Diploma Thesis

**SEARCH-AND-FETCH AND GATHERING WITH THREE ROBOTS**

**Frederikos Leandrou**

**UNIVERSITY OF CYPRUS**



**DEPARTMENT OF COMPUTER SCIENCE**

**May 2018**

**UNIVERSITY OF CYPRUS**

**DEPARΤMENT OF COMPUTER SCIENCE**

**Search-and-Fetch and Gathering with Three Robots**

**Frederikos Leandrou**

Supervisors

Anna Philippou

Chryssis Georgiou

Marios Mavronicolas

This Diploma Thesis was submitted as part of the requirements needed for obtaining a degree in Computer Science from the Department of Computer Science of the University of Cyprus

May 2018

**Acknowledgements**

I would like to express my deep gratitude to Professor Marios Mavronicolas for giving me the opportunity to work with him as well as his patient guidance and the useful advice he has given me during the preparation of this thesis.

I am particularly grateful for the assistance given by Associate Professor Anna Philippou and Associate Professor Chryssis Georgiou. Their continuous support and invaluable help, in the absence of Dr. Mavronicolas, were vital to completing this thesis.

I would like to extend my thanks to the rest of the teaching staff that have taught me throughout my studies as well as my fellow students for any help they have given me.

Lastly, special thanks to my family and friends for the inexpressible support they gave me all this time.

**ABSTRACT**

Autonomous mobile robots have numerous applications in various field. The most important application can be said to be replacing humans working in dangerous environments thus reducing the risk of injury or loss of life. This atomization is not only about replacing humans. Autonomous robots can provide additional “manpower” by working alongside humans in situations that call for it, for example in rescue operations where people can use all the help they can get. As such the study of autonomous mobile robots is very important.

Our study concerns two of the basic problems in the study of autonomous robots, the *search-and-fetch problem* and the *gathering problem*. The *search-and-fetch problem* concerns autonomous robots searching for a treasure then moving it to an exit. The variation we study is about the *search-and-fetch problem* on a disk unit with three robots that can communicate with each other wirelessly. Our aim is to design an algorithm that minimizes the worst-case treasure evacuation time and to see if it is smaller than a given limit. The *gathering problem* concerns the gathering of robots in finite time while then robots operate asynchronously. In order for gathering to be possible, we assume that the robots are equipped with coloured lights and are able to measure distances between one another. Aside from its light, a robot has no memory of its past actions and its protocol is deterministic. Our aim is to design an algorithm that solves the *gathering problem* for three robots with the lowest amount of colours, i.e. two.

**Table of Contents**

[Chapter 1 Introduction 1](#_Toc515554429)

[1.1 Motivation 1](#_Toc515554430)

[1.2 Objective 2](#_Toc515554431)

[1.3 Methodology and Contributions 2](#_Toc515554432)

[1.4 Document Structure 3](#_Toc515554433)

[Chapter 2 Background and Related Work 4](#_Toc515554434)

[2.1 Search-and-Fetch with Two Robots on a Disk 4](#_Toc515554435)

[2.1.1 Background 4](#_Toc515554436)

[2.1.2 Related Work 6](#_Toc515554437)

[2.2 Optimally Gathering Two Robots 8](#_Toc515554438)

[2.2.1 Background 8](#_Toc515554439)

[2.2.2 Related Work 9](#_Toc515554440)

[Chapter 3 Search-and-Fetch with Three Robots on a Disk 12](#_Toc515554441)

[3.1 Objective 12](#_Toc515554442)

[3.2 Suggested solution by [1] for 13](#_Toc515554443)

[3.3 Proposal 1 15](#_Toc515554444)

[3.4 Improvement 20](#_Toc515554445)

[3.4.1 Improving the value of by allowing robots to find interesting points 20](#_Toc515554446)

[3.4.2 Case 3: Value of when is outside but is inside the arc 21](#_Toc515554447)

[3.4.3 How Case 2 is affected 22](#_Toc515554448)

[3.5 Combing both algorithms to improve time 23](#_Toc515554449)

