Robot Fence-Jumping Search

by

Ioana Alexandra Steau, B.Sc.

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

Master of Computer Science

Department of Computer Science

Carleton University

Ottawa, Ontario

January, 2017

©Copyright

Ioana Alexandra Steau, 2017

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Robot Fence-Jumping Search

submitted by **Ioana Alexandra Steau, B.Sc.**

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Professor Evangelos Kranakis, Thesis Supervisor

Professor Konstantinos Georgiou, Thesis Co-supervisor

Professor Anil Maheshwar, Director,
OCICS

Department of Computer Science

Carleton University

January, 2017

Abstract

We study a new search problem on the plane involving a robot and an immobile treasure, initially placed at distance 1 from each other. The length β of an arc (a fence) within the perimeter of the corresponding circle, as well as the promise that the treasure is outside the fence, is given as part of the input. The goal is to devise movement trajectories so that the robot locates the treasure in minimum time. Although the presence of the fence limits searching uncertainty, the location of the fence is unknown, and worst case analysis is determined adversarially. Nevertheless, the robot has the ability to move in the interior of the circle. In particular the robot can attempt a number of chord-jump moves if it happens to be within the fence or if an endpoint of the fence is discovered.

The optimal solution to our question can be obtained as a solution to a complicated optimization problem, which involves trigonometric functions, and equations. For the 1-Jump Algorithm, we fully describe the optimal trajectory, and provide an analysis of the associated cost as a function of β . Our analysis indicates that the optimal k-Jump Algorithm requires that the robot has enough memory and computation power to compute the optimal chord-jumps. Motivated by this, we give an abstract performance analysis for every k-Jump Algorithm. Subsequently, we present a highly efficient, limited memory, Halving Heuristic k-Jump Algorithm. Lastly, we present randomized algorithms which reduce the power of the adversary and worst-case performance.

I dedicate this thesis to my parents who have constantly pushed me to become a better person and a harder worker.

Acknowledgments

I would like to express my deepest appreciation to my supervisors, who provided unlimited support, patience and incredible intelligence. I thank you for your trust, guidance and honesty, I could not have done it without you.

Table of Contents

Abstract	iii
Acknowledgments	v
Table of Contents	vi
List of Tables	ix
List of Figures	x
Nomenclature	xii
1 Introduction	1
1.1 Geometric Domain	2
1.2 Searching	3
1.2.1 Knowledge of Domain	4
1.2.2 Target Search	5
1.2.3 Co-operative Searching	6
1.3 Robot Model	8
1.4 Related Work	9
1.4.1 Knowledge of Domain	10
1.4.2 Target Movement	12

1.4.3	Co-operative Searching	14
1.4.4	Obstacles	17
1.5	Publications	18
1.6	Thesis Overview	19
1.7	Summary	19
2	Naive Algorithm	21
2.1	0-Jump Algorithm	21
2.2	Summary	24
3	Divide-by-Two Algorithms	25
3.1	Repeat-Divide-by-Two Algorithm	26
3.2	Final-Jump-Divide-by-Two Algorithm	28
3.3	Summary	30
4	Main Algorithms	32
4.1	The Optimal 1-Jump Algorithm	32
4.2	Generic Description of k -Jump Algorithms	38
4.3	The Halving Heuristic k -Jump Algorithm	41
4.4	Some Optimal k -Jump Algorithms & Comparison	50
4.5	Summary	54
5	Randomized Algorithms	55
5.1	Single Random-Bit Algorithm	56
5.2	Unbounded Random-Bits Algorithms	59
5.3	Summary	65
6	Conclusion	67

6.1 Future Work	67
Mathematica Code	69
List of References	74

List of Tables

5.1	Randomized Algorithm Analysis and Worst-Case Termination Time .	66
-----	---	----

List of Figures

1.1	Geometric Domain: Disk, Treasure and Fence (β)	3
1.2	Robot Model	8
1.3	Kranakis <i>et al</i> communication types.	16
2.1	Basic landing point.	23
3.1	Repeat-Divide-by-Two Algorithm showing robot movement	26
3.2	Final-Jump-Divide-by-Two Algorithm	29
4.1	1-Jump Algorithm basic landing point.	34
4.2	k-Jump Algorithm	40
4.3	The plot jumps $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_6$ of the Halving Algorithm as a function of β	46
4.4	The performance of the Halving Algorithm for 1,2,3 and 4 jumps (de- creasing in the number of jumps, respectively) as a function of β . . .	50
4.5	Performance comparison between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β	51
4.6	Comparison of jump choices between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β	51
4.7	Performance comparison between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β	52

4.8	Comparison of the two jump choices between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β . The first jump of each algorithm is always no smaller than the second one, and eventually they all attain the value $2\pi - \beta$	52
4.9	Comparison of the three jump choices between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β . For both algorithms, the first jump of each algorithm is always no smaller than the second one, which is no smaller than the third one. Eventually they all attain the value $2\pi - \beta$	53
4.10	Performance comparison between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β . Notably, performance is nearly the same for all values of $\beta > 5$. The bigger discrepancy is observed for values of β close to 4, for which also the jump steps between the two algorithms exhibit the larger gaps (see Figure 4.9). .	53
5.1	The Robot randomly chooses a direction to move in.	57
5.2	RFW Case 1 (Random basic landing point)	61
5.3	RFW Case 2 (Random basic landing point)	61
5.4	RFW Case 3 (Random basic landing point)	62
5.5	Performance of Algorithm 5.2 (yellow line) compared to the performance of Algorithm 5.1 (blue line) as a function of the fence length $\beta \leq \beta_0$	63
5.6	Algorithm 5.3 performance (yellow line) compared to Algorithm 5.1 performance (blue line) for all fence lengths $0 \leq \beta \leq 2\pi$	65

Nomenclature

Geometric Model (1.1)

A disk (known environment) with a radius of 1 (unit).

Fence (1.1)

Unknown stationary barrier shown as an arc on the perimeter of the disk. The length of the fence, denoted by β (where $\beta > 0$), and the knowledge that the treasure is not located on the fence is given to the robot as part of the input. However, the robot is not aware of the location of the fence.

Treasure (1.1)

It is a stationary point on the perimeter of the disk whose location is unknown to the robot.

Robot Model (1.3)

Robot has the following properties:

- Initial placement in the center of the disk, 1 (unit) from the treasure and/or fence

- No vision capabilities
- Can move on the perimeter or inside of the disk
- Moves in a constant, maximum speed of 1
- Computational power

Basic Landing Point (2)

Initial robot movement from the center to the perimeter of the disk. The robot can either fall:

- Outside of the fence (β).
- Inside of the fence (β).

Remedy Phase (3)

This occurs in any of the fence-jumping algorithms when the robot jumps and lands outside of the fence. The robot will do the following:

1. Move clockwise along the perimeter of the disk until the endpoint of the fence is found, say at arc distance x
2. Return to the 1st-jump landing point along the corresponding chord of length $2 \sin(x/2)$
3. Continue executing Algorithm

Chapter 1

Introduction

Geometric search is concerned with finding a target placed in a geometric region and has been investigated in many areas of mathematics, theoretical computer science, and robotics. In each instance one aims to provide search algorithms that optimize a certain cost, which may take into account a variety of important characteristics and features of the domain, computational abilities of the searcher, assumptions about the target, etc. In this thesis, we introduce and study *fence-jumping search*, a new search problem involving a robot, an unknown stationary fence (barrier), and an unknown stationary target (or treasure) in the plane.

The location of the treasure is unknown to the robot. However, the robot has knowledge that at the start the treasure is located at a distance of 1 (unit) away from the treasure. Equivalently, the treasure is stationed on the perimeter of a disk (within the known environment), which is centered at the start point of the robot. A *fence*, a given circular arc of length β , is placed on the perimeter of the disk, whose location is also unknown to the robot. Further, the robot has the knowledge that the treasure is located on the perimeter but not on the fence. Depending on its trajectory, the robot may move along the perimeter of the circle and occasionally, say when within the fence, it may want to move along a chord, or as we say to *fence-jump*, to reduce

the time necessary to perform the search. We will analyze several fence-jumping algorithms that will allow us to reach the treasure in minimal time.

We study the fence-jumping search problem for one robot starting at the center of the disk and moving at a constant speed of 1. We assume the treasure is stationary and placed by an adversary at the beginning of each round depending on the fence location. The adversary positions the treasure on the perimeter, but outside the fence. The robot may move anywhere on the disk in an attempt to find this treasure; it is also able to use geometric knowledge so as to decrease the amount of time spent during the search. That is to say, since the robot knows that the treasure is not located on the fence, it could try to bypass it by “jumping over the fence”. The goal of this thesis is to determine a trajectory so that the robot finds the treasure in optimal time.

1.1 Geometric Domain

Let us begin with an understanding of our model’s geometric domain. In this thesis, we will make the assumption that the robot is aware of the environment shape: a disk. We will now provide further definitions for the domain constraints, which are visually depicted in Figure 1.1. We can observe that the fence is shown using a red outline, while the treasure is a small square, both located on the perimeter of the disk. We will also discuss the knowledge the robot has about these domain constraints.

Fence (β)

We denote the length of the fence by β (where $\beta > 0$), given to the robot as part of the input. It is considered to be an unknown stationary barrier depicted as an arc on the perimeter of the disk. Although the location of the fence (β) is unknown, the

Figure 1.1: Geometric Domain: Disk, Treasure and Fence (β)

robot has knowledge of its existence and its length. Furthermore, it is aware that the treasure is not located on the fence.

Treasure

We define the treasure to be a stationary point located on the perimeter of the known environment (disk), at an unknown location. The treasure does not take any space on the perimeter.

1.2 Searching

In this section, we will provide the basic background necessary to understand search problems. Search problems have been thoroughly studied in the field of Computer Science and Mathematics. We will examine the main attributes of the search problem, while taking into account essential features such as geometric knowledge of the domain, stationary or mobile target etc., as well as the number of searchers. Afterwards, we will consider a more specific search problem that focuses on a co-operative

approach. An example of a co-operative problem includes evacuation, where all robots attempt to exit the specific domain. The goal of the search problem is for searchers (i.e. robots) to find a specific item (i.e. target). Providing the background necessary for search problems is essential in beginning to understand our model's framework.

1.2.1 Knowledge of Domain

Let us begin by examining environment knowledge that is given as part of the initial input to the robot. We will explore two types of potential knowledge of the domain the robot may possess:

1. Zero Knowledge of Domain

On the one hand, we have a search problem in which the searchers have no environment awareness. In this case, the main focus of the problem is for the robot to gain an understanding of the environment. The search problem would, thus, have a slight modification taken in the form of mapping and/or recognition of the domain as the first priority. Meanwhile, searching for the target would be an additional bonus. Of course, in this type of search, we are looking to minimize the time necessary to find the target and/or find the shortest path to the target.

2. Full Knowledge of Domain

On the other hand, we look at the search problem with knowledge of the domain being fully present. In this case, our main goal is to find the target within the environment in efficient time and/or taking the shortest path. Since the searcher(s) have insight on the most important problem input, the environment, in order to make the problem more natural, constraints such as stationary

or adversarial targets, constrained search motion etc. are introduced. Furthermore, we can also consider the different types of environments available: graphs or geometric. In a graph environment, the targets reside on graph edges or nodes. Meanwhile, in a known geometric terrain, a specific shape is given as part of the input. If we are given a geometric domain, then we must consider additional constraints in the form of robot movement; is the robot allowed to move in the shape's area and/or just on the perimeter?

It is quite evident that the type of knowledge of the domain given as the initial input would drastically change a problem. For example, problems with zero knowledge of the domain focus foremost on the recognition of the terrain and then on the shortest paths/time. Meanwhile, problems that consider initial input with full knowledge of the domain, such as ours, center around the creation of more efficient algorithms to decrease the time and paths necessary to find a target.

1.2.2 Target Search

We discuss two different types of targets that are considered in our search investigation. We will define the targets to be points in an (un)known environment, which searchers are trying to locate. The two types of targets discussed are as follows:

1. Stationary Target

Based on the definition, we know that the target will not move from its initial placement.

2. Mobile Target

Meanwhile, the moving (evading) target is one which will move depending on the searcher's location. In that, the evading target will move so that the searcher will not find it. The introduction of evading targets proposes the important

concept of adversarial movement. It forces us to think of such questions as: what is the allowed sequence of movements, how far away can the target move etc.?

1.2.3 Co-operative Searching

With the inclusion of additional searchers, we introduce several types of searches, which are dependent on the requirements given to the participating search robots. We are aware that, within these requirements, the addition of multiple searchers opens up the possibility for communication. Were communication models not present within the problem, it would be difficult to justify problem termination. This is due to the fact that the other searchers might not have knowledge of whether the target has/hasn't been found. Since the goal of the search is for searcher(s) to find the target, this poses a grave complication. Therefore, we must acquaint ourselves with the two searcher communication methods: wireless and non-wireless (face-to-face).

We begin by proposing two types of terminations within the search problem:

1. In the simplest type, the problem terminates when only one of the searchers finds the target.
2. In the second type, all of the searchers are required to reach the target in order for search conclusion to occur. The need for all of the robots to find the target introduces a different search problem called *evacuation* (group search). In the evacuation problem, the target is now called an exit and *all* searchers must reach this exit for the problem to terminate.

Understanding these two different types of search problem terminations (one searcher finding the target vs. all finding the target) is essential in order to discuss the different communication models.

Next, we discuss the different types of communication models available within the search problem: wireless and non-wireless (face-to-face).

