ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 701–712. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23718-8. doi: 10.1007/978-3-642-23719-5_59.

[63] J. Czyzowicz, A. Kosowski, and A. Pelc. How to meet when you forget: log-space rendezvous in arbitrary graphs. *Distributed Computing*, 25:165–178, 2012. ISSN 0178-2770. doi: 10.1007/s00446-011-0141-9.

[64] J. Czyzowicz, A. Pelc, and A. Labourel. How to meet asynchronously (almost) everywhere. *Transactions on Algorithms*, 8(4):37:1–37:14, Oct. 2012. ISSN 1549-6325. doi: 10.1145/2344422.2344427.

[65] S. Das. *Distributed computing with mobile agents: solving rendezvous and related problems*. PhD thesis, University of Ottawa, Ottawa, Canada, 2007. AAINR49344.

[66] S. Das. Mobile agent rendezvous in a ring using faulty tokens. In S. Rao, M. Chatterjee, P. Jayanti, C. Murthy, and S. Saha, editors, *Distributed Computing and Networking*, volume 4904 of *Lecture Notes in Computer Science*, pages 292–297. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-77443-3. doi: 10.1007/978-3-540-77444-0_29.

[67] G. De Marco, L. Gargano, E. Kranakis, D. Krizanc, A. Pelc, and U. Vaccaro. Asynchronous deterministic rendezvous in graphs. *Theoretical Computer Science*, 355(3):315–326, Apr. 2006. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.12.016.

[68] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492.

[69] A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46(1):69–96, Sept. 2006. ISSN 0178-4617. doi: 10.1007/s00453-006-0074-2.

[70] Y. Dieudonné and A. Pelc. Deterministic polynomial approach in the plane. In F. Fomin, R. Freivalds, M. Kwiatkowska, and D. Peleg, editors, *Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 533–544. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39211-5. doi: 10.1007/978-3-642-39212-2_47.

[71] Y. Dieudonné, A. Pelc, and D. Peleg. Gathering despite mischief. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 527–540. SIAM, 2012.

[72] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Multiple agents rendezvous in a ring in spite of a black hole. In M. Papatriantafilou and P. Hunel, editors, *Principles of Distributed Systems*, volume 3144 of *Lecture Notes in Computer Science*, pages 34–46. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22667-3. doi: 10.1007/978-3-540-27860-3_6.

[73] M. Dror, A. Efrat, A. Lubiw, and J. S. B. Mitchell. Touring a sequence of polygons. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 473–482, New York, NY, USA, 2003. ACM. ISBN 1-58113-674-9. doi: 10.1145/780542.780612.

[74] A. Dumitrescu and C. D. Tóth. Watchman tours for polygons with holes. *Computational Geometry: Theory and Applications*, 45(7):326–333, Aug. 2012. ISSN 0925-7721. doi: 10.1016/j.comgeo.2012.02.001.

[75] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, Lecture Notes In Computer Science, pages 139–147, London, UK, 1992. Springer-Verlag. ISBN 3-540-57340-2.

[76] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[77] K. Easton and J. Burdick. A coverage algorithm for multi-robot boundary inspection. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005*, pages 727 – 734, april 2005. doi: 10.1109/ROBOT.2005.1570204.

[78] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31:79–113, 2001. ISSN 0178-4617. doi: 10.1007/s00453-001-0040-8.

[79] Y. Elmaliach, A. Shiloni, and G. A. Kaminka. A realistic model of frequency-based multi-robot polyline patrolling. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, volume 1 of *AAMAS '08*, pages 63–70, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-0-9.

[80] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):293–320, Dec. 2009. ISSN 1012-2443. doi: 10.1007/s10472-010-9193-y.

[81] Y. Elor and A. M. Bruckstein. Autonomous multi-agent cycle based patrolling. In *Proceedings of the 7th international conference on Swarm intelligence*, ANTS'10, pages 119–130, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15460-3.

[82] Y. Emek, E. Kantor, and D. Peleg. On the effect of the deployment setting on broadcasting in euclidean radio networks. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, PODC '08, pages 223–232, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-989-0. doi: 10.1145/1400751.1400782.

[83] Y. Emek, L. Gąsieniec, E. Kantor, A. Pelc, D. Peleg, and C. Su. Broadcasting in UDG radio networks with unknown topology. *Distributed Computing*, 21:331–351, 2009. ISSN 0178-2770. doi: 10.1007/s00446-008-0075-z.

[84] Facebook. Facebook reports first quarter 2013 results. http://investor.fb.com/releasedetail.cfm?ReleaseID=761090, 2013.