[3.5.1 Time of combined algorithm 24](#_Toc515554450)

[3.5.1.1 Helper robot partnered with 24](#_Toc515554451)

[3.5.1.2 Helper robot partnered with 26](#_Toc515554452)

[3.5.2 Total time of combined algorithm 27](#_Toc515554453)

[Chapter 4 Optimally Gathering Three Robots 30](#_Toc515554454)

[4.1 Objective 30](#_Toc515554455)

[4.2 Our algorithm 30](#_Toc515554456)

[4.3 Analysis 33](#_Toc515554457)

[4.3.1 Robot , case (a) 34](#_Toc515554458)

[4.3.1.1 Robot activated before completes its COMPUTE phase 35](#_Toc515554459)

[4.3.1.2 Robot not activated before completes its COMPUTE phase 36](#_Toc515554460)

[4.3.2 Robot , case (b) 36](#_Toc515554461)

[4.3.2.1 A robot is activated 37](#_Toc515554462)

[4.3.3 Robot , case (c) 37](#_Toc515554463)

[4.3.3.1 A robot is activated 37](#_Toc515554464)

[4.3.3.2 A robot is not activated 37](#_Toc515554465)

[4.3.4 Irregular case 38](#_Toc515554466)

[Chapter 5 Conclusion and Future Work 41](#_Toc515554467)

# Introduction

1.1 Motivation 1

1.2 Objective 2

1.3 Methodology and Contributions 2

1.4 Document Structure 3

## Motivation

Robotics can be said to be the crowning point of our current technological development and as such the study of it is a very important subject. Robots can theoretically be used in any situation and for any purpose but one of and maybe the most important use for them is in environments that are dangerous or impossible for humans to survive in. Using robots to replace humans operating in these dangerous environments reduces the risk of injury or loss of human life. While these dangerous environments can range from the mundane, such as operating dangerous machinery during manufacturing or construction work, to the extreme, such as outer space, we do not have to go to outer space to find extremely dangerous environments. Places with high temperature (like an active volcano), freezing temperatures (the artic regions), high pressure levels (oceanic trenches) or places which present chemical, biological, radiological or nuclear risk, are all places where humans would preferably not be in. As technology advances, robots may completely replace humans in operating in these environment although the human factor may still be present depending on whether the robots are completely autonomous or remotely controlled by a person.

Autonomous robots can, not only replace humans in these dangerous activities, but also provide an extra pair of hands in situations when not enough manpower is available. However, before any complex tasks can be automated, there are a number of basic problems to solve. One of them is the field of navigation and exploration. This field is about how the robot can navigate its environment, avoid collisions and unsafe situations, and find specific places if needed. Problems in this field are the ones that incorporate the elements of robot navigation, self-localisation, the ability to determine its position in its frame of reference, path planning, the ability to plan a path towards a location it needs to go to, and map building and interpretation, the ability to create a representation of its surroundings and interpreting that representation.

## Objective

Our study concerns two basic problems in the field of navigation. A variation of the search-and-fetch problem and the gathering problem.

First we have a variation of the *search-and-fetch problem*, a problem which concerns robots trying to find a “treasure” and then moving it to a different location. This part of our study is about the search-and-fetch problem where the search space is a disk and the number of robots involved is three. The problem and model used in this part are based upon the work in [1] where they consider two robots. Our objective is to solve the Search-and-Fetch on a Disk using *three robots* and see if that can be achieved in a better time than when using the solution that the work in [1] suggests.

The second part of our study has to do with the *gathering* problem. A benchmarking problem, gathering has received a lot of attention since its introduction [2, 3] and involves a number of autonomous robots, represented as points on a 2-dimensional Euclidean plane, trying to reach a, previously unknown, single point in finite time and, in other words, gather. The difficulty of this problem stems from the fact that the robots are anonymous (i.e. indistinguishable from each other), oblivious (i.e. no persistent memory of the past is available) and disoriented (i.e. they do not agree on a common coordinate system). Our objective is to solve the gathering problem for 3 robots in the non-rigid asynchronous model.