1. Wireless Communication

Robots are able to converse with each other via remote messages. In other words, the robots can talk to each other without any dependence on location. In the case where all searchers must find the target, the wireless communication model makes it very simple for searcher(s) to report the coordinate location of the target. Meanwhile, when the goal is for one searcher to find the target, the searcher who finds the target can simply announce that the target has been found, thus, terminating the search.

2. Non-wireless (Face-to-Face) Communication

As the name suggests, the robots are able to talk to each other only when they are located at the same point within the environment. Were the searcher who located the target cognizant of the other searchers' movements, it could calculate and move to their location in order to inform them of either the precise target location, or of search termination.

Overall, this section introduced search problems with more than one searcher. It allowed for a background on two different termination search problems and familiarized us with two communication types. We are able to realize the importance of the termination type in the proof of algorithmic conclusion. Further, the type of transmission knowledge is extremely important in determining optimal algorithms for problems with more than one searcher.

1.3 Robot Model

We now introduce our robot model, which includes the movement of a robot within the specified domain. In this section, we will gain an understanding of the robot definition within the given environment. We will gain this knowledge by defining items such as the initial domain placement of the robot, the robot's vision capabilities, as well as the robot's movement i.e. where it can move and movement speed. Thus, this section will provide a comprehensive understanding of the robot model.

Initially, we make the assumption that the robot is located in the center of the disk with a radius of 1, and a treasure is located at distance of 1 from the robot, on the perimeter of the disk. This can be seen in Figure 1.2 where the robot starts in the middle of the disk, then makes a movement of 1 (unit) to reach the treasure. Further, the treasure location is always unknown to the robot until it moves directly over its point location. That is to say, the robot has no vision capabilities, in that it becomes aware of what each point on the circle is, i.e. a fence point, treasure or nothing special, only if the point is visited.

Figure 1.2: Robot Model

The robot moves with a constant, maximum speed of 1 (unit) throughout its search, with the ability to stop and restart at any time. It moves in a clockwise or counter-clockwise direction and, depending on its location, the robot can move on the boundary (perimeter) and/or interior of the disk. Therefore, the robot has the computational power to numerically solve trigonometric equations through the use of deterministic processors which possess the required memory for these processes i.e for fence-jumping. The ability to jump the fence is essential to the formation of our algorithms as it would allow the robot to bypass parts of the perimeter that will, in turn, minimize the time required to reach the treasure.

In this thesis, the goal of the robot is to optimize the length of its trajectory using various types of movements, i.e. the robot may walk on the fence or even jump over the fence moving along a chord (within the interior of the circle) in order to minimize time. Several algorithms will be mentioned in order to provide a further understanding of the complexity of the problem, as well as showing the algorithm optimality.

1.4 Related Work

In this section, we will perform a literature review on the research resembling the problem discussed in Section 1.1 and Section 1.3. All of the papers discussed in this section have the commonality of being search problems. Utilizing the search problem background provided in Section 1.2, we will examine problems studied with different types of domains, targets and communication models. In each of these subsections, we will emphasize the similarities, as well as the differences between current research problems completed in this area and our thesis. Using these comparisons, we will prove that the *Robot Fence-Jumping Search* paper introduces a number of new,

relevant concepts, thus making a contribution to the field of Computer Science.

1.4.1 Knowledge of Domain

Often, the essential part of the robot activity is the perception and/or mapping of the domain, which is generally the case when the searcher(s) have zero knowledge of the domain. Meanwhile, in the case where searcher(s) are aware of the current domain, the main objective of the search is to minimize the time to find the target. We will first look at problems with zero knowledge of the domain and then move to full knowledge of the domain, inspecting graphs and geometric domains.

We begin our analysis with a paper by Baeza-Yates *et al* who studied the search problem within a unbounded plane [1]. In their paper, they examined the movement of an entity (a robot) who had zero knowledge of the environment, so as to replicate maze-like problems. One of the aims of the paper was to provide an understanding of the most important knowledge when the robot has no awareness of the environment. Using rigorous proofs and analysis, Baeza-Yates *et al* were able to confirm the intuition that when the robot does not know the environment, the most relevant information is having knowledge of the direction in which the item searched for is located, compared to the distance to that particular item.

This problem that Baeza-Yates *et al* introduced in [1] is also known as the w-lane cow-path problem. Let us consider a cow who is standing at a crossroads with w unknown, unbounded paths. The cow knows that there is a grazing field located somewhere along one of the paths, however, does not know which path it should take. Further, the cow does not have any vision capabilities. As previously mentioned in Section 1.3, this means that the cow will not know it has reached the grazing ground until it is located at that particular point. The goal of the cow-path problem is for the cow to find the field with the least amount of distance traveled. This problem has

been studied numerous times including with a randomized approach [2], finding the minimum number of street turns necessary in order to get to a destination [3], goal modification to include probabilistic grazing locations [4], along with many others.

In continuation to this maze cow problem, we introduce 1940's Shannon's mouse algorithm [5]. In this problem, we observe a mouse (the searcher) looking for a piece of cheese (the target). The mouse is placed in a maze-like structure with a 25-square checkerboard, where each square has two bits associated to it as illustrated in [5]. These bits indicate whether the mouse has already visited the square and an arrow depicting the direction the mouse moved. This concept of restrained memory usage is proposed later in our thesis with the Halving Heuristic k -jump Algorithm where we attempt to limit the amount of memory used by the robot through various limitations of movement such as number of jumps [6].

More similar to our problem, we explore search problems in which the robot has knowledge of the domain. The paper by Baeza-Yates *et al* also examined the movement of a robot within a known plane (i.e. a line) with a target that is located at an unknown location. Upon further calculations, [1] determined the worst-case movement for a robot searching for an unknown target at an unknown location in the plane, such as a simple line. The simple strategy of doubling the distance traveled in each direction utilized in [1], guarantees finding the target in $9d$. We further observe that there exists no better deterministic strategy.

The problem with this approach is that it occurs when there is no unit to mark distance. In the beginning, the searcher would need to make an infinite number of steps. Therefore, [7] showcases a new, more practical modification to the doubling strategy, introducing a cost (δ) for each turn. Through the use of the cost, [7] is able to create a finite number of turns and a search optimal time of $9d + 2\delta$. The search problem considered in our thesis is related to the line search idea, utilizing the concept

that we are searching on a boundary of a known shape, in our case a disk. In contrast, we provide the additional functionality regarding the ability to make fence-jumps in order to allow for our fence constraint, as well as no distance or direction knowledge for the robot.

Overall, this section provides an analysis on the different types of knowledge of the domain the robot has (known vs. unknown), along with the similarities of the current research available to our thesis. We conclude that when the robot has zero knowledge of the environment, the search problem tends to focus on terrain exploration. Meanwhile, when full knowledge of the environment is present, various techniques, such as doubling distance covered, are used in order to optimize the amount of time needed to find the target. Similarly, in our thesis, we focus on the utilization of fence-jumps in order to limit the time required to reach the target.

1.4.2 Target Movement

This section will examine the available research for two different types of target movements: moving and stationary. For further reference, see Section 1.2.2 where we discuss the definitions of these target types.

We will first analyze the research completed on moving targets that, again, have the option to change locations based on different constraints. An example of a target movement could be that of search avoidance. [8] examines this type of target location change through the use of various algorithms. It concludes that the guarantee of the searcher finding the target is limited upon the target speed. In other words, in order to have full certainty that the target will be found at some point, there exists a dependence on the target speed being slower than that of the searcher. The concept introduced by [8] has been implemented in robot events such as RoboCup [9]. The RoboCup event was initially set up to implement robots with the capability of

playing against the World Cup soccer champions. The applications of such a robotic implementations has moved forward to include items such as robot rescue missions and home assistance.

Next, we look at stationary targets. This is of extreme importance as it is one of the basic characteristics of our model. Around sixty years ago, Bellman asked an important, yet simplistic question tied to the minimization of distance. It went in the following way: if a hiker were to be stuck in the middle of the woods and was aware of the various dimensions and shapes of the woods, what is the best path of escape [10]? Naturally, we know that the best path would be that which would get the hiker out the fastest. An analysis of this idea can also be seen in the lost in a forest problem [11, 12]. Although the basis of these problems center around the minimal distance to exit, we know that there is a fixed exit (target) that hikers must find. Similar to this problem, our thesis introduces a robot (the hiker) which requires a minimized time trajectory algorithm required to find a treasure (the escape).

Searching for a motionless target has been studied in the cow-path problem [13], and plane searching problem [1, 14]. The cow-path problem, also called the linear search problem, focuses on cows trying to find a pasture (stationary target) in minimal time using linear lanes and is mentioned in thorough detail in Section 1.4.1.

As can be observed, mobile and stationary targets create two different problems. Implementing mobile targets within the problem, we notice that the main focus is to create an algorithm that will guarantee that the target is found. That is to say, the algorithms must prove that, even though the target is mobile, the searcher will still be able to locate it at some point i.e. decreased speed for the target. Meanwhile, with the implementation of a stationary target, the problem centers around the minimization of distance traveled by the searcher and/or optimized time to finding the target.

1.4.3 Co-operative Searching

The co-operative search type of algorithms arises with the addition of more than one searcher. This type of search is explained in more detail in Section 1.2.3. We will be focusing on the type where, in order for the algorithm termination to occur, *all* searchers must find the target. More specifically, we will be examining related works that are of evacuation (gathering) search type.

Let us begin by looking at the evacuation of a robot (searcher), with knowledge of the environment, in minimal time. If there is more than one robot, the aim of this problem is for all of the robots to optimally exit the domain. This problem has been studied intensely by Kranakis *et al* in numerous papers. It was studied in [15, 16] where at least two robots initially move from the center of a known environment, a disk, to an unknown point located on the perimeter. The basic characteristics of the robots are as follows:

- Robots are 1 (unit) away from the exit at the initial position
- Robots have the ability to move inside of the disk
- Communication may be used in order to speed the task
- Robots work together in order to achieve a common goal: exit.

As can be observed, there are preliminary resemblances between the papers presented by Kranakis *et al* and ours, including the initial robot placement, being 1 (unit) away from the treasure or exit, the robot moving from the center of the disk to the perimeter, robot constant time movement, and robot perimeter searching. However, it is of importance to note that in [15, 16], the robot(s) do not have any fence-like barrier placed on the perimeter as to interfere with the robot movement. This is an essential characteristic of our problem. Again, since we know that the treasure is not

placed on the fence, we are trying to find the most optimal fence-jump algorithms to find the treasure. Since we have mentioned the fence must be greater than zero, the problems brought forward by Kranakis *et al* will never be like the problem proposed in this thesis.

Interestingly, it is important to observe that our thesis mainly focuses on the search for treasure with one robot. Therefore, this problem could be viewed as both a search problem, in the sense that the robot is searching for a treasure, as well as an evacuation problem with the treasure being seen as an exit. There is only one robot, therefore all entities have “exited” the domain. However, when we discuss the addition of a robot within our current environment, this observation of our problem being both an evacuation and a search problem changes. During this time, we will discuss whether we are using the evacuation approach from the papers [15, 16] or simply performing a search where one robot finding the exit completes the problem.

In continuation, the use of wireless and non-wireless communication is applied to [15, 16] in order to observe the time complexity for each of the two communication methods. The definitions of the type of communications used in these papers (wireless vs. face-to-face) are explained in Section 1.2.3. In Kranakis *et als*’ problem definition, it is mentioned that the robots have the ability to make short-cuts. In the wireless model the notified robots can calculate the fastest trajectory to exit using these short-cuts. Thus, allowing the robots to move in the disk circumference as can be seen in Figure 1.3a. Meanwhile, with the face-to-face model, if there are only two robots, the robot who found the exit must either wait for the other robot to reach the exit, or determine the possible future location of the other robot and notify it by moving to that particular future location, which can be observed in Figure 1.3b.

Intuitively, to optimize the evacuation time, the goal is to make the non-wireless

Figure 1.3: Kranakis *et al* communication types.

algorithm complexity as close to the wireless as possible. This is done through various techniques such as using shortcuts after finding the exit to notify the other robot. Whilst the shortcuts technique is present in our thesis in the form of fence-jumping, the main similarity is the use of robot geometric knowledge in order to perform said task, such as finding the treasure or exit.

Overall, we can observe similarities in the problem definitions of [15, 16] and our thesis. However, through our examination, we are able to determine that these problems could never be the same. A main reason for this is the fence structure that our thesis implements into the problem definition. The fence is one of the most important parts of our problem as it is what we are attempting to optimize. That is to say, since the treasure is never located on the fence, our thesis finds the best jumps to make in order to exit the fence in minimal time. Meanwhile, the focus of Kranakis *et al* paper is for all robots to exit in minimal time.

1.4.4 Obstacles

This introduction of a fence brings us to the the problem of obstacle avoiding trajectory. The avoidance of obstacles is also thoroughly studied, whether the robot has knowledge of the terrains or does not have knowledge of the terrain [17–19]. It is important to keep in mind that, like target movements, obstacles can be stationary or moving. Most obstacle avoiding algorithms give the robot particular goal coordinates with assumptions, i.e. no obstacles, and the robot attempts to find the shortest path. The robot continues on the path until it encounters a new map event, like an obstacle, adds the information and then continues to plan for the shortest path.