[85] F. Falciani, L. Gąsieniec, and O. Vasieva. Efficient biological networks discovery and analysis. http://environmentalomics.org/efficient-biological-networks-discovery-and-analysis/, 2013.

[86] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous oblivious robots with limited visibility. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '01, pages 247–258, London, UK, 2001. Springer-Verlag. ISBN 3-540-41695-1.

[87] P. Flocchini, B. Mans, and N. Santoro. Sense of direction in distributed computing. *Theoretical Computer Science*, 291(1):29–53, 2003. ISSN 0304-3975. doi: 10.1016/S0304-3975(01)00395-4.

[88] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, June 2008. doi: 10.1016/j.tcs.2008.02.040.

[89] F. V. Fomin, P. A. Golovach, A. Hall, M. Mihalák, E. Vicari, and P. Widmayer. How to guard a graph? *Algorithmica*, 61:839–856, 2011. ISSN 0178-4617. doi: 10.1007/s00453-009-9382-4.

[90] P. Fraigniaud and A. Pelc. Deterministic rendezvous in trees with little memory. In *Proceedings of the 22nd international symposium on Distributed Computing*, DISC '08, pages 242–256, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87778-3. doi: 10.1007/978-3-540-87779-0_17.

[91] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[92] G. Froc, I. Mabrouki, and X. Lagrange. Random walk based routing protocol for wireless sensor networks. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, ValueTools '07, pages 71:1–71:10, Brussels, Belgium, 2007. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. ISBN 978-963-9799-00-4.

[93] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software Practice and Experience*, 21(11):1129–1164, November 1991. ISSN 0038-0644. doi: 10.1002/spe.4380211102.

[94] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1927–1933, 2001. doi: 10.1109/ROBOT.2001.932890.

[95] S. Gal. Rendezvous search on the line. *Operations Research*, 47(6):pp. 974–976, 1999. ISSN 0030364X.

[96] A. Ganguli, J. Cortes, and F. Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *Robotics, IEEE Transactions on*, 25(2):340–352, april 2009. ISSN 1552-3098. doi: 10.1109/TRO.2009.2013493.

[97] Gephi Consortium. Gephi. http://gephi.org, 2013.

[98] S. K. Ghosh. Approximation algorithms for art gallery problems in polygons and terrains. In *Proceedings of the 4th international conference on Algorithms and Computation*, WALCOM'10, pages 21–34, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-11439-3, 978-3-642-11439-7. doi: 10.1007/978-3-642-11440-3_3.

[99] C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filling curves. *Image Processing, IEEE Transactions on*, 5(5):794–797, may 1996. ISSN 1057-7149.

[100] D. Gottfrid. Self-service, prorated supercomputing fun! *All the Code That's Fit to printf()*, November 2007. URL `http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/`.

[101] A. J. Guttmann. *Polygons, Polyominoes and Polycubes*. Springer Publishing Company, Incorporated, 1 edition, 2009. ISBN 1402099266, 9781402099267.

[102] D. D. Hamilton. Graph drawing algorithmics. Bachelor's dissertation, University of Liverpool, 2013.

[103] Q. Han, D. Du, J. C. Vera, and L. F. Zuluaga. Improved bounds for the symmetric rendezvous value on the line. *Operations Research*, 56(3):772–782, 2008.

[104] N. Hazon and G. A. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*, 56(12):1102–1114, Dec. 2008. ISSN 0921-8890. doi: 10.1016/j.robot.2008.01.006.

[105] M. Himsolt. Gml: A portable graph file format. Technical report, University of Passau, 2010.

[106] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[107] J. Huang, T. Zhu, and D. Schuurmans. Web communities identification from random walks. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006*, volume 4213 of *Lecture Notes in Computer Science*, pages 187–198. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-45374-1. doi: 10.1007/11871637_21.

[108] I. Jensen. A parallel algorithm for the enumeration of self-avoiding polygons on the square lattice. *Journal of Physics A: Mathematical and General*, 36(21):5731, 2003.

[109] A. Kawamura and Y. Kobayashi. Fence patrolling by mobile agents with distinct speeds. In K.-M. Chao, T.-s. Hsu, and D.-T. Lee, editors, *Algorithms and Computation*, volume 7676 of *Lecture Notes in Computer Science*, pages 598–608. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35260-7. doi: 10.1007/978-3-642-35261-4_62. URL `http://dx.doi.org/10.1007/978-3-642-35261-4_62`.

[110] G. Kazazakis and A. Argyros. Fast positioning of limited-visibility guards for the inspection of 2d workspaces. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2843–2848, 2002. doi: 10.1109/IRDS.2002.1041701.