## Methodology and Contributions

Upon being introduced to both of these problems, firstly to the search-and-fetch problem and then to the gathering problem, we began to study them in depth in order to understand the models that the problems are based upon.

At first, we sought to solve the search-and-fetch problem without basing our solution on the solution suggested by [1] and instead sought to use the unique morphology we can achieve for the problem when using three robots. After that we incorporated the suggested solution in our solution with successful results. While not disproving the suggested solution and achieving the same upper bound in the time needed to solve the problem, our solution only achieves that upper bound around 50% of the time while reducing the time needed the rest of the time.

For the gathering problem, we did the opposite: first we based our solution on the logic used by [4] and tried to gather the robots in one of the other robots’ location. That failed to bring the desired results and thus, once again, we took advantage of the morphology that arises from having three robots and used it to gather the robots to a common point successfully. As such, we present a solution that solves the gathering problem for three robots in the non-rigid asynchronous model.

## Document Structure

The remainder of the paper is organized as follows. In Chapter 2 we present the background of the subject of our study. It consists of the history of the problems that are our study topics. Previous research regarding our topics and application examples of the results are also presented. In Chapters 3 and 4 we present our work on the search-and-fetch on a disk problem and the gathering problem, respectively. In each of these chapters we firstly introduce the problem, our objective and the model used in more depth before presenting our solution and the rest of our work. Chapter 5 includes the final conclusions where we also present future work that can be done on the subjects.

# Background and Related Work

2.1 Search-and-Fetch with Two Robots on a Disk 4

2.1.1 Background 4

2.1.2 Related Work 6

2.2 Optimally Gathering Two Robots 8

2.2.1 Background 8

2.2.2 Related Work 9

## Search-and-Fetch with Two Robots on a Disk

### Background

This part of our study is based on the paper “Search-and-Fetch with 2 Robots on a Disk: Wireless and Face-to-Face Communication Models” by Konstantinos Georgiou, George Karakostas and Evangelos Kranaki [1].

The paper introduces a “search and fetch” problem variation called “treasure evacuation” as follows:

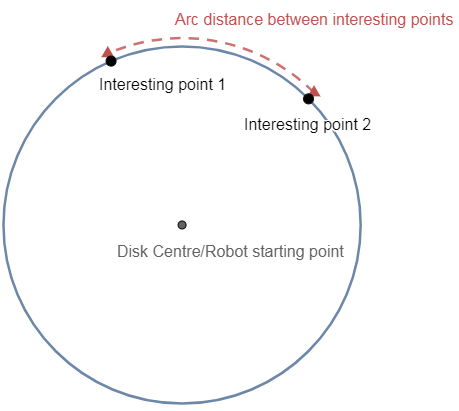


Figure 2.1 Treasure evacuation on a disk

Two robots are placed at the centre of a unit disk, while an exit and a treasure lie at unknown positions on the perimeter of the disk (see Figure 2.1). Robots search with constant (and maximum) speed of 1, and they detect an interesting point (either the treasure or the exit) only if they pass over it. The exit is immobile, while the treasure can be carried by any of the robots. The goal of the search is for at least one of the robots to bring (fetch) the treasure to the exit, i.e. evacuate the treasure, in the minimum possible completion time. The robots do not have to evacuate, rather they only need to co-operate, possibly by sharing information, so as to learn the locations of the interesting points and bring the treasure to the exit.

Finally, the two robots have some knowledge, the arc-distance between the exit and the treasure.

The goal of the paper is to design distributed algorithms that minimize the worst-case treasure-evacuation time, i.e. the time it takes for the treasure to be discovered and brought (fetched) to the exit by any of the robots. The time is measured in time units, where in one time unit the robots travel one unit of distance. The communication protocol between the robots is either wireless, where information is shared at any time or face-to-face (i.e. non-wireless), where information can be shared only if the robots meet. Communication, whatever the protocol, is instantaneous and does not take up time.

The proposed protocols induce a worst case evacuation time for the wireless model and for the face-to-face model.