We will begin by examining moving obstacles in both known and unknown environments. The first paper we will look at determines an exploration algorithm for probabilistic, unknown domains. In this particular problem, the searcher (car) is given a particular, unknown environment that has mobile pedestrians (obstacles) who follow a list of diverse, typical motions [20]. The planning algorithm mentioned attempts to extend the rapid-exploring random algorithm. In the rapid-exploring algorithm, the probability and future trajectory of the obstacle, along with its probability of collision are specifically examined. Therefore, the extension of the algorithm tries to determine part of a motion planner whose probability has a real-time update according to most recent estimate calculations. Overall, [20] relies on past movements of the obstacles to determine possible future ones, meaning that the searcher has a probabilistic idea of the location of the obstacles.

Next, we look at the possible algorithms exploring stationary obstacles. We examine [21], which attempts to find the minimum-cost trajectory in a graph. As the robot moves and comes along new items such as obstacles, it tries to re-plan its path so as to reduce the movement cost. Further, the robot has the ability to compute a different path to find a target. Therefore, as soon as it finds new information, it could

either wait to figure out a new path, potentially wasting time, or go in the wrong direction. This causes the robot's ability to plan rapidly to be essential. Therefore, the algorithm produced in [21] focuses on incrementally repairing the path as the robot gets new information. Additional implementations to the algorithm that focus on reducing the time to calculate repairs for the initial path and reduce the time required to calculate these paths are made. Overall, this paper attempts to create algorithms in which the robot is able to quickly calculate a new, minimum path based on the new events it encounters, such as obstacles.

To summarize, this section focused on two different types of obstacles: mobile versus stationary. Whilst examining the moving obstacle types, we observe that there is an emphasis on trying to provide obstacle location predictions based on their past behaviour. Contrary to this, our thesis focuses on an obstacle known length (β), with a completely unknown location within the environment. Meanwhile, the second type of obstacle is more comparable to our fence definition as it focuses on an adjustment of the searcher's movement plan. This, again, is due to the knowledge that the target (treasure) is not located within the fenced area. Contrary to the obstacle papers mentioned, we allow the robot to move on the fence. Thus, the robot can land on the fence when it moves from the center and/or from the fence-jumps it makes to exit the fence or walk along it.

1.5 Publications

Part of the material presented in this thesis has been published [22] and presented at a conference [23].

1.6 Thesis Overview

The main objective of this thesis is to design algorithms which have the ability to locate the treasure in optimal time, while adapting to a fence structure located on the perimeter. We will first begin with a naive algorithm (Chapter 2). Here, we define an algorithm requiring zero fence-jumps, evaluating its worst case termination. This algorithm will act as the foundation of all future algorithms. Afterwards, Chapter 3 focuses on two algorithms which use the divide-by-two approach as a fence-jumping technique. The divide-by-two approach will make decreasing jumps of $\frac{\beta}{2}$ etc. in order to exit the fence structure. Meanwhile, in Chapter 4, we improve upon the divide-by-two algorithms in order to determine some main algorithms which focus on optimality. In that, we will define algorithms for the most optimal 1-jump, as well as k-jump algorithms in order to provide a comparison of worst case termination time optimality. In the last part of the thesis, Chapter 6, we target the addition of randomization within the 0-jump algorithm mentioned in Chapter 2. We will focus on two different additions of randomness: 1-bit and infinite bit. We will analyze the worst case termination time for each of these algorithms and determine an algorithm which improves upon both of these randomization techniques and provide a proof of improvement. We then conclude the thesis, including opportunities for future work.

1.7 Summary

In this chapter, we study the background necessary to define the fence-jumping search problem for one robot starting in the center of the disk and moving at a constant speed 1 (unit). We provide definitions for terminologies necessary in understanding this thesis problem. It is clear that for a single robot, such as in our problem, communication is not a necessity. We discussed communication types in Section 1.2.3

in order to recognize the possible addition of robots within future work of this problem. We also analyzed the related works that have been done, justifying that the problem in this thesis introduces a new, relevant concept within the field.

Furthermore, we give a thorough definition of the problem model. We assume the treasure is stationary and placed by an adversary at the beginning of each round depending on the fence location. The adversary positions the treasure on the perimeter, but outside the fence. The robot may move anywhere on the disk in an attempt to find this treasure; it is also able to use geometric knowledge so as to decrease the amount of time spent during the search. That is to say, since the robot knows that the treasure is not located on the fence, it could try to bypass it by “jumping over the fence”. The goal of this thesis is to determine a trajectory so that the robot finds the treasure in optimal time.

Chapter 2

Naive Algorithm

To begin, we will provide a naive solution to our treasure finding problem. In this chapter, we will define an algorithm which does not require any fence-jumps. We will also evaluate the worst case termination of the algorithm, which will act as the skeletal support for future algorithms.

2.1 0-Jump Algorithm

As previously mentioned in Section 1.1, we denote the length of the fence by β , given as part of the input. Independently of the algorithm considered, any deterministic algorithm we examine will first have the robot move to an arbitrary point on the perimeter of the circle, thereafter referred to as the *basic landing point*. The intention of the robot will be to start moving/searching the disk counterclockwise, which is further examined in Algorithm 2.1.

Algorithm 2.1 0-Jump Algorithm

- 1: Walk to the perimeter of the disk
 - 2: Continue walking on the perimeter counterclockwise
 - 3: **if** you reach fence endpoint **then**
 - 4: Hop along the corresponding chord of length
 - 5: $2 \sin(\beta/2)$
 - 6: **else**
 - 7: Walk on perimeter until you find treasure
 - 8: **end if**
-

This thesis focuses upon algorithms that perform well under worst case analysis. As such, the performance of any algorithm will be determined after an adversary decides on the location of the:

- Basic landing point
- Fence
- Treasure

For the sake of exposition, we now present the worst case termination time, depending on Algorithm 2.1's location of the basic landing point forin Lemma 2.1.1.

Lemma 2.1.1. *The worst case termination time of Algorithm 2.1 is¹*

$$\left\{ \begin{array}{ll} c_0^0 := 1 + 2\pi - \beta + 2 \sin(\beta/2) & \text{Figure 2.1a} \\ c_1^0 := 1 + 2\pi & \text{Figure 2.1b} \end{array} \right.$$

where the reference in the right column indicating the Figure which applies to the case.

¹The usefulness of notation c_j^i for the cost of the algorithm will be transparent in the next sections

Figure 2.1: Basic landing point.

Proof. In order to fully analyze the performance of this algorithm, we must examine both basic landing cases as follows:

- Basic Landing Point Outside β

Suppose that the basic landing point is outside the fence, as seen in Figure 2.1a, and say that the clockwise distance between the landing point and the fence is $x \in (0, 2\pi - \beta)$. It is quite evident that the adversary would place the treasure clockwise in-between the landing point and the fence, at a clockwise distance of $y \in (0, x)$ from the landing point. Then, for all $x \in (0, 2\pi - \beta)$ the cost of the algorithm would be:

$$\sup_{y \in (0, x)} \{1 + 2\pi - \beta + 2 \sin(\beta/2) - y\} = 1 + 2\pi - \beta + 2 \sin(\beta/2).$$

- Basic Landing Point Inside β

In this case, the landing point is within the fence, as illustrated in Figure 2.1b. Suppose that the clockwise distance between the landing point and the endpoint

of the fence is $x \in (0, \beta)$. Let us also say that the clockwise distance between the same fence endpoint and the treasure is $y \in (0, 2\pi - \beta)$. Then, the robot will locate the treasure in the following time:

$$\sup_{x,y} \{1 + 2\pi - x - y\} = 1 + 2\pi.$$

The worst case analyses of the two basic landing points that could occur during the algorithm proves Lemma 2.1.1. □

It is intuitive that having the basic landing point outside the fence is a “favorable event”, as the robot can completely skip the fenced area. This means that, for all β , $c_0^0 \leq c_1^0$. It can be followed formally from the fact that the non-negative expression $\beta - 2\sin(\beta/2)$ is increasing in $\beta > 0$. Hence, the performance of Algorithm 2.1 is $1 + 2\pi$.

Next, we focus on algorithms that can address the choice of the adversary placing (basic) landing points within the fence. In such algorithms the robot will try to jump in an attempt to land outside the fence.

2.2 Summary

This chapter focused on the creation of a framework for the addition of fence-jumps. We elucidate an algorithmic description of the movements necessary for the robot to find the treasure without any fence-jumps. Further, we assessed the worst case algorithm termination and concluded that the best case basic landing point would be that where the robot falls outside of the fence.

Chapter 3

Divide-by-Two Algorithms

This section will focus on two algorithms which try to make a jump(s) of $\frac{\beta}{2}$. The first, Repeat-Divide-by-Two Algorithm, will attempt to make decreasing jumps of $\frac{\beta}{2}$ in order to attempt to make its way out of the fence. Specifically, this algorithm will attempt to fence-jumps of $\frac{\beta}{2^i}$ (where i represents the number of jumps). We try to use small jump values so as to not jump too far outside of the fence. Otherwise, the 1st-jump landing point is outside the fence and the robot applies the following *remedy phase*: move clockwise along the periphery of the circle till the endpoint of the fence is found, say at arc distance x , and then return to the 1st-jump landing point along the corresponding chord of length $2 \sin(x/2)$, and continue executing the current algorithm. Meanwhile, the second, Final-Jump-Divide-by-Two Algorithm, focuses on getting the robot out of the fence as soon as possible. This algorithm makes 3 jumps equaling a total of β . Overall, both of the algorithms discussed in this chapter use some form of $\frac{\beta}{2}$ jumps in order to allow the robot to exit the fenced area.

3.1 Repeat-Divide-by-Two Algorithm

This algorithm focuses on the robot jumping $\frac{\beta}{2^i}$ until it reaches the end of the fence or lands outside of the fenced area. Again, i is the number of jumps that the robot has made in within the fence. That means that, as the robot makes more jumps, the jump length will be decreasing by a factor of two. This is demonstrated in Figure 3.1 where the robot makes a basic landing point inside of the fence and then jumps $\frac{\beta}{2^i}$ until it exits. This figure also showcases the remedy phase where, in the final jump, the robot lands x away from the fence endpoint. The precise robot movements and diverse cases that could occur are described in Algorithm 3.1.

Figure 3.1: Repeat-Divide-by-Two Algorithm showing robot movement

Algorithm 3.1 Repeat-Divide-by-Two Algorithm

```

1: Walk to the perimeter of the disk
2: if you land outside of the fence then
3:     Perform Algorithm 2.1
4: end if
5: while you land inside the fence do
6:     jump a chord of  $\frac{\beta}{2^i}$  ( $i = \#$  of fence jumps)
7:     if you reach  $i+1$  jump then
8:         Walk on perimeter until reach the treasure
9:     end if
10: end while
11: if you land outside after the jump then
12:     Perform remedy phase
13: end if

```

Theorem 3.1.1. *The time required to find the treasure is at most $S \leq 1 + \sum_{i=0}^{k+2} 2 \sin\left(\frac{\beta}{2^i}\right) + 2\pi - \beta$.*

Proof. Let us analyze the time required if you land outside of the fence. We consider Chapter 2 for the analysis of the robot jumping outside of the fence and moving x . We also want to take into account that we must make $\frac{\beta}{2^i}$ jumps if we land in the fence, beginning with $i = 1$. We get the time complexity of:

$$\begin{aligned}
 S &= 1 + \sum_{i=0}^{k+1} 2 \sin\left(\frac{\beta}{2^i}\right) + x + 2 \sin\left(\frac{x}{2}\right) + 2\pi - \beta - x \\
 &\leq 1 + \sum_{i=0}^{k+1} 2 \sin\left(\frac{\beta}{2^i}\right) + 2 \sin\left(\frac{x}{2}\right) + 2\pi - \beta.
 \end{aligned}$$

Now, because we landed outside, $x \leq 2\pi - \beta$ and $x \leq \frac{\beta}{2^{k+1}}$. Therefore, $x \leq \min\{2\pi - \beta, \frac{\beta}{2^{k+1}}\}$. We get that $\frac{x}{2} \leq \min\{\pi - \frac{\beta}{2}, \frac{\beta}{2^{k+2}}\}$. This allows to modify our time to the following:

$$S \leq 1 + \sum_{i=0}^{k+2} 2 \sin\left(\frac{\beta}{2^i}\right) + 2\pi - \beta$$

This completes the proof. □

Although a logical approach, if the robot lands initially at the mid point of the fence this algorithm will lead to *Zeno's Paradox*. This means that we can potentially have a infinite number of steps until the robot is able to reach either the end of the fence, or outside of the fence area.

3.2 Final-Jump-Divide-by-Two Algorithm

Since we observed that the Repeat-Divide-by-Two Algorithm produced a possible infinite number of jumps, we can modify the algorithm in the following way so that we can ensure we exit the fence: limit the robot's fence-jumps to three. No matter where the robot is located on the fenced area, after 3 jumps the robot will always exit the fence. Instead of constantly jumping a chord of $\frac{\beta}{2^i}$, we will decrease by a factor of two but will not continue to do so until we exit. We will perform the $\frac{\beta}{4}$ fence-jump more than once as seen in Figure 3.2. Algorithm 3.2 provides a thorough analysis of the robot movement using this approach. Since we are aware that whenever the robot lands outside of the fence from a jump, it performs a remedy phase, we will not include the phase in the algorithm.