[111] L. B. Kish. End of moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305(3âĂŞ4):144–149, 2002. ISSN 0375-9601. doi: 10.1016/S0375-9601(02)01365-8.

[112] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321295358.

[113] D. E. Knuth. *Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley Professional, 3 edition, 1997. ISBN 0201896834.

[114] S. Kondo and A. J. Yee. 5 trillion digits of pi - new world record: Pushing the limits of personal computing... how much further can we go?, 2010. URL `http://www.numberworld.org/misc_runs/pi-5t/details.html`.

[115] S. Kondo and A. J. Yee. Round 2... 10 trillion digits of pi: Same program, same computer, just a longer wait..., 2011. URL `http://www.numberworld.org/misc_runs/pi-10t/details.html`.

[116] A. Kosowski, R. Elsässer, and T. Sauerwald. Personal communication, 2008.

[117] D. R. Kowalski and A. Malinowski. How to meet in an anonymous network. In *In Proc. 13th Sirocco*, pages 44–58, 2006.

[118] D. R. Kowalski and A. Malinowski. How to meet in anonymous network. *Theoretical Computer Science*, 399(1-2):141–156, June 2008. ISSN 0304-3975. doi: 10.1016/j.tcs.2008.02.010.

[119] G. Kozma, Z. Lotker, M. Sharir, and G. Stupp. Geometrically aware communication in random wireless networks. In *Proceedings of the* 23^{rd} *annual ACM symposium on Principles of distributed computing*, PODC '04, pages 310–319, New York, NY, USA, 2004. ACM. ISBN 1-58113-802-4. doi: 10.1145/1011767.1011813.

[120] E. Kranakis, N. Santoro, C. Sawchuk, and D. Krizanc. Mobile agent rendezvous in a ring. In *Proceedings of the* 23^{rd} *International Conference on Distributed Computing Systems*, pages 592–599, 2003. doi: 10.1109/ICDCS.2003.1203510.

[121] E. Kranakis, D. Krizanc, and S. Rajsbaum. Mobile agent rendezvous: A survey. In P. Flocchini and L. Gąsieniec, editors, *Structural Information and Communication Complexity*, volume 4056 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35474-1. doi: 10.1007/11780823_1.

[122] E. Kranakis, D. Krizanc, and E. Markou. *The Moblie Agent Rendezvous Problem in the Ring.* Synthesis Lectures on Distributed Computing Theory Series. Morgan & Claypool, 2010. ISBN 9781608451364. doi: 10.2200/S00278ED1V01Y201004DCT001.

[123] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, PODC '03, pages 63–72, New York, NY, USA, 2003. ACM. ISBN 1-58113-708-7. doi: 10.1145/872035.872044.

[124] D. B. Lange and M. Oshima. Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89, Mar. 1999. ISSN 0001-0782. doi: 10.1145/295685.298136.

[125] C. Lefevre. Lhc: The guide. Technical Report CERN-Brochure-2008-001-Eng, CERN, 2008.

[126] J. Leskovec. Stanford network analysis project, 2013. URL `http://snap.stanford.edu`.

[127] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM. ISBN 1-59593-135-X. doi: 10.1145/1081870.1081893.

[128] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6:29–123, 2009. doi: 10.1080/15427951.2009.10129177.

[129] A. V. Levitin. *Introduction to the Design and Analysis of Algorithms.* Addison-Wesley, Boston, MA, USA, 2011. ISBN 9780132316811.

[130] J. Lin, A. S. Morse, and B. D. O. Anderson. The multi-agent rendezvous problem - the asynchronous case. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 2, pages 1926–1931, dec 2004. doi: 10.1109/CDC.2004.1430329.

[131] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul. Multi-agent patrolling: an empirical analysis of alternative architectures. In *Proceedings of the 3rd international conference on Multi-agent-based simulation II*, MABS'02, pages 155–170, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-00607-9.

[132] C. A. Mack. Fifty years of moore's law. *Semiconductor Manufacturing, IEEE Transactions on*, 24(2):202–207, 2011. ISSN 0894-6507. doi: 10.1109/TSM.2010.2096437.

[133] A. Marino, L. Parker, G. Antonelli, and F. Caccavale. Behavioral control for multi-robot perimeter patrol: A finite state automata approach. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 831–836, may 2009. doi: 10.1109/ROBOT.2009.5152710.

[134] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 548–556, 2012.

[135] Mersenne Research, Inc. Great internet mersenne prime search (GIMPS), 2013. URL `http://www.mersenne.org`.

[136] J. S. Mitchell. Chapter 15 - geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. North-Holland, Amsterdam, 2000. ISBN 978-0-444-82537-7. doi: 10.1016/B978-044482537-7/50016-4.