Our study concerns the wireless mode of operation where [1] proposes the following algorithm:

|  |
| --- |
| **Algorithm 1** Wireless Algorithm |
| 1. If , then the two robots move together to an arbitrary point on the ring and start moving in opposite directions, else they move to arbitrary antipodal points , on the circle and start moving in the same direction. 2. Let be the first interesting point discovered by , at time . Let , be the points that are at clockwise and counter-clockwise arc-distance from respectively. 3. If then the robots learn that the other interesting point is in , else moves to , moves to . 4. Evacuate. |

### Related Work

Conventional search algorithms concern solving search problems, finding an object of certain properties, within a search space. Without dismissing the difficulty of normal search algorithms, searching is usually more demanding when done as part of a computational problem [5, 6, 7]. Even more so when the searcher has no knowledge about the environment. This rise in difficulty become more clear when dealing with problems concerning the exploration of given geometric domains by groups of independent yet communicating robots. When a certain geometry comes in play, then the goal changes from not only designing an algorithm that fulfils the requirements needed to solve the search problem to designing an algorithms that does that while adhering to the computational and geometric constraints. As always the searcher must complete its task in the minimum time possible, something that when dealing with search problems in the context of geography the initial configuration of the robot also plays an important role [8].

In the long and extensive history of searching, several models have been suggested and explored that have found use in various areas such as searching on a graph [9], in game theory applications [6], in pathfinding problems [10, 11], in pursuit-and-evasion theory [12] and many others. As for search problems similar to our own, these include evacuation problems where the search domain is a unit circle [13, 14, 15] (or another simple geometric shape such as a square or triangle [16]), problems about searching for a stationary point such as the lost at sea problem [17] as well as search-and-fetch problems of the robotics community [18]. One source of inspiration for our problem comes from real-life search-and-rescue operation involving unmanned vehicles with these operations mostly consisting of drones searching an area for victims following a disaster. Similarities can be drawn between our problem and search-and-rescue operations such as a group of rescue agents (robots), with an initial position in a strategically central area, that when activated by a signal, e.g. an alarm, will seek out victims (the treasure) and bring it to safety (the exit). Similarities can also be drawn to pursuit-and-evasion problems such as the cops-and-robbers game. The robots rest at a central location of the domain (in our case, the centre of the disk) until an alarm is triggered by the robbers (in our case, the treasure) upon which the robots must seek out the robber and bring him to jail (in our case, the exit). Other search-and-evacuate problems include bomb-disposal in which a robot must locate the bomb and remove it from the premises as fast as possible before it attempts to defuse it or the bomb detonates.

The three discerning properties of our model are the number of robots, the domain (a disk) and the fact that the robots have some knowledge concerning the interesting points (the distance between them). As in [1], these three properties directly influence our results and if we attempt to generalize them in order to apply our result to more generic problems then some modifications to our algorithms will be needed. Our concern though is about optimizing the time needed for three robots to solve this problem.

The domain of our problem is not unnatural as, in a basic search-and-rescue operation, the rescue robots are gathered in their base at a central location until a signal activates them. This signal can provide information about the location of the victim needing rescue such as its distance from the robots’ base and even the distance between it and another victim. The distance between the robots’ base and the victim can be considered to be the radius of a circle and as such the locations of the victims can be considered as a point on various concentric circles around the robots’ base and when the distances are equal then they are all part of one circle just like in our problem.

One of the most important properties of the model is the fact that the robots know the distance between the two points. This is extremely important as it allows the robots to make deductions about the location of points and detect them without actually passing over them. Our strategy revolves around using this knowledge. Without it we would not be able to deduce important information about the topology and we would need a different algorithmic approach in order to solve the problem.

## Optimally Gathering Two Robots

### Background

The second part of our study concerns the gathering problem in the non-rigid asynchronous model. In our model the robots are anonymous (i.e. indistinguishable from each other), oblivious (i.e. no persistent memory of the past is available), and disoriented (i.e. they do not agree on a common coordinate system). The robots operate in Look-Compute-Move cycles. In each cycle a robot “Looks" at its surroundings and obtains (in its own coordinate system) a snapshot containing the locations of all robots. Based on this visual information, the robot “Computes" a destination location (still in its own coordinate system) and then “Moves" towards the computed location. Since the robots are identical, they all follow the same deterministic algorithm. The algorithm is oblivious if the computed destination in each cycle depends only on the snapshot obtained in the current cycle (and not on the past history of execution). The snapshots obtained by the robots are not consistently oriented in any manner (i.e. the robots' local coordinate systems do not share a common direction nor a common chirality).