Figure 3.2: Final-Jump-Divide-by-Two Algorithm

Algorithm 3.2 Final-Jump-Divide-by-Two Algorithm

- 1: Walk to the perimeter of the disk
 - 2: **if** you land outside of the fence **then**
 - 3: Perform Algorithm 2.1
 - 4: **end if**
 - 5: **if** you land inside of the fence **then**
 - 6: Jump a chord of $\frac{\beta}{2}$
 - 7: **if** you land inside **then**
 - 8: Jump a chord of $\frac{\beta}{4}$
 - 9: **if** you land inside **then**
 - 10: Jump a chord of $\frac{\beta}{4}$
 - 11: **if** you land inside after the jump **then**
 - 12: Walk on perimeter until reach the treasure
 - 13: **end if**
 - 14: **end if**
 - 15: **end if**
 - 16: **end if**
-

Theorem 3.2.1. *The time required to find the treasure is at most $1 + 2 \sin\left(\frac{\beta}{2}\right) + 6 \sin\left(\frac{\beta}{4}\right) + 2\pi - \beta$.*

Proof. We must consider the event where the robot moves from the center of the disk and lands inside of the fenced area. Based on the algorithm definition, we will make a jump of $\frac{\beta}{2}$. If this doesn't allow us to exit, we make another jump, this time of $\frac{\beta}{4}$. In the occurrence where the robot still does not exit, we make another jump of $\frac{\beta}{4}$. Again, we must also keep in mind the remedy phase, which included with this algorithm produces the following time complexity:

$$\begin{aligned} S &= 1 + 2 \sin\left(\frac{\beta}{4}\right) + 2 \sin\left(\frac{\beta}{8}\right) + 2 \sin\left(\frac{\beta}{8}\right) + x + \\ &\quad 2 \sin\left(\frac{x}{2}\right) + 2\pi - \beta - x \\ &\leq 1 + 2 \sin\left(\frac{\beta}{4}\right) + 4 \sin\left(\frac{\beta}{8}\right) + 2 \sin\left(\frac{x}{2}\right) + 2\pi - \beta \end{aligned}$$

We can make the following observations, which are similar to those in the proof of Theorem 3.1.1: $x < 2\pi - \beta$ and $\beta \geq \frac{4\pi}{3}$ which means $\pi - \frac{\beta}{4} \leq \frac{\beta}{8}$. We can, therefore, say that $\frac{x}{2} < \pi - \frac{\beta}{4}$. Thus, giving us: $\frac{x}{2} \leq \frac{\beta}{8}$. Using this proof, we get: $S \leq 1 + 2 \sin\left(\frac{\beta}{4}\right) + 6 \sin\left(\frac{\beta}{8}\right) + 2\pi - \beta$. $\square$

3.3 Summary

In this chapter, we looked at two possible algorithms that attempt to make a jump of $\frac{\beta}{2}$. We first examined the Repeat-Divide-by-Two Algorithm that tries to constantly decrease the size of the jump by 2 in order to exit the fence. We noticed that this actually led to Zeno's Paradox, causing the robot to produce a possible infinite number of jumps. We then tried to improve on this by allowing the robot to make only

3 possible jumps equaling a total of β . This was done to ensure that the robot exits the fence in a finite number of steps. These two algorithms are provided in order to understand the complexity of the problem and to create a basis for the robot movement. Next, we will focus on finding more optimal fence-jump results and providing comparisons on the algorithms in order to find the minimal amount of time necessary to find the treasure.

Chapter 4

Main Algorithms

In this chapter, we will improve upon the Divide-by-two approach and will look at several algorithms with an attempt to find optimal fence-jumps when the basic jump is inside of the fence. First, we will observe the optimal 1-Jump algorithm. Afterwards, we will define a generic description of k -Jump algorithms and analyze their cost depending on the jump landings. Subsequently, we will analyze a Halving k -Jump algorithms, where k is the max number of jumps the robot makes so as to overcome the fence and find the target. We will then provide a comparison between the worst-case termination of these main algorithms.

4.1 The Optimal 1-Jump Algorithm

In this section we analyze the optimal 1-Jump Algorithm, which also serves as a warm-up for the analysis of the generic k -Jump Algorithm. 1-Jump Algorithms are fully determined by the (unique) chord jump of corresponding arc-length α they make in the case where the basic landing point (of the robot) is within the fence.

It is worthwhile discussing the required specifications for the algorithm to be correct. Such requirements include:

- Jump to be in “counter-clockwise” direction i.e. that $\alpha \leq \pi$ (this breaks the symmetry for the adversarial placements of the fence and the treasure)
- Chord jump does not pass over the area that could hold the treasure, landing back to the fence ($\alpha \leq 2\pi - \beta$)

To summarize, the 1-Jump Algorithm is fully determined by choosing α satisfying

$$0 < \alpha \leq \min\{\pi, 2\pi - \beta\}.$$

To resume, Algorithm 4.1 with parameter α runs similarly to Algorithm 2.1, except from the case in which the last basic landing point of Algorithm 2.1 is within the fence. On the one hand, if that happens, Algorithm 4.1 makes a counter-clockwise jump corresponding to arc length α . On the other hand, if the 1st-jump landing point is in the fence, then it runs Algorithm 2.1. Otherwise, the 1st-jump landing point is outside the fence, the robot applies the *remedy phase*, mentioned in Chapter 2, and continues executing Algorithm 2.1.

Algorithm 4.1 1-Jump

Walk to the perimeter of the disk

if your landing point is inside the fence **then**

 make a ccw chord jump of arc length α

end if

Perform Algorithm 2.1

Figure 4.1: 1-Jump Algorithm basic landing point.

Lemma 4.1.1. *Depending on the landing points, the cost of Algorithm 4.1 with parameter α is:*

$$\left\{ \begin{array}{ll} c_0^1 := 1 + 2\pi - \beta + 2 \sin(\beta/2) & \text{Figure 2.1a} \\ c_1^1 := c_0^1 + 4 \sin(\alpha/2) - 2 \sin(\beta/2) & \text{Figure 4.1a} \\ c_2^1 := 1 + 2\pi - (\alpha - 2 \sin(\alpha/2)) & \text{Figure 4.1b} \end{array} \right. \quad (4.1)$$

with the understanding that c_0^1, c_1^1, c_2^1 are functions on β and α , and the reference in the right column indicates the Figure which applies to the case.

Proof. It is clear that, if the basic landing point is outside the fence, then the cost of Algorithm 4.1 c_0^1 is equal to cost c_0^0 of the Algorithm 2.1 (for the same case).

Let us now suppose that the basic landing point falls inside of the fence. Algorithm 4.1 performs a counter-clockwise chord jump of length $2 \sin(\alpha/2)$. We examine two more subcases.

- The 1st-jump landing point is outside the fence as seen in Figure 4.1a, say at clockwise distance $x \in (0, \alpha)$ from the fence.

The robot follows the remedy phase spending $x + 2 \sin(x/2)$ more time to come back to the same landing point. Clearly, the worst positioning of the treasure is to be arbitrarily clockwise close to the fence. That would make the robot search for an additional time of $2\pi - \beta - x$ for a total of:

$$1 + x + 2 \sin(x/2) + 2\pi - \beta - x = 1 + 2 \sin(x/2) + 2\pi - \beta.$$

Since $x \leq \alpha \leq \min\{\pi, 2\pi - \beta\}$ and by the monotonicity of $\sin(x/2)$ we can observe that the cost in that case is no more than:

$$\begin{aligned} \sup_{0 < x < \alpha} \{1 + 2 \sin(\alpha/2) + 2 \sin(x/2) + 2\pi - \beta\} \\ &= 1 + 4 \sin(\alpha/2) + 2\pi - \beta \\ &= c_0^1 + 4 \sin(\alpha/2) - 2 \sin(\beta/2) \end{aligned}$$

- The 1st-jump landing point is in the fence and is illustrated in Figure 4.1b.

The worst configuration in this case is when the robot's basic landing point is close to the clockwise endpoint of the fence, with the treasure close to the same endpoint but outside the fence. The robot would then have to traverse the whole disk, saving only an arc of length α which is jumped over using the corresponding chord of length $2 \sin(\alpha/2)$. The cost, in this case, becomes $1 + 2\pi - \alpha + \sin(\alpha/2)$

□

Let us consider expression $\alpha + 2 \sin(\alpha/2)$ which is monotonically increasing. As such, for every $\beta \in \mathbb{R}$, the equation $\alpha + 2 \sin(\alpha/2) = \beta$ admits a unique solution in

α . Using this observation we write that : “ α_β is the unique real number satisfying equation $\alpha_\beta + 2 \sin(\alpha_\beta/2) = \beta$ ”.

Moreover, since $\alpha + 2 \sin(\alpha/2)$ is increasing in the variable α , so is α_β in the variable β . We are now ready to define and analyze the optimal 1-Jump Algorithm.

Theorem 4.1.2. *Let γ be the unique solution to equation $\pi = \gamma - \sin(\gamma/2)$ ($\gamma \approx 4.04196$). The optimal 1-Jump Algorithm chooses a jump corresponding to arc length $\alpha = \alpha_\beta$ if $\beta \leq \gamma$, $\alpha = 2\pi - \beta$ if $\beta > \gamma$ and terminates in time*

$$1 + \begin{cases} 2\pi - \alpha_\beta + 2 \sin(\alpha_\beta/2) & \text{if } \beta \leq \gamma \\ \beta + 2 \sin(\beta/2) & \text{if } \beta > \gamma. \end{cases} \quad (4.2)$$

Proof. By Lemma 4.1.1, the optimal 1-Jump Algorithm is determined by choosing α that minimizes:

$$\sup_{0 < \alpha < \min\{\pi, \pi - \beta\}} \{c_0^1, c_1^1(\alpha), c_2^1(\alpha)\},$$

where α 's dependence is explicit. Again, it should be clear that having the basic landing point outside the fence is a “favorable event”. Intuitively, this is the only case that the robot makes full use of the fact that the treasure does not lie within the fence, jumping over it and using the corresponding chord. Effectively, this implies that for all β, α we have $c_0^1 \leq \min\{c_1^1(\alpha), c_2^1(\alpha)\}$.

Next, for any $\beta \in (0, 2\pi)$ we need to choose α so as to minimize $\max\{c_1^1(\alpha), c_2^1(\alpha)\}$. To that end, note that c_1^1, c_2^1 exhibit different monotonicities with respect to α so that, if possible, the minimum will be attained when the two costs are equal. Equating the two costs gives that $\alpha + 2 \sin(\alpha/2) = \beta$. Recall that we have denoted the unique solution to the equation by α_β which is increasing in β . Since the jump step needs to stay no more than $\min\{\pi, 2\pi - \beta\}$, the choice $\alpha = \alpha_\beta$ (which is the best possible) is

valid only when $\alpha_\beta \leq \min\{\pi, 2\pi - \beta\}$. Numerically we can compute $\alpha_\pi \approx 1.66$, which due to the monotonicity of α_β implies that the dominant constraint is that $\alpha_\beta \leq 2\pi - \beta$, and hence any restrictions will be imposed for $\beta > \pi$. Indeed, setting $\alpha_\beta = 2\pi - \beta$, and substituting in $\alpha_\beta + 2 \sin(\alpha_\beta/2) = \beta$ we obtain $2\pi - \beta + 2 \sin(\pi - \beta/2) = \beta$. The value of β that satisfies this equation is $\gamma \approx 4.04196$.

To resume, as long as $\beta \leq \gamma$, the best choice for the jump is the solution to the equation $\alpha + 2 \sin(\alpha/2) = \beta$. When $\beta > \gamma$, the best jump step is equal to $2\pi - \beta$.

From the discussion above, the induced cost when $\beta \leq \gamma$ would be equal to $c_1^1(\alpha_\beta)$, as it reads in Lemma 4.1.1. Finally, when $\beta > \gamma$ the induced cost would be

$$\begin{aligned}
& \max\{c_1^1(2\pi - \beta), c_2^1(2\pi - \beta)\} \\
&= 1 + 2\pi + \max\{4 \sin(\pi - \beta/2) - \beta, 2 \sin(\pi - \beta/2) - 2\pi + \beta\} \\
&= 1 + 2\pi + 2 \sin(\beta/2) + \max\{2 \sin(\beta/2) - \beta, -2\pi + \beta\} \\
&= 1 + \beta + 2 \sin(\beta/2)
\end{aligned}$$

where the last equality is due to that $\beta \geq \gamma$, the definition of γ and the fact that $-2\pi + \beta$ is increasing in β . This proves Theorem 4.1.2. $\square$

Notably, the proof of Theorem 4.1.2 suggests that for the best strategy α as a function of β , we have that $c_0^1 \leq c_2^1(\alpha) \leq c_1^1(\alpha)$. This was expected, since we are aware that having the basic landing outside the fence is more favourable than having it inside the fence. It also does not require the remedy phase, which is, evidently, more favourable than needing the remedy phase. It is also interesting to note that for $\beta \leq \gamma$, the best jump choice α_β attains values close to $\beta/2$. This suggests an alternative approach to the problem that does not require the ability to solve technical trigonometric equations, and that will be explored later. Finally, there is a recursive relation between costs c_0^1, c_1^1 that is soon to be generalized for k-Jump Algorithms.

4.2 Generic Description of k -Jump Algorithms

Analogously to the previous sections, the k -Jump Algorithm has parameters $\alpha_1, \dots, \alpha_k$ and runs similarly to the $(k-1)$ -Jump Algorithm, except in the case where the last landing point of the $(k-1)$ -Jump Algorithm (which happens to be the $(k-1)$ st-jump landing point) is within the fence. If that happens, the k -Jump Algorithm makes an additional counter-clockwise jump corresponding to arc length α_k . If the k th-jump landing point is in the fence, then it runs Algorithm 2.1. Otherwise, the k th-jump landing point is outside the fence, and the robot applies the remedy phase from Algorithm 4.1 in Section 4.1.