[137] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Trans. Knowl. Data Eng.*, 13(1):124–141, 2001.

[138] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Bitcoin Project, 2009. URL `http://bitcoin.org/bitcoin.pdf`.

[139] M. Naor. Verification of a human in the loop or identification via the turing test. http://www.wisdom.weizmann.ac.il/ naor/PAPERS/human_abs.html, 1996.

[140] B. J. Nilsson. *Guarding art galleries; Methods for mobile guards.* PhD thesis, Lund University, Sweden, 1995.

[141] B. J. Nilsson and D. Wood. Optimum watchmen route in spiral polygons. In *2nd Canadian Conference on Computational Geometry*, pages 269–272, 1990.

[142] D. Nishar. 200 million members! In *Linkedin Blog*. Linkedin Corporation, 2013. URL `http://blog.linkedin.com/2013/01/09/linkedin-200-million`.

[143] S. Ntafos. Watchman routes under limited visibility. *Computational Geometry: Theory and Applications*, 1(3):149–170, Mar. 1992. ISSN 0925-7721. doi: 10.1016/S0925-7721(99)00022-X.

[144] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, jan. 2007. ISSN 0018-9219. doi: 10.1109/JPROC.2006.887293.

[145] J. O'Rourke. *Art gallery theorems and algorithms.* Oxford University Press, Inc., New York, NY, USA, 1987. ISBN 0-19-503965-3.

[146] W. J. C. Orr. Statistical treatment of polymer solutions at infinite dilution. *Transactions of the Faraday Society*, 43:12–27, 1947. doi: 10.1039/TF9474300012.

[147] E. Packer. Computing multiple watchman routes. In *Proceedings of the 7th international conference on Experimental algorithms*, WEA'08, pages 114–128, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-68548-0, 978-3-540-68548-7.

[148] F. Pasqualetti, A. Franchi, and F. Bullo. On optimal cooperative patrolling. In *49th IEEE Conference on Decision and Control*, pages 7153–7158, Atlanta, GA, USA, 12 2010.

[149] C. Percival. The quadrillionth bit of pi is '0'. Available from: `http://oldweb.cecm.sfu.ca/projects/pihex/announce1q.html`, 2001.

[150] G. Prencipe. Impossibility of gathering by a set of autonomous mobile robots. *Theoretical Computer Science*, 384(2-3):222–231, 2007. ISSN 0304-3975. doi: 10.1016/j.tcs.2007.04.023.

[151] A. J. Quigley. *Large Scale Relational Information Visualization, Clustering, and Abstraction.* PhD thesis, University of Newcastle, Newcastle, UK, 2001.

[152] A. J. Quigley. Experience with fade for the visualization and abstraction of software views. In *10th International Workshop on Program Comprehension*, pages 11–20, 2002. doi: 10.1109/WPC.2002.1021304.

[153] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. ISSN 0033-3123. doi: 10.1007/BF02289527.

[154] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984. ISSN 0360-0300. doi: 10.1145/356924.356930.

[155] K. Scarfone, W. Jansen, and M. Tracy. Guide to general server security. Technical Report SP 800-123, National Institute of Standards and Technology, 2008.

[156] T. C. Schelling. The strategy of conflict. *Journal of Politics*, 23:374–376, 4 1960. ISSN 1468-2508. doi: 10.2307/2126712.

[157] M. F. Schilling. The longest run of heads. *College Mathematics Journal*, 21(3), 1991.

[158] S. Shekhar, V. Gunturi, M. R. Evans, and K. Yang. Spatial big-data challenges intersecting mobility and cloud computing. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '12, pages 1–6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1442-8. doi: 10.1145/2258056.2258058.

[159] T. Shermer. Recent results in art galleries [geometry]. *Proceedings of the IEEE*, 80(9):1384 –1399, sep 1992. ISSN 0018-9219. doi: 10.1109/5.163407.

[160] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots.* The MIT Press, 2 edition, 2011. ISBN 9780262015356.

[161] J. Simpson and E. Weiner, editors. *Parasite, n. - Oxford English Dictionary.* Oxford University Press, Oxford, 3 edition, 2005.

[162] J. Simpson and E. Weiner, editors. *Virus, n. - Oxford English Dictionary.* Oxford University Press, Oxford, 3 edition, 2008.

[163] J. Simpson and E. Weiner, editors. *Algorithm, n. - Oxford English Dictionary.* Oxford University Press, Oxford, 3 edition, 2012.