The execution model significantly impacts the ability to solve collaborative tasks. Three different levels of synchronization have been considered. 1) The strongest model, the fully synchronized (FSYNC) model where each phase of each cycle is performed simultaneously by all robots. 2) The semi-synchronous (SSYNC) model considers that time is discretized into rounds, and that in each round an arbitrary yet non-empty subset of the robots are active. The robots that are active in a particular round perform exactly one atomic Look-Compute-Move cycle in that round. It is assumed that the scheduler (seen as an adversary) is fair in the sense that in each execution, every robot is activated infinitely often. 3) The weakest model is the asynchronous model (ASYNC), which allows arbitrary delays between the Look, Compute and Move phases, and the movement itself may take an arbitrary amount of time. In this paper, we consider the most general ASYNC model.

We assume a system of two robots with a light (There are two possible colours, Black, lights off, and White, lights on) that are modelled as mobile coloured points in a Euclidean plane . All robots execute cycles that consist of three phases: LOOK, COMPUTE and MOVE. When a robot is not executing a LOOK-COMPUTE-MOVE cycle, it is considered to be in a WAIT phase. Each phase is described as follows:

* WAIT: The robot is idle and awaits activation.
* LOOK: The robot takes a snapshot of its surroundings, the position of the other robots as well as the colours of the other robots and its own. This phase is assumed to be instantaneous.
* COMPUTE: The robot computes its next destination using the snapshot taken. A robot is able to change the colour of its own light at the end of its COMPUTE phase.
* MOVE: The robot moves towards its destination.

The duration of the COMPUTE and MOVE phases, and the delay between two phases, are chosen by an adversary and can be arbitrary long, but are finite. The adversary decides when robots are activated assuming a fair scheduling i.e., in any configuration, all robots are activated within finite time. The adversary also controls the robots movement along their target path and can stop a robot before reaching its destination, but not before traveling at least a distance ( being unknown to the robots). The existence of this is one of the liveness properties of the system (the other liveness property is about the fair scheduling mentioned above and explained in Section 4.2) that ensures that progress is being made i.e. the robots move. In other words, if a robot has a target at a distance *x*, we assume that, at the end of its MOVE phase, the robot has moved a distance in the interval The exact position reached is determined by the adversary scheduler. Robots are anonymous, meaning they are indistinguishable, and they execute the exact same algorithm. Each robot has a local coordinate system about which we make no assumptions, in particular, they may have distinct North, chirality, and unit distance. We also assume that, except from their light colour, robots have no mean of explicitly communicating with each other. Finally the robots are oblivious, meaning that they do not remember their past actions. This implies that the COMPUTE phase can have no other input than the snapshot the robot took from the last LOOK phase.

### Related Work

Due to their varied applications in dangerous environments, such as rescue and exploration, networks of robots that evolve while operating in a 2-dimensional Euclidian space have been the subject of many studies ever since the model’s initial presentation. [2]

The gathering problem is a benchmarking problem and because of that it has received a lot of attention over the years. ( [3] and therein references). Gathering task involve robots, represented as dimensionless points on a 2-dimensional Euclidean plane, that must reach a previously unknown single point in finite time. Studies [2, 19] have shown that there is an underlying impossibility, in the SSYNC and by extension the ASYNC model there is no deterministic algorithm that can solve the gathering problem without additional assumptions. In addition to that, robots are likely to fail due to the dangers the hostile environments in the applications we imagine provide.