Algorithm 4.2 k -Jump Algorithm

- 1: Walk to the perimeter of the disk
 - 2: $i \leftarrow 0$
 - 3: **while** landing point is inside the fence & $i < k$ **do**
 - 4: $i \leftarrow i + 1$
 - 5: make a ccw chord jump of arc length α_i
 - 6: **end while**
 - 7: Perform Algorithm 2.1
-

It is clear from the discussion above that any k -Jump Algorithm is specified by the jump steps $\alpha_1, \alpha_2, \dots, \alpha_k$, where the i -th jump is realized only if the basic landing point, along with the landing points of the previous $i - 1$ jumps fall within the fence. In order to preclude the possibility that a jump passes over the area that holds the treasure and bring the robot back to the fence, we add the requirement that $\alpha_i \leq \beta$. Moreover, for the jumps to be in counter-clockwise direction (and to break the symmetry) we further require that $\alpha_i \leq \pi$.

Similarly, for the 1-Jump Algorithm, we required that $\alpha_1 \leq \min\{\pi, 2\pi - \beta\}$.

However, according to Theorem 4.1.2, the optimal jump step is less than β (for all β), meaning that the correctness condition for choosing the jump step could have been replaced by $\alpha_1 \leq \min\{\beta, 2\pi - \beta\}$. Indeed, our intuition tells us that an algorithm, which after the basic landing point falls within the fence, makes a jump more than the length of the fence, will land outside the fence and subsequently will need to apply the (suboptimal) remedy phase. Motivated by this observation, we require the following condition regarding the step sizes of k-Jump Algorithm's: $\alpha_i \leq \min\{\pi, 2\pi - \beta\}$, $i = 1, \dots, k$.

The next lemma generalizes Lemma 4.1.1 and provides a handy recurrence description of the cost of the k-Jump Algorithm with jump steps $\alpha_1, \dots, \alpha_k$ depending on the first landing point outside the fence. In this direction, we denote by c_t^k to be the worst case cost of the k-Jump Algorithm when the basic landing point along with the landing points of the first $t - 1$ jumps fall all inside the fence and the robot lands outside the fence in the the t th jump, which is shown in Figure 4.2. Let us observe that, c_0^k is the cost of the case when the basic landing point is outside the fence, while c_{k+1}^k corresponds to the case that the landing points of all k jumps, as well as the basic landing point, fall inside the fence.

Lemma 4.2.1. *For any β , let $\alpha_0 = \beta$. Depending on the landing points, the cost of the k-Jump Algorithm with jump steps $\alpha_1, \dots, \alpha_k$ is*

$$\begin{cases} c_0^k &:= 1 + 2\pi - \alpha_0 + 2 \sin(\alpha_0/2) \\ c_t^k &:= c_{t-1}^k + 4 \sin(\alpha_t/2) - 2 \sin(\alpha_{t-1}/2) \\ c_{k+1}^k &:= 1 + 2\pi - \sum_{i=1}^k (\alpha_i - 2 \sin(\alpha_i/2)) \end{cases} \quad (4.3)$$

Figure 4.2: k-Jump Algorithm

with the understanding that c_t^k are functions on β and $\alpha_1, \dots, \alpha_{t-1}$, for $t = 1, \dots, k+1$.

Proof. As previously mentioned, when the basic landing point is outside the fence, the cost is indeed $c_0^k = c_0^0$. Furthermore, when all landing points, including the basic one, fall within the fence, then similarly to the cost c_2^1 of Lemma 4.1.1, the worst positioning of the fence makes the basic landing point inside and arbitrarily close to the counterclockwise endpoint of the fence. Meanwhile, the treasure is arbitrarily close to the same endpoint but outside the fence. Effectively, in this case, the robot will traverse the entire disk in a counter-clockwise direction, saving exactly $\alpha_i - 2 \sin(\alpha_i/2)$, $i = 1, \dots, k$ from each jump.

For the most interesting case, we need to compare the costs c_t^k, c_{t-1}^k , for some $t \in \{1, \dots, k\}$. In both cases, the worst positioning of the treasure is arbitrarily close to the clockwise endpoint of the fence. The worst positioning of the basic landing point should bring the robot inside, as well as arbitrarily close to the counterclockwise endpoint of the fence, so as to induce the maximum possible remedy phase cost. Note that for the case of cost c_t^k , the robot traverses twice the chord of length $2 \sin(\alpha_t/2)$, but only once the chord of length $2 \sin(\alpha_{t-1}/2)$. Other than that, in both cases, the

robot perform exactly the same jumps, and search exactly the same subperimeter of the circle. This proves Lemma 4.2.1. $\square$

4.3 The Halving Heuristic k -Jump Algorithm

In this section, we present a simple heuristic that is meant to approximate the optimal jump steps without relying on solutions of trigonometric optimization problems. Most importantly, our algorithm requires very limited memory and does not need to perform numerical operations other than simple algebraic manipulations. In fact, there are only constant many operations needed to determine every possible jump size. Moreover, parameter k , i.e. the number of jumps, may not necessarily be determined in advance, and is allowed to be even infinite. First, we present the algorithm and analyze it. Then, in Section 4.4, we contrast it to the Optimal k -Jump Algorithm (for certain values of k).

Closely examining the optimal solution for the 1-Jump Algorithm in Section 4.1, we are tempted to choose an alternative first jump step equal to $\beta/2$, which is a good approximation to α_β . This choice is valid, as long as the jump does not exceed $2\pi - \beta$, and indeed for large enough values of β , i.e. for $\beta \geq 4.041$, as per Theorem 4.1.2, the best choice for just one jump is $2\pi - \beta$. Note that changing the first jump from α_β to $\beta/2$ results to a new threshold value $\frac{4}{3}\pi \approx 4.188$ after which the first jump should become $2\pi - \beta$. Interestingly, the pattern repeats also in the optimal k -Jump Algorithms (see Section 4.4).

The previous observation suggests a natural heuristic for k -Jump Algorithms. First, go to an arbitrary point on the circle. While in (some unknown position in) the fence, make a valid jump (i.e. no more than $\pi, 2\pi - \beta$) equal to half of the unexplored fence, unless this value exceeds $2\pi - \beta$ in which case the jump should be $2\pi - \beta$.

Formally, the description of the heuristic follows if we can determine the length of the chord-jump α_i in every i -th jump, and then invoke Algorithm 4.2.

Algorithm 4.3 Halving Heuristic jumps

```

1:  $explored \leftarrow 0$ 
2:  $temp \leftarrow \frac{\beta - explored}{2}$ 
3: if  $temp \leq 2\pi - \beta$  then
4:    $\alpha_i \leftarrow temp$ 
5: else
6:    $jump \leftarrow 2\pi - \beta$ 
7: end if
8:  $explored \leftarrow explored + jump$ 

```

Note that the calculations of jumps α_i can be incorporated within Algorithm 4.2 and do not need to be computed in advance. As the maximum number of jumps can be part of the input, Algorithm 4.3 can be performed only for k many landings within the fence (see Algorithm 4.2), or as long as the the jump step does not drop below a given threshold. Interestingly, the definition of step sizes on the fly by Algorithm 4.3 evenn allows for $k = \infty$. That would correspond to the theoretical case that the robot makes an infinite number of jumps for which all landings happen within the fence. Still, the time for the robot to reach the endpoint of the fence would be finite (Zeno's paradox).

The process above fully determines the jump step of the t -th jump as a function of β , for every $t = 1, \dots, k$, and for every k . In what follows we provide an analytic description of these values so that we can analyze the performance of the algorithm. The lemma below will allow us to derive later a nicer closed formula for the jump steps of the halving Algorithm.

Lemma 4.3.1. *Let $h_t = \frac{2\pi(t+1)}{t+2}$ for $t \geq 1$ and $h_0 = 0$. For any $\beta \in (0, 2\pi)$, the value of the i -th jump in the Halving Algorithm equals*

$$\alpha_i = \begin{cases} \frac{j\beta - (j-1)2\pi}{2^{i-j+1}} & \text{if } \beta \leq h_i \text{ and } \beta \in (h_{j-1}, h_j] \\ 2\pi - \beta & \text{if } \beta > h_i \end{cases}$$

Proof. We will derive the promised formulas from scratch, without relying on the statement of the lemma. First, note that the process above defines natural threshold values h_i for β , after which the i -th jump step α_i becomes $2\pi - \beta$. In particular, the value of α_i will depend on which interval $(h_{j-1}, h_j]$ value β belongs to, where $j = 1, \dots, k$, and with the understanding that $\alpha_i = 2\pi - \beta$ if $\beta > h_i$. Therefore, it is natural to introduce notation

$$A_{i,j} := \alpha_i, \quad \text{when } \beta \in (h_{j-1}, h_j]$$

It is easy to see that if $A_{i,j} = 2\pi - \beta$ then $A_{r,j} = 2\pi - \beta$ for all $r = 1, \dots, i-1$, and in general that $A_{i,j} = 2\pi - \beta$ whenever $j \geq i+1$. In other words, $A(i, i)$ is the last expression (in β) for α_i before it becomes $2\pi - \beta$, while all previous jump steps should be equal to $2\pi - \beta$. Since at every step, the algorithm attempts a jump of half the unexplored fence, right before the i -th jump there has been explored a total of $(i-1)(2\pi - \beta)$ part of the fence. Hence,

$$A_{i,i} = \frac{\beta - (i-1)(2\pi - \beta)}{2} = \frac{i\beta - (i-1)2\pi}{2}$$

The threshold h_i is determined by requiring that $A_{i,i} \leq 2\pi - \beta$, from which we obtain

that

$$h_i = \frac{2\pi(i+1)}{i+2}$$

which is indeed increasing in i .

Our next claim is that

$$A_{i,j} = \frac{j\beta - (j-1)2\pi}{2^{i-j+1}}, \text{ for all } i \geq j.$$

The proof is by induction on $i - j$. Indeed, the claim is true when $i = j$. So assume that $i = j + t$ for some $t \geq 1$. The explored part of the fence up to the first $(i - 1)$ jumps is equal to

$$\begin{aligned} \sum_{r=1}^{i-1} A_{r,j} &= \sum_{r=1}^{j-1} A_{r,j} + \sum_{r=j}^{j+t-1} A_{r,j} \\ &= (j-1)(2\pi - \beta) + \sum_{r=j}^{j+t-1} \frac{j\beta - (j-1)2\pi}{2^{r-j+1}} \\ &= (j-1)(2\pi - \beta) + \left(1 - \frac{1}{2^t}\right) (2\pi(j-1) - j\beta) \\ &= \frac{\beta(2^t - j) + 2\pi(j-1)}{2^t}. \end{aligned}$$

According to the Halving Algorithm, the i -th jump step will be exactly half of the unexplored fence, if that value does not exceed h_j . Indeed, the candidate step size is

$$\begin{aligned} \frac{\beta - \frac{\beta(2^t - j) + 2\pi(j-1)}{2^t}}{2} &= \frac{j\beta - (j-1)2\pi}{2^{t+1}} \\ &= \frac{j\beta - (j-1)2\pi}{2^{i-j+1}} \\ &= A_{i,j}. \end{aligned}$$

Finally, for this jump to be valid, we need to show that $0 < A_{i,j} \leq 2\pi - \beta$. To

that end, recall that $A_{i,j}$ corresponds to the i -th jump when $h_{j-1} < \beta \leq h_j$, i.e. when $\frac{2\pi j}{j+1} < \beta \leq \frac{2\pi(j+1)}{j+2}$. Note that we are in the case where $i \geq j$, and so we have $\beta > \frac{2\pi j}{j+1} > \frac{2\pi(j-1)}{j}$ and hence $A_{i,j} > 0$. Also,

$$\begin{aligned} 2\pi - \beta - A_{i,j} &\geq 2\pi - \frac{2\pi(j+1)}{j+2} - \frac{j\beta - (j-1)2\pi}{2^{i-j+1}} \\ &\geq 2\pi - \frac{2\pi(j+1)}{j+2} - \frac{j\beta - (j-1)2\pi}{2} \\ &= \frac{j(2\pi(j+1) - \beta(j+2))}{2(j+2)}, \end{aligned}$$

which is again non negative, since $\beta \leq \frac{2\pi(j+1)}{j+2}$, as wanted. This proves Lemma 4.3.1. $\square$

We are now ready to present a closed formula for the jump steps of the Halving Algorithm along with its performance, as a function on the number of jumps.

Theorem 4.3.2. *For any $\beta \in (0, 2\pi)$, let $\rho_\beta := \max \left\{ \frac{2\beta-2\pi}{2\pi-\beta}, 1 \right\}$. The value of the i -th jump ($i \geq 1$) in the Halving Algorithm equals*

$$\alpha_i = \begin{cases} 2\pi - \beta & \text{if } i < \rho_\beta \\ \frac{2\pi - \lceil \rho_\beta \rceil (2\pi - \beta)}{2^{i - \lceil \rho_\beta \rceil + 1}} & \text{otherwise} \end{cases}$$

Proof. According to Lemma 4.3.1, $\beta > h_i$ is satisfied as long as $i < \frac{2\beta-2\pi}{2\pi-\beta}$. Hence, if $i < \rho_\beta$ we have $\alpha_i = 2\pi - \beta$. For the same reason $\beta \leq h_i$ if and only if $i \geq \rho_\beta$, so ρ_β is the smallest integer for which $\beta \leq h_{\rho_\beta}$, meaning that $\beta \in (h_{\rho_\beta-1}, h_{\rho_\beta}]$. Therefore, again by Lemma 4.3.1, we set $j = \lceil \rho_\beta \rceil$ (and rearrange the terms) to derive the promised formula. This proves Theorem 4.3.2. $\square$

In Figure 4.3, we depict the behaviour of the decreasing sequence α_i (in i) as a function of β for a $i = 1, 2, 3, 4, 5$ and 6. Notably, for every k there is some threshold

value of β , after which $\alpha_i = 2\pi - \beta$ for all $i \leq k$.