[164] N. J. A. Sloane. Number of self-avoiding polygons of length 2n on square lattice (not allowing rotations). Technical Report A002931, The On-Line Encyclopedia of Integer Sequences, 2013. URL `http://oeis.org/A002931`.

[165] S. Souissi, X. Défago, and M. Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *Transactions on Autonomous and Adaptive Systems*, 4 (1):9:1–9:27, Feb. 2009. ISSN 1556-4665. doi: 10.1145/1462187.1462196.

[166] G. Stachowiak. Asynchronous deterministic rendezvous on the line. In M. Nielsen, A. Kučera, P. Miltersen, C. Palamidessi, P. Tůma, and F. Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404 of *Lecture Notes in Computer Science*, pages 497–508. Springer Berlin Heidelberg, 2009. ISBN 978-3-540-95890-1. doi: 10.1007/978-3-540-95891-8_45.

[167] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, Mar. 1999. ISSN 0097-5397. doi: 10. 1137/S009753979628292X.

[168] A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In *Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms*, SODA '07, pages 599–608, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

[169] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. ISSN 1460-244X. doi: 10.1112/plms/s2-42.1.230.

[170] P. P. Uthaisombut. Symmetric rendezvous search on the line using move patterns with different lengths. Technical report, University of Pittsburgh, 2006.

[171] A. J. Wakefield. Statistics of the simple cubic lattice. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 47, pages 419–435. Cambridge University Press, 1951.

[172] F. T. Wall, L. A. Hiller, and D. J. Wheeler. Statistical computation of mean dimensions of macromolecules. i. *The Journal of Chemical Physics*, 22(6):1036–1041, 1954. doi: 10.1063/1. 1740258.

[173] F. T. Wall, J. L. A. Hiller, and W. F. Atchison. Statistical computation of mean dimensions of macromolecules. iii. *The Journal of Chemical Physics*, 23(12):2314–2321, 1955. doi: 10.1063/1. 1741872.

[174] F. T. Wall, J. L. A. Hiller, and W. F. Atchison. Statistical computation of mean dimensions of macromolecules. ii. *The Journal of Chemical Physics*, 23(5):913–921, 1955. doi: 10.1063/1.1742147.

[175] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684): 440–442, 1998.

[176] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela. Setihome massively distributed computing for seti. *Computing in Science and Engineering*, 3(1):78–83, Jan. 2001. ISSN 1521-9615. doi: 10.1109/5992.895191.

[177] K. Wickre. Celebrating #twitter7. In *Twitter Blog*. Twitter Inc., 2013. URL `https://blog.twitter.com/2013/celebrating-twitter7`.

[178] Wikipedia. Seven bridges of Königsberg — wikipedia, the free encyclopedia, 2013. URL `http://en.wikipedia.org/w/index.php?title=Seven_Bridges_of_K%C3%B6nigsberg&oldid=551783723`.

[179] K. Williams. Social network cohesiveness, centrality, and connectedness. Bachelor's dissertation, University of Liverpool, 2013.

[180] B. Xu and D. Z. Chen. Density-based data clustering algorithms for lower dimensions using space-filling curves. In Z.-H. Zhou, H. Li, and Q. Yang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 4426 of *Lecture Notes in Computer Science*, pages 997–1005. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71700-3. doi: 10.1007/978-3-540-71701-0_112.

[181] M. Yamashita and T. Kameda. Computing on anonymous networks. part i. characterizing the solvable cases. *Parallel and Distributed Systems, IEEE Transactions on*, 7(1):69 –89, jan 1996. ISSN 1045-9219. doi: 10.1109/71.481599.

[182] M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for mobile intruders in a polygonal region by a group of mobile searchers. *Algorithmica*, 31:208–236, 2001. ISSN 0178-4617. doi: 10.1007/s00453-001-0045-3.

[183] V. Yanovski, I. A. Wagner, and A. M. Bruckstein. A distributed ant algorithm for efficiently patrolling a network. *Algorithmica*, 37:165–186, 2003. ISSN 0178-4617. doi: 10.1007/s00453-003-1030-9.

[184] X. Yu and M. Yung. Agent rendezvous: A dynamic symmetry-breaking problem. In F. Meyer and B. Monien, editors, *Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 610–621. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-61440-1. doi: 10.1007/3-540-61440-0_163.

[185] yWorks. *yFiles for Java Developer's Guide: Exporting a Graph's Visual Representation*, chapter 9. yWorks GmbH, 2012.

[186] B. Zagrovic, E. J. Sorin, and V. Pande. β-hairpin folding simulations in atomistic detail using an implicit solvent model. *Journal of Molecular Biology*, 313(1):151–169, 2001. ISSN 0022-2836. doi: 10.1006/jmbi.2001.5033.