Thus far there are three kinds of failures that have been considered for the problem of deterministic gathering:

1. **Transient faults:** In which transient faults corrupt the robots’ state that lead to an initial configuration from which they many not recover. The only self-stabilizing solution considers only odd sets of robots [20] where other self-stabilizing solutions have stricter conditions on the scheduler [21, 22].
2. **Crash faults:** In which robots may crash and stop executing the algorithm unexpectedly. Crashed robots are initially indistinguishable from the others and guaranteeing that the remaining correctly operating robots still gather is a challenge. All proposed solutions up to this point operate in the SSYNC model [21, 22, 23, 24].
3. **Byzantine faults:** In which robots exhibit random and possibly malicious behavior. There is no deterministic solution to the gathering problem in the SSYNC and by extension to the ASYNC model when even one robot is Byzantine.

To bypass these impossibility results, it has been suggested to give each robot a light [25]. This light is able to emit a fixed amount of colours and is visible to all other robots. With this additional assumption, the problem of gathering two robots has been solved in the most general ASYNC model, given though that the robots are able to emit at least two colours and without assuming rigid movement or knowing the initial configuration of the colors or knowing the distance. ( [4] and therein references)

The algorithm in [4] that solves the gathering problem for two robots in the most general ASYNC model is presented below:

|  |
| --- |
| **ASYNC 2 robot gathering with two colours** |
| **if** me.colour = White **then**  **if** (me.position = other.position) **then**  do nothing  **else if** other.colour = White **then**  me.destination = other.position/2  me.colour = Black  **else if** other.colour = Black **then**  me.destination = other.position  **end if**  **else if** me.colour = Black **and** other.colour = Black **then**  me.colour = White  **end if** |

# Search-and-Fetch with Three Robots on a Disk

3.1 Objective 12

3.2 Suggested solution by [1] for 13

3.3 Proposal 1 15

3.4 Improvement 20

3.4.1 Improving the value of by allowing robots to find interesting points 20

3.4.2 Case 3: Value of when is outside but is inside the arc 21

3.4.3 How Case 2 is affected 22

3.5 Combing both algorithms to improve time 23

3.5.1 Time of combined algorithm 24

3.5.1.1 Helper robot partnered with 24

3.5.1.2 Helper robot partnered with 26

3.5.2 Total time of combined algorithm 27

## Objective

As mentioned in Section 2.1.1, this part of our study is based upon the work in [1]. One of the open problems of the paper is how the problem changes when there are more than two robots. When the number of robots, , is greater than 2, the paper suggests extending the 2-robot algorithms to the -robot case by splitting the robots into pairs, deﬁning points in intervals of length on the cycle, assigning each pair of robots to each such point, and letting them run the corresponding 2-robot algorithm. (When is even, otherwise ignore one robot).

The objective of our study is to tackle the problem when the number of robots is three, , in the wireless model and see if we can put the third robot to good use and provide a better solution than the 2-robot algorithm that has been extended and ignores one robot.

## Suggested solution by [1] for

As mentioned above, the suggested solution from [1] for when is to ignore one of the robots, making and essentially making the three robot case identical to the original problem. As such it will run the suggested 2 robot algorithm for the wireless model as follows.

In what follows point will always be the starting point of the second robot,, and denotes its antipodal point. For the sake of the analysis and w.l.o.g. we will assume that is the one that ﬁrst ﬁnds an interesting point , where is the exit and is the treasure, say at time := . We call , the points that are at clockwise and counter-clockwise arc-distance from respectively. At this point, it is unknown whether is the exit or the treasure (see Figure 3.1).