Figure 4.3: The plot jumps $\alpha_1 \geq \alpha_2 \geq \dots \geq \alpha_6$ of the Halving Algorithm as a function of β .

Finally, we use the closed formula for the jump steps to derive a closed formula for the cost of the Halving k -Jump Algorithm. The main idea for the proof is to show that the worst case for the algorithm is when all k jumps fall within the fence, and that the performance is strictly decreasing in k . This is what the next lemma establishes.

Lemma 4.3.3. *The Halving k – Jump Algorithm incurs the maximum possible cost when all jump landings (including the basic one) fall within the fence.*

Proof. For the values of α_i as defined in Theorem 4.3.2, we will show that the worst configuration is when all k jump landings (together with the basic one) fall within the fence. In the language of Lemma 4.2.1 we will show that $c_t^k < c_{t+1}^k$ for all $t = 0, k$. Also note that c_{k+1}^k is decreasing in k , in fact no matter what the jump steps are, since $x - \sin(x/2) > 0$, for all $x > 0$, which concludes the lemma.

As already claimed, it is immediate that $c_0^k < c_i$, for all i , since the cost c_0^k is incurred exactly when the basic landing falls outside the fence. Due to the fact that

the robot has knowledge of the length β , the robot can fully avoid the fence by jumping over it. Next, according to Lemma 4.2.1 we have that

$$c_t^k - c_{t-1}^k = 4 \sin(\alpha_t/2) - 2 \sin(\alpha_{t-1}/2),$$

for all $t = 1, \dots, k$. In particular, for $t < \rho_\beta$ the jump steps remain equal to $2\pi - \beta > 0$, and hence $c_t^k - c_{t-1}^k > 0$ for all $t < \rho_\beta$.

When $t = \lceil \rho_\beta \rceil$ we have

$$\begin{aligned} c_{\lceil \rho_\beta \rceil}^k - c_{\lceil \rho_\beta \rceil - 1}^k &= 4 \sin(\alpha_{\lceil \rho_\beta \rceil}/2) - 2 \sin((2\pi - \beta)/2) \\ &= 4 \sin\left(\frac{\pi}{2} - \frac{\lceil \rho_\beta \rceil}{4}(2\pi - \beta)\right) - 2 \sin(\beta/2) \\ &\geq 4 \sin\left(\frac{\pi}{2} - \frac{\rho_\beta}{4}(2\pi - \beta)\right) - 2 \sin(\beta/2) \\ &\geq 4 \sin\left(\frac{\pi}{2} - \frac{\frac{2\beta - 2\pi}{2\pi - \beta}}{4}(2\pi - \beta)\right) - 2 \sin(\beta/2) \\ &= 4 \sin(\beta/2) - 2 \sin(\beta/2) \geq 0. \end{aligned}$$

When $t \geq \lceil \rho_\beta \rceil + 1$, the jump steps drop by a factor of two in each iteration. Since for all $x > 0$ we have that $2 \sin(x/4) - \sin(x/2) > 0$, we obtain easily that $c_t^k - c_{t-1}^k > 0$, for $t = \rho_\beta + 1, \dots, k$.

Hence it remains to show that $c_{k+1}^k > c_k$. To that end assume that $k \geq \lceil \rho_\beta \rceil$.

Then we have

$$\begin{aligned}
c_{k+1}^k - c_k^k &= \beta - \sum_{i=1}^k \alpha_i - 2 \sin(\alpha_k/2) \\
&= \beta - \left(([\rho_\beta] - 1)(2\pi - \beta) + \sum_{i=0}^{k-[\rho_\beta]} \frac{\alpha_{[\rho_\beta]}}{2^i} \right) - 2 \sin(\alpha_k/2) \\
&= \beta - \left(([\rho_\beta] - 1)(2\pi - \beta) + 2\alpha_{[\rho_\beta]} \left(1 - \frac{1}{2^{k-[\rho_\beta]+1}} \right) \right) - 2 \sin(\alpha_k/2) \\
&= \beta - \left(([\rho_\beta] - 1)(2\pi - \beta) + 2\left(\pi - \frac{[\rho_\beta]}{2}(2\pi - \beta)\right) \left(1 - \frac{1}{2^{k-[\rho_\beta]+1}} \right) \right) \\
&\quad - 2 \sin(\alpha_k/2) \\
&= \frac{2\pi - [\rho_\beta](2\pi - \beta)}{2^{k-[\rho_\beta]+1}} - 2 \sin(\alpha_k/2).
\end{aligned}$$

Recall that $\alpha_k = \frac{2\pi - [\rho_\beta](2\pi - \beta)}{2^{k-[\rho_\beta]+1}}$, so the last expression is non negative since $x - \sin(x/2) > 0$ for all $x > 0$. Finally, for the case that $k < [\rho_\beta]$, and since k is an integer, we have $k \leq \rho_\beta$, and so

$$\begin{aligned}
c_{k+1}^k - c_k^k &= \beta - k(2\pi - \beta) - 2 \sin(\beta/2) \\
&\geq \beta - \rho_\beta(2\pi - \beta) - 2 \sin(\beta/2) \\
&= 2\pi - \beta - 2 \sin(\beta/2).
\end{aligned}$$

The last expression is non negative since $\sin(\beta/2) = \sin((2\pi - \beta)/2)$ and $x - 2 \sin(x/2) > 0$, for all $x > 0$. $\square$

We are ready to conclude with the cost of the Halving k-Jump Algorithm.

Theorem 4.3.4. *The cost of the Halving k-Jump Algorithm is strictly decreasing with*

k and it equals

$$1 + 2\pi + \frac{\lceil \rho_\beta \rceil (2\pi - \beta) - 2\pi}{2^{k - \lceil \rho_\beta \rceil + 1}} + 2(\lceil \rho_\beta \rceil - 1) \sin(\beta/2) + 2 \sum_{i=0}^{k - \lceil \rho_\beta \rceil} \sin\left(\frac{\alpha_{\lceil \rho_\beta \rceil}}{2^{i+1}}\right),$$

where $\rho_\beta = \max\left\{\frac{2\beta - 2\pi}{2\pi - \beta}, 1\right\}$ and $\alpha_{\lceil \rho_\beta \rceil} = \pi - \frac{\lceil \rho_\beta \rceil}{2}(2\pi - \beta)$.

Proof. Using the terminology of Lemma 4.2.1, and by Lemma 4.3.3, the cost of the Halving k-Jump Algorithm equals

$$c_{k+1}^k = 1 + 2\pi - \beta - \sum_{i=1}^k (\alpha_i - 2 \sin(\alpha_i/2))$$

where the jump steps are as determined in Theorem 4.3.2. From the expression above, it is immediate that the cost is strictly increasing in k , as long as all jump steps are positive. Next, we compute the summation in parts. We have,

$$\begin{aligned} \beta + \sum_{i=1}^k \alpha_i &= \beta + \sum_{i=1}^{\lceil \rho_\beta \rceil - 1} \alpha_i + \sum_{i=\lceil \rho_\beta \rceil}^k \alpha_i \\ &= \beta + (\lceil \rho_\beta \rceil - 1)(2\pi - \beta) + \sum_{i=0}^{k - \lceil \rho_\beta \rceil} \frac{\alpha_{\lceil \rho_\beta \rceil}}{2^i} \\ &= \beta + (\lceil \rho_\beta \rceil - 1)(2\pi - \beta) + \alpha_{\lceil \rho_\beta \rceil} \left(2 - \frac{1}{2^{k - \lceil \rho_\beta \rceil}}\right) \\ &= \beta + (\lceil \rho_\beta \rceil - 1)(2\pi - \beta) + \left(\pi - \frac{\lceil \rho_\beta \rceil}{2}(2\pi - \beta)\right) \left(2 - \frac{1}{2^{k - \lceil \rho_\beta \rceil}}\right) \\ &= \frac{\lceil \rho_\beta \rceil (2\pi - \beta) - 2\pi}{2^{k - \lceil \rho_\beta \rceil + 1}}. \end{aligned}$$

Finally,

$$\begin{aligned} \sum_{i=1}^k \sin(\alpha_i/2) &= \sum_{i=1}^{\lceil \rho_\beta \rceil - 1} \sin(\alpha_i/2) + \sum_{i=\lceil \rho_\beta \rceil}^k \sin(\alpha_i/2) = (\lceil \rho_\beta \rceil - 1) \sin(\beta/2) \\ &\quad + \sum_{i=0}^{k - \lceil \rho_\beta \rceil} \sin\left(\frac{\alpha_{\lceil \rho_\beta \rceil}}{2^{i+1}}\right) \end{aligned}$$

Putting the two expression together gives the promised cost. This proves Theorem 4.3.4. $\square$

Figure 4.4 summarizes the cost of the Halving k-Jump Algorithm for $k = 1, 2, 3$ and 4.

Figure 4.4: The performance of the Halving Algorithm for 1,2,3 and 4 jumps (decreasing in the number of jumps, respectively) as a function of β .

4.4 Some Optimal k-Jump Algorithms & Comparison

It is apparent from Lemma 4.2.1 that choosing the optimal jump steps $\alpha_1, \dots, \alpha_k$ amounts to solving the involved optimization problem $\min_{\alpha_1, \dots, \alpha_k} \max_{t=1, \dots, k+1} \{c_t^k\}$,

where $\alpha_i \leq \min\{\pi, 2\pi - \beta\}$. In this section we contrast the choices and performance of the Halving k -Jump Algorithm with the choices and performance of the Optimal k -Jump Algorithm that was obtained using optimization software packages, for $k \leq 3$ (except from $k = 1$ whose formal analysis appears in Section 4.1). Our findings are summarized in the figures below.

Figure 4.5: Performance comparison between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β .

Figure 4.6: Comparison of jump choices between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β .

Figure 4.7: Performance comparison between the Optimal 1-Jump Algorithm and the Halving 1-Jump Algorithm, as a function of β .

Figure 4.8: Comparison of the two jump choices between the Optimal 2-Jump Algorithm and the Halving 2-Jump Algorithm, as a function of β . The first jump of each algorithm is always no smaller than the second one, and eventually they all attain the value $2\pi - \beta$.

For $k = 1, 2, 3$ we numerically compute the optimal k -Jump Algorithm (note that for $k = 1$ the rigorous analysis appears in Section 4.1). Then, we contrast the performance of the optimal and of the Halving algorithm (for the same number of jumps), as well as contrasting their corresponding jump steps. The numerical calculations indicate that the choices of the Halving algorithm are nearly optimal for a wide spectrum of β (with the largest discrepancy for $\beta \approx \gamma$). Interestingly, for larger

Figure 4.9: Comparison of the three jump choices between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β . For both algorithms, the first jump of each algorithm is always no smaller than the second one, which is no smaller than the third one. Eventually they all attain the value $2\pi - \beta$.

Figure 4.10: Performance comparison between the Optimal 3-Jump Algorithm and the Halving 3-Jump Algorithm, as a function of β . Notably, performance is nearly the same for all values of $\beta > 5$. The bigger discrepancy is observed for values of β close to 4, for which also the jump steps between the two algorithms exhibit the larger gaps (see Figure 4.9).

values of β , the choices of the Halving algorithm are nearly optimal that also reflects on the cost of the two algorithms which becomes nearly identical. More importantly, experiments indicate that for large values of β , the optimal choices for k jump steps is to make all equal to $2\pi - \beta$, which is also the choice of the Halving algorithm.

4.5 Summary

In this chapter, we examined several algorithms with an attempt to find more optimal jumps if initial landing point is within the fence. First we determined the optimal 1-Jump algorithm for the robot to find the target. Then we provided a generic description of k -Jump algorithms and analyzed their cost depending on the jump landings. Subsequently we analyzed the Halving k -Jump algorithms, where k is the max number of jumps the robot makes so as to overcome the fence and find the target.

Chapter 5

Randomized Algorithms

In this section we explore how randomness can be used to improve the performance of our algorithmic. Intuitively, random algorithmic choices reduce the power of the adversary, and as such, the (expected) worst-case performance will be reduced.

In order to exhibit the power of randomness, we will restrict ourselves to variations of Algorithm 2.1, i.e. algorithms that perform no jumps. Still, we will be able to improve performance significantly.

Taking the 0-Jump Algorithm into account, we will first examine the addition of 1-bit randomness. During this modification of the 0-Jump Algorithm, when the robot first lands on the perimeter (basic landing point), the robot has the ability to choose either a clockwise or counter-clockwise direction at random. We will call this the R-Robot Direction Algorithm and will provide an analysis of the worst case termination time.

In the following section, we will allow the robot to use infinite randomness, meaning the robot will fall on a random point on the perimeter. This randomized algorithm will be named the R-Fence Walk Algorithm, and similarly to the R-Robot Direction Algorithm, we will provide a worst case termination time. We will then provide a

comparison on the worst case termination of the two randomized events and modifications to the algorithms in the form of the RD-Fence Walk Algorithm.

5.1 Single Random-Bit Algorithm

Since randomness is considered a computational resource, we are motivated to explore a randomized algorithm, Algorithm 5.1, that uses a single random bit. One way of exploring a random bit is to run Algorithm 2.1 with the modification that the exploration of the perimeter of the disk is either clockwise or counter-clockwise, each with probability $1/2$.

Algorithm 5.1 R-Robot Direction

- 1: Walk to the perimeter of the disk
 - 2: Start walking on the perimeter, either in cw or in ccw direction, each with probability $1/2$.
 - 3: Perform steps 3-7 of Algorithm 2.1.
-

Theorem 5.1.1. *The worst case expected termination time of Algorithm 5.1 is $1 + \pi$, i.e. it is independent of the length β of the fence.*

Proof. We are aware that the random choice of Algorithm 5.1 involves only the direction of the robot's movement (ccw or cw). In particular, an adversary can still choose where the robot will land, as well as choosing how far away the treasure is from the fence. The worst case (expected) termination time over these choices will be the algorithm's performance. For notational convenience, we omit the deployment cost of 1 in our calculations, i.e. the time it takes the robot to reach the perimeter.

Clearly, the adversary can choose to have the robot land either outside the fence, or inside the fence. In each case we will determine the worst positioning of the treasure,

and then we will find the maximum of the two. We denote X be the random variable that equals the termination time for the associated experiment, over the random choice of the algorithm.

Figure 5.1: The Robot randomly chooses a direction to move in.

- Adversary chooses to have the robot land outside the fence (See 5.1a).

Let x denote the clockwise distance between the robot and the fence and let d denote the clockwise distance between the treasure and the fence, with $d \geq 0$. Depending on the values of x, d , together with the random choice of the algorithm, the robot will first either encounter the treasure or the fence. Without loss of generality we may assume that $x + d + \beta \leq 2\pi$, and then we may focus only on the random choice of the algorithm. Thus, let A denote the event that the fence is encountered first (5.1a ccw movement). It is easy to see that the robot will jump over the fence, avoiding the cost of walking along the fence perimeter. We define A to be the event in which the robot encounters the fence

before the treasure. The termination time is:

$$E[X|A] = x + 2 \sin\left(\frac{\beta}{2}\right) + d \quad (5.1)$$

For the complementary event, A^c , the robot encounters the treasure first (5.1a cw movement). The incurred cost in this case is:

$$\mathbb{E}[X|A^c] = 2\pi - \beta - d - x \quad (5.2)$$

Overall, if the adversary makes the robot land outside the fence, then the worst case expected running time is:

$$\begin{aligned} \mathbb{E}[X] &= \max_{x, d: x+d+\beta \leq 2\pi, \text{ \& } x, d \geq 0} \{\mathbb{P}[A] \cdot \mathbb{E}[x|A] + \mathbb{P}[A^c] \cdot \mathbb{E}[x|A^c]\} \\ &= \frac{1}{2} \left(2\pi - \beta + 2 \sin\frac{\beta}{2} \right) \end{aligned} \quad (5.3)$$

- Adversary chooses to have the robot land inside the fence (See 5.1b).

As before, let x denote the ccw distance of the robot to the end of the fence and let d be the ccw distance between the fence and the treasure. Clearly, we have that $0 \leq x \leq \beta$, and $0 \leq d \leq 2\pi - \beta$. We now let D denote the event that the robot moves ccw. It is immediate that

$$\mathbb{E}[X|D] = \beta - x + d \quad (5.4)$$

and that

$$\mathbb{E}[X|D^c] = x + 2\pi - \beta - d \quad (5.5)$$

So, the worst case expected termination time becomes

$$\begin{aligned}\mathbb{E}[X] &= \max_{x, d: 0 \leq x \leq \beta, \text{ \& } 0 \leq d \leq 2\pi - \beta} \{\mathbb{P}[D] \cdot \mathbb{E}[X|D] + \mathbb{P}[D^c] \cdot \mathbb{E}[X|D^c]\} \\ &= \pi\end{aligned}\tag{5.6}$$

Therefore it is clear that the worst case expected termination time is computed as the maximum between (5.3) and (5.6), corresponding to the adversarial choice as to whether the robot lands inside or outside the fence. Since $x \geq 2 \sin(x/2)$ for all x , we see that the dominant expression is (5.6) (for every β) and the theorem follows. $\square$

It is noteworthy that the performance of the algorithm is not dependent on β . Furthermore, the adversary's only concern is to make sure the robot lands within the fence, in which case the actual relative placement of the fence and the treasure become irrelevant.

5.2 Unbounded Random-Bits Algorithms

Our next attempt is toward further improving the worst case expected termination time. Provably, a favourable event/case in the analysis of all our algorithms this far is when the robot lands outside the fence for the first time. Indeed, even for Algorithm 5.1 the adversary chooses to have the robot land inside the fence. Motivated by this, we introduce a new randomized option, Algorithm 5.2, i.e. we choose a random initial landing point on the perimeter of the disk (and then run Algorithm 2.1).

Algorithm 5.2 R-Fence Walk

- 1: Walk to a point of perimeter of the disk, chosen uniformly at random.
 - 2: Perform steps 3-7 of Algorithm 2.1.
-

Against such an algorithm, an adversary can only choose the relative distance

between the treasure and the fence. On the other hand, the required randomness in order to choose a random point on the disk would be enormous. Nevertheless, we are motivated to examine whether such an approach would improve performance.

Theorem 5.2.1. *Let β_0 be the unique solution of equation $\beta - \sin(\beta/2) = \pi$ in $[0, 2\pi]$, i.e. $\beta_0 \approx 4.04$. When the length of the fence is β , then the worst case expected termination time of Algorithm 5.2 is*

$$1 + \pi + \begin{cases} \frac{1}{2\pi} (\beta - \sin(\frac{\beta}{2}) - \pi)^2, & \text{if } \beta \leq \beta_0 \\ 0, & \text{otherwise.} \end{cases}$$

Proof. As before, we omit in the calculations the deployment cost 1, i.e. the time it takes the robot to reach the perimeter. Let X be the random variable that is equal to the worst case (over the adversarial choices) expected (over the algorithmic choices) termination time of Algorithm 5.2. We partition the possible outcomes in three events. In all cases, we assume that the treasure is placed at cw distance d from the fence, where $0 \leq d \leq 2\pi - \beta$.

- Let A_1 be the event that the robot lands inside the fence (see 5.2).

Clearly, we have that $\mathbb{P}[A_1] = \beta/2\pi$. In that case, the cw distance of the robot to the end of the fence is x , where $0 \leq x \leq \beta$, and so

$$\mathbb{E}[X|A_1] = \frac{1}{\beta} \int_0^\beta (\beta - x + d) dx = \frac{\beta}{2} + d. \quad (5.7)$$

- Let A_2 be the event that the robot lands outside the fence and that it encounters the treasure before it encounters the fence (see 5.3).

We have that $\mathbb{P}[A_2] = d/2\pi$. Let also y denote ccw distance between the robot

Figure 5.2: RFW Case 1 (Random basic landing point)

Figure 5.3: RFW Case 2 (Random basic landing point)

and the fence, where $0 \leq y \leq d$. Then

$$\mathbb{E}[X|A_2] = \frac{1}{d} \int_0^d (d - y) dy = \frac{d}{2}. \quad (5.8)$$

- Finally, let A_3 denote the event that the robot lands outside the fence and that it encounters the fence before it encounters the treasure (see 5.4).

We have $\mathbb{P}[A_3] = (2\pi - \beta - d)/2\pi$. Let also z denote ccw distance between the robot and the treasure, where $0 \leq z \leq 2\pi - \beta - d$ (inducing restriction

Figure 5.4: RFW Case 3 (Random basic landing point)

$d \geq 2\pi - \beta$ for the adversary). Then

$$\begin{aligned} \mathbb{E}[X|A_3] &= \frac{1}{2\pi - \beta - d} \int_0^{2\pi - \beta - d} \left(2\pi - z - \beta + 2 \sin\left(\frac{\beta}{2}\right) + d \right) dz \\ &= \frac{3d}{2} + \pi - \frac{\beta}{2} + 2 \sin\left(\frac{\beta}{2}\right). \end{aligned} \quad (5.9)$$

Expressions (5.7), (5.8) and (5.9), along with elementary calculations, allow us to conclude that

$$\begin{aligned} \mathbb{E}[X] &= \mathbb{P}[A_1] \cdot \mathbb{E}[X|A_1] + \mathbb{P}[A_2] \cdot \mathbb{E}[X|A_2] + \mathbb{P}[A_3] \cdot \mathbb{E}[X|A_3] \\ &= \frac{(d - 2\pi + \beta)(-d + \beta - 2 \sin(\beta/2)) + 2\pi^2}{2\pi}. \end{aligned} \quad (5.10)$$

Now, we determine the adversarial choice for d which would induce the worst case expected termination time, i.e. that would maximize (5.10). Observe that (5.10) is a degree 2 polynomial in d with negative leading coefficient. Hence, the expression is maximized when the derivative of the expression is 0. Elementary calculations show that the maximizer is $d_0 = \pi - \sin(\frac{\beta}{2})$, and in particular, when $d < d_0$ expression (5.10) is strictly increasing and when $d > d_0$ expression (5.10) is strictly

decreasing.

Notably, we have explicitly assumed above that $d \leq 2\pi - \beta$, so d_0 is a valid adversarial choice only if $d_0 \leq 2\pi - \beta$. Given the value of d_0 , we conclude that the adversary would choose $d = d_0$ as long as $\beta - \sin(\beta/2) \leq \pi$, i.e. as long as $\beta \leq \beta_0$ (where β_0 is as in the statement of the theorem). For these values of β , the worst case expected performance is obtained by replacing $d = \pi - \sin(\frac{\beta}{2})$ in Equation (5.10). If on the other hand $\beta > \beta_0$, then due to the monotonicity of (5.10), the adversarial choice inducing the worst case expected performance is when $d = 2\pi - \beta$. Again, substituting in (5.10) together with elementary calculations derive the promised formula. $\square$

A comparison between the performances of Algorithm 5.1 and 5.2, is depicted in 5.5. Surprisingly, Algorithm 5.2 is strictly worse than Algorithm 5.1, for all fence

Figure 5.5: Performance of Algorithm 5.2 (yellow line) compared to the performance of Algorithm 5.1 (blue line) as a function of the fence length $\beta \leq \beta_0$.

lengths $\beta < \beta_0 \approx 4.04$. From the analysis of Algorithm 5.2 we learned that although the adversary cannot choose the robot's landing point, the fact that the robot may land inside the fence, and since the robot's movement direction is known, the position

of the treasure can be chosen so that expected termination cost is high. At the same time, we should recall that the power of Algorithm 5.1 lies in that the movement direction of the robot is random, and in particular that the relative positioning of the fence and the treasure are irrelevant. Therefore we are motivated in combining the randomized choices of Algorithms 5.1 and 5.2. This is what we explore with Algorithm 5.3.

Algorithm 5.3 RD-Fence Walk

- 1: Walk to point of perimeter of the disk, chosen uniformly at random.
 - 2: Start walking on the perimeter, either in cw or in ccw direction, each with probability $1/2$.
 - 3: Perform steps 3-7 of Algorithm 2.1.
-

Theorem 5.2.2. *The worst case expected termination time of Algorithm 5.3 is*

$$1 + \frac{\beta}{2} + \left(1 - \frac{\beta}{2\pi}\right) \left(\pi - \frac{\beta}{2} + \sin\left(\frac{\beta}{2}\right)\right).$$

Proof. The analysis is very similar to the one we did for Algorithm 5.1, i.e. the proof of Theorem 5.1.1. We examine two cases as to whether the robot lands inside or outside the fence.

- Robot lands inside the fence (which happens with probability $\beta/2\pi$)
The expected performance of the algorithm is π , independently of where the treasure is placed.
- Robot lands outside the fence (which happens with probability $1 - \beta/2\pi$)
Then the expected performance of the algorithm is $\pi - \beta/2 + \sin(\beta/2)$, again independently of the positioning of the treasure.

Figure 5.6: Algorithm 5.3 performance (yellow line) compared to Algorithm 5.1 performance (blue line) for all fence lengths $0 \leq \beta \leq 2\pi$.

Multiplying the conditional expectations with the probabilities of the corresponding events results in the promised formula. $\square$

As expected, Algorithm 5.3 strictly improves Algorithm 5.1, and this is depicted in 5.6.

5.3 Summary

In this section we introduced a few simple algorithms which incorporated a form of randomness to Algorithm 2.1. First, we defined a one bit random algorithm (R-Robot Direction Algorithm, also denoted as Algorithm 5.1). This particular algorithm allowed for the random choice of the robot's perimeter direction movement (clockwise or counter-clockwise). We calculated the worst case termination time for the robot and, interestingly, we observe that it is π , independent of the fence's length.

Next, we introduced an algorithm that used infinitely many random bits in the form of Algorithm 5.2, R-Fence Walk Algorithm. In this algorithm, a random point

	<i>Random:</i>	<i># bits</i>	<i>Worst-Case Termination Time</i>
R-Robot	Dir	1	π
R-Fence	Point	∞	$\frac{\beta^2 - 2(\beta + d - 2\pi) \sin(\frac{\beta}{2}) - 2\pi\beta - d^2 + 2\pi(d + \pi)}{2\pi}, if \leq 4.04$ $\pi, if \beta > 4.04$
RD-Fence	Dir + point	$\infty + 1$	$\frac{\beta}{2} + (1 - \frac{\beta}{2\pi}) (\pi - \frac{\beta}{2} + \sin(\frac{\beta}{2}))$

Table 5.1: Randomized Algorithm Analysis and Worst-Case Termination Time

is chosen for the robot to land on the perimeter. We observe that this algorithm is dependent on the distance from the endpoint of the fence to the treasure (d) and the length of the fence (β).