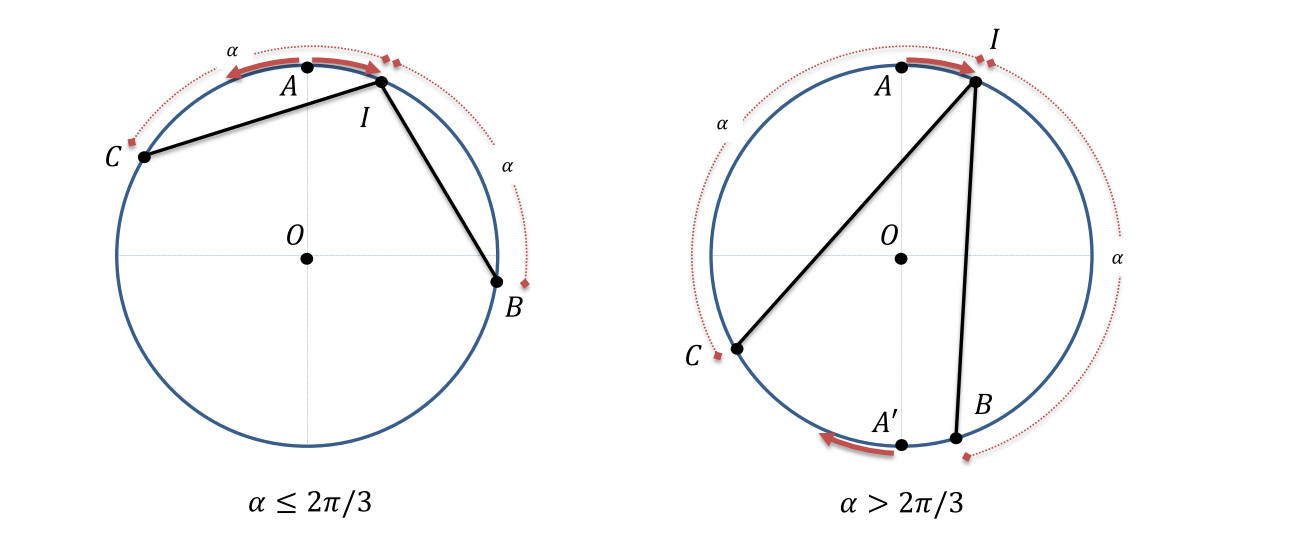


Figure 3.1: Interesting points for the suggested algorithm

**Wireless Algorithm**

1. If , then the two robots move together to an arbitrary point on the ring and start moving in opposing directions, else they move to arbitrary antipodal points on the cycle and start moving in the same direction.
2. Let be the ﬁrst interesting point discovered by R2, at time . Let be the points that are at clockwise and counter-clockwise arc-distance from respectively.
3. If then robots learn that the other interesting point is in , else moves to , moves to .
4. Evacuate.

The time needed for evacuation depends on which interesting point is at point and whether the other interesting point is at either point or point. Presented below are the times required in each case. The continuous arrows are the path of and the dotted arrows are the path of .

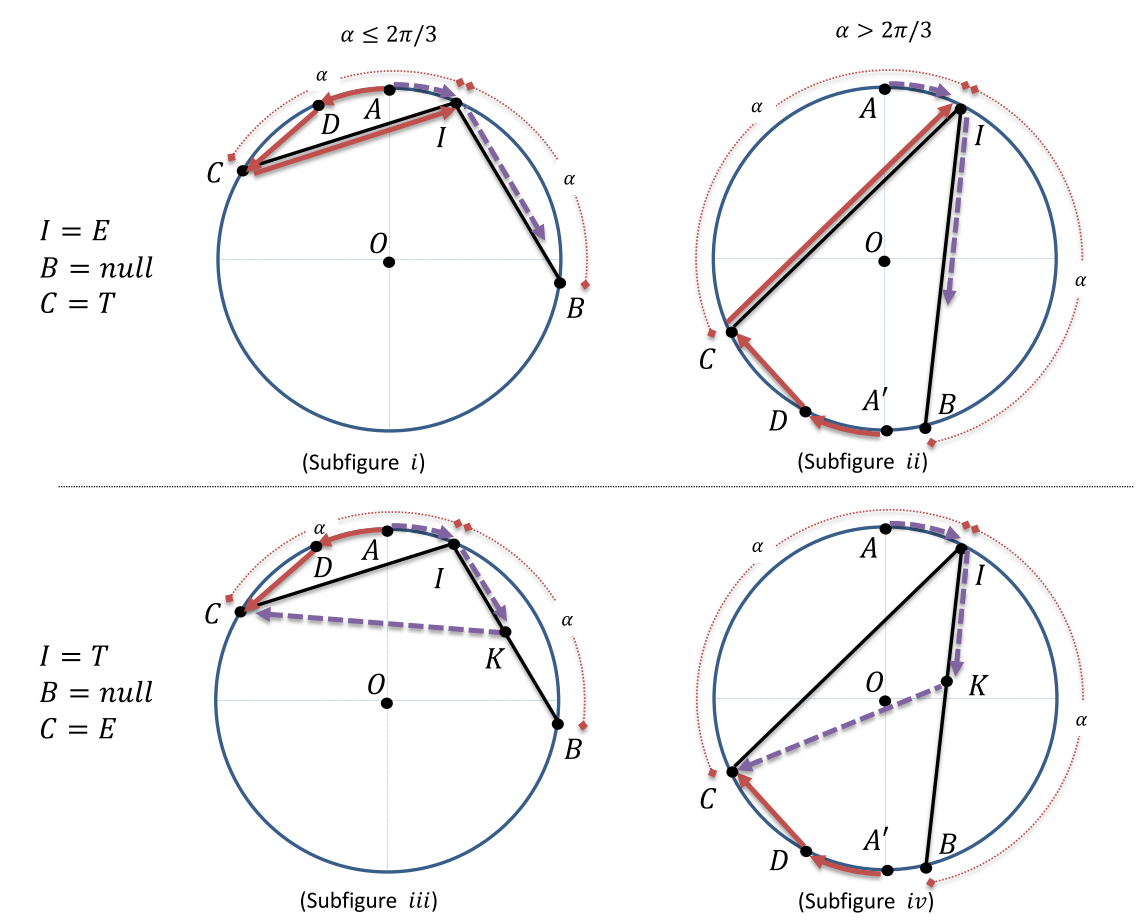


Figure 3.2: Case analysis of suggested algorithm when

In Figure 3.2.i, the time required is:

In Figure 3.2.ii, the time required is:

In Figure 3.2.iii, the time required is:

In Figure 3.2.iv, the time required is:

In the case where , the time required is either or , depending on whether orrespectively.

Each of the times presented above is less or equal to . Combined with the fact that the robots take time 1 to reach the perimeter of the disk, we have an upper bound of .

## Proposal 1

Our proposed solution is a slightly different algorithm that takes advantage of a formation the robots can take when there are three of them.

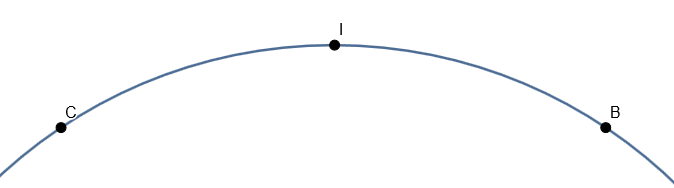


Figure 3.3: Possible interesting points created

The basic idea depends on the fact that when a robot discovers an interesting point two potential interesting points are created clockwise and counter-clockwise from it. (see Figure 3.3)

In the above example, a robot finds an interesting point (e.g. the treasure) at point . Given that we know the arc-distance between the exit and the treasure we can deduce that the remaining interesting point (e.g. the exit) will be to a point that is arc-distance away from point , either at or . Since we have three points, , and , the idea is for each robot to head to a different point and check what that point is. To achieve that we first, place one robot on a random point on the perimeter of the disk. After that we place one robot at clockwise arc-distance from the initial robot, and one robot at counter-clockwise arc-distance from the initial robot. These three robots will then start moving in the same direction in unison. Once the robot in the middle finds point , the other robots will be at points and as shown in Figure 3.3 and thus finding the remaining interesting point immediately.

As such our algorithm makes the three robots take the above formation:

**Algorithm 1**

1. Before doing anything the robots arbitrarily choose between them one of three roles, a Master and two Helpers, and . The roles differ slightly in what they do but which robot assumes which role does not matter. They can decide the roles by any random mean, for example, by each generating a random number, comparing them and with the higher number being the Master, the second higher being the Helper and the lowest number being Helper .
2. The Master robot () chooses an arbitrary point on the disk to move to and conveys that information to the Helper robots.

Helper robot left () will move