Lastly, we made a comparison between the two randomized algorithms, R-Fence Walk and R-Robot Direction. We calculated and visually depicted that even though the R-Robot Direction Algorithm only uses one bit of randomness, it has the most advantageous worst expected case termination time for the robot. Further, we created Algorithm 5.3, RD-Fence Walk Algorithm, which took the R-Fence Walk Algorithm and added the R-Robot Direction Algorithm within it. Therefore, the algorithm was modified from allowing an infinite amount of randomness, to an infinite amount plus one bit. We proved that this new algorithm, which introduced the concept of randomness through the use of a random point and random direction, is even more beneficial for the robot than the 1-bit random R-Robot Algorithm.

We give an analysis of the worst-case termination time between the algorithms in Table 5.1. Most importantly, in this section we observe that using an infinite number of bits for randomness does not always positively influence the robot. However, if we integrate the one bit and infinite bit algorithms, we are able to produce an algorithm whose worst-case termination time will be favourable for the robot.

Chapter 6

Conclusion

In this thesis, we investigated a new search problem for a mobile robot to find a stationary target placed at an unknown location at the initial distance of 1 unit, with the additional presence of a fence placed on the perimeter of a unit disk. First, we determined a naive approach, in which the robot did not possess the capability of fence-jumping. Second, we were able to provide simple divide-by-two algorithms that they did not take into account optimal jumps. We then improved upon these algorithms to get an optimal 1-jump and k-jump analysis of worst case termination time, providing observations and comparisons between these two main algorithms. Next, we proceeded to successfully explore the role of randomization towards improving the performance of our algorithms.

6.1 Future Work

This thesis allows for much opportunity for future research. One of the interesting questions that remains open is: can we devise efficient search algorithms for two (or more) communicating and co-operating robots? What algorithms exist and how does the addition of one or more robots affect the search time? This modification to the

thesis would allow for the inclusion of communication between the robots. We use Section 1.2.3 to define the different types of communication applicable to this new model in order to form a foundation for this future open problem.

Several other possible questions remain open, such as when there are multiple fences on the perimeter of the disc. In this type of problem, factors such as the usage of the same fence-jumping approach to both fences or creating two different fence-jumping techniques which are dependent on the number of fences the robot has jumped on are critical in order to determine an optimal algorithm. Another type of problem which can be built from this thesis is allowing the robot to change speeds when traversing a fence. It would be interesting to see if the speed change would create a more optimal implementation of our algorithms.

Thus, it is quite evident that this thesis allows for many stimulating future problems. While, at first, it may have seemed like this problem was intuitive, it is quite gripping and surprising to find the complexity arising from this robot fence-jumping problem which contains stricter constraints than those mentioned in forthcoming analyses.

Mathematica Code

1 Jump Algorithm (Section 4.1) and Halving Algorithm (Section 4.3)

```
CostCase1[a_, b_] := 2 * Sin[b / 2];
CostCase2[a_, b_] := 2 * Sin[a / 2] - a + b;
CostCase3[a_, b_] := 4 * Sin[a / 2];

OneJumpAlgo[b_] := NMinimize[ {x, x ≥ CostCase1[a, b], x ≥ CostCase2[a, b], x ≥ CostCase3[a, b], a ≥ 0, a ≤ Min[Pi, 2 * Pi - b]}, {x, a}]
```

```
ValueofAOneJumpAlgo[b_] := a /. OneJumpAlgo[b] [[2]]
```

```
plotOneJumpAlgo = Evaluate[Plot[OneJumpAlgo[b] [[1]], {b, 0, 2 * Pi}, PlotStyle → Red]]
plotValueofAOneJumpAlgo = Evaluate[Plot[{ValueofAOneJumpAlgo[b], b / 2, Pi}, {b, 0, 2 * Pi}]]
```

```
test = Evaluate[Plot[2 * Sin[b / 2], {b, 0, 2 * Pi}]];
Show[{plotOneJumpAlgo, test}]
```

```
CCostCase1[a_, b_, c_] := 2 * Sin[b / 2];
CCostCase2a[a_, b_, c_] := b + 2 * Sin[a / 2] - a + 2 * Sin[c / 2] - c;
CCostCase2b[a_, b_, c_] := 2 * Sin[a / 2] + 4 * Sin[c / 2];
CCostCase3[a_, b_, c_] := 4 * Sin[a / 2];

TwoJumpAlgo[b_] := NMinimize[{x, x ≥ CCostCase1[a, b, c], x ≥ CCostCase2a[a, b, c], x ≥ CCostCase2b[a, b, c], x ≥ CCostCase3[a, b, c], a ≥ 0,
  b ≤ Min[Pi, 2 * Pi - b], c ≥ 0, c ≤ Min[Pi, 2 * Pi - b], a + c ≤ b, x ≥ 0, c ≤ b / 2, c ≤ a}, {x, a, c}]

ValueofATwoJumpAlgo[b_] := a /. TwoJumpAlgo[b][[2]]
ValueofCTwoJumpAlgo[b_] := c /. TwoJumpAlgo[b][[2]]
```

```
plotTwoJumpAlgo = Evaluate[Plot[TwoJumpAlgo[b][[1]], {b, 0, 2 * Pi}, PlotStyle → Blue]]
plotValueofACOneJumpAlgo = Evaluate[Plot[{ValueofATwoJumpAlgo[b], ValueofCTwoJumpAlgo[b], b / 2, Pi}, {b, 0, 2 * Pi}]]
```

Show[plotOneJumpAlgo, plotTwoJumpAlgo]

$\frac{b}{2} / \pi \cdot \text{Integrate}[(b - x + d) / b, \{x, 0, b\}] + d / 2 / \pi \cdot \text{Integrate}[y / d, \{y, 0, d\}] + (1 - b / 2 / \pi - d / 2 / \pi) \cdot \text{Integrate}[(2 \cdot \pi - z + 2 \cdot \sin[b / 2] - b) / (2 \cdot \pi - d - b), \{z, 0, 2 \cdot \pi -$

$\text{Simplify}[d^2 / (4 \pi) + (b (b / 2 + d)) / (2 \pi) + 1 / 2 (1 - b / (2 \pi) - d / (2 \pi)) (-b + d + 2 \pi + 4 \sin[b / 2])]$

$$\frac{b^2 + b (d - 2 \pi) + 2 \pi^2 - 2 (b + d - 2 \pi) \sin\left[\frac{b}{2}\right]}{2 \pi}$$

ttest[b_] := NMaximize[{b^2 + b (d - 2 \pi) + 2 \pi^2 - 2 (b + d - 2 \pi) \sin[b / 2], d \ge 0, d \le \pi - b / 2}, d]

Plot[d /. ttest[b][[2]], \pi - b / 2, {b, 0, 2 \cdot \pi}]

$\text{Simplify}[(b^2 + b (-b / 2 - \pi) + 2 \pi^2 - 2 (b / 2 - \pi) \sin[b / 2]) / (2 \pi)]$

$\text{Plot}[(b^2 - 2 b \pi + 4 \pi^2 + (-2 b + 4 \pi) \sin[b / 2]) / (4 \pi), 2 \cdot \pi - b + 2 \cdot \sin[b / 2], \{b, 0, 2 \cdot \pi\}]$

$\text{Plot}[(b^2 - 2 b \pi + 4 \pi^2 + (-2 b + 4 \pi) \sin[b / 2]) / (4 \pi + 2 \cdot \pi), \{b, 0, 2 \cdot \pi\}]$

Randomized Algorithms (Section 6)

```
Simplify[ (b^2 + b (-b/2 - π) + 2 π^2 - 2 (b/2 - π) Sin[b/2]) / (2 π)]
Plot[ {(b^2 - 2 b π + 4 π^2 + (-2 b + 4 π) Sin[b/2]) / (4 π), 2 * Pi - b + 2 * Sin[b/2]}, {b, 0, 2 * Pi}]
```

```
Plot[ {(b^2 - 2 b π + 4 π^2 + (-2 b + 4 π) Sin[b/2]) / (4 * Pi + 2 * Pi)}, {b, 0, 2 * Pi}]
```

```
a[x_, y_] := Piecewise[
{
  {b/2, x == 1 && y == 1}, {2 * Pi - b, x == 1 && y ≥ 2},
  {Simplify[ (b - Sum[a[t, y], {t, 1, x - 1}]) / 2],
    x ≥ 2 && x ≥ y}
}, 2 * Pi - b]
```

```
CriticalValue[t_] := b /. Solve[ a[t, t] == 2 * Pi - b, b] [[1]]
```

```
CriticalValue[10]
```

$$\frac{11\pi}{6}$$

```
Plot[ CriticalValue[t], {t, 1, 500}]
```

```
Plot[ {b - Sin[b/2], Pi}, {b, 4.04195, 4.04197}]
```

```
bi[b] := (b^2 - 2 b Pi - (Pi - Sin[b/2])^2 + 2 Pi (2 Pi - Sin[b/2]) - 2 (b - Pi - Sin[b/2]) Sin[b/2]) / (2 Pi)
```

```
Plot[{Pi, bi[b]}, {b, 0, 4.3}, PlotRange -> {2, 2 + Pi}]
```

```
n1[b_] := Pi - b/2 + Sin[b/2]
```

```
Plot[{Pi, n1[b]}, {b, 0, 6.28}, PlotRange -> {0, 2 + Pi}]
```

List of References

- [1] R. A. Baeza-Yates, J. C. Culberson, and G. J. Rawlins, “Searching in the plane,” *Information and computation*, vol. 106, no. 2, pp. 234–252, 1993.
- [2] M.-Y. Kao, J. H. Reif, and S. R. Tate, “Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem,” *Information and Computation*, vol. 131, no. 1, pp. 63–79, 1996.
- [3] S. K. Ghosh and S. Saluja, “Optimal on-line algorithms for walking with minimum number of turns in unknown streets,” *Computational Geometry*, vol. 8, no. 5, pp. 241–266, 1997.
- [4] M.-Y. Kao and M. L. Littman, “Algorithms for informed cows,” in *Proceedings of the AAAI 1997 Workshop on Online Search*, 1997.
- [5] I. E. Sutherland, “A method for solving arbitrary-wall mazes by computer,” *IEEE Transactions on Computers*, vol. 100, no. 12, pp. 1092–1097, 1969.
- [6] O. Ore and Y. Ore, *Theory of graphs*, vol. 38. American Mathematical Society Providence, 1962.
- [7] E. D. Demaine, S. P. Fekete, and S. Gal, “Online searching with turn cost,” *Theoretical Computer Science*, vol. 361, no. 2, pp. 342–355, 2006.
- [8] T. Ishida and R. E. Korf, “Moving target search.,” in *IJCAI*, vol. 91, pp. 204–210, 1991.
- [9] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, “Robocup: A challenge problem for ai,” *AI magazine*, vol. 18, no. 1, p. 73, 1997.
- [10] G. Berzsenyi, “Lost in a forest,” *Quantum (November/December, 1995)*, vol. 41, 1995.

- [11] S. R. Finch and J. E. Wetzel, “Lost in a forest,” *The American Math. Monthly*, vol. 111, no. 8, pp. 645–654, 2004.
- [12] J. Isbell, “An optimal search pattern,” *Naval Research Logistics Quarterly*, vol. 4, no. 4, pp. 357–359, 1957.
- [13] A. Beck, “On the linear search problem,” *Israel Journal of Mathematics*, vol. 2, no. 4, pp. 221–228, 1964.
- [14] R. Baeza-Yates and R. Schott, “Parallel searching in the plane,” *Computational Geometry*, vol. 5, no. 3, pp. 143–154, 1995.
- [15] J. Czyzowicz, L. Gasieniec, T. Gorrry, E. Kranakis, R. Martin, and D. Pajak, “Evacuating robots via unknown exit in a disk,” in *DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pp. 122–136, 2014.
- [16] J. Czyzowicz, K. Georgiou, E. Kranakis, L. Narayanan, J. Opatrny, and B. Vogtenhuber, “Evacuating robots from a disk using face-to-face communication (extended abstract),” in *CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, pp. 140–152, 2015.
- [17] S. Badal, S. Ravela, B. Draper, and A. Hanson, “A practical obstacle detection and avoidance system,” in *Appl. of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pp. 97–104, IEEE, 1994.
- [18] A. Blum, P. Raghavan, and B. Schieber, “Navigating in unfamiliar geometric terrain,” in *STOC 1991*, pp. 494–504, ACM, 1991.
- [19] C. Pozna, F. Troester, R.-E. Precup, J. K. Tar, and S. Preitl, “On the design of an obstacle avoiding trajectory: Method and simulation,” *Mathematics and Computers in Simulation*, vol. 79, no. 7, pp. 2211–2226, 2009.
- [20] C. Fulgenzi, A. Spalanzani, and C. Laugier, “Probabilistic motion planning among moving obstacles following typical motion patterns,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4027–4033, IEEE, 2009.
- [21] A. Stentz, “The focussed D* algorithm for real-time replanning,” in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (C. S. Mellish, ed.), (San Mateo), pp. 1652–1659, Morgan Kaufmann, Aug. 20–25 1995.

- [22] K. Georgiou, E. Kranakis, and A. Steau, “Searching with advice: Robot fence-jumping,” *arXiv preprint arXiv:1606.08023*, 2016.
- [23] K. Georgiou, E. Kranakis, and A. Steau, “Evacuating robots from a disk using face-to-face communication (extended abstract),” in *CCCG 2016, Vancouver, Canada, Aug 3-5, 2016. Proceedings*, pp. 21–28, 2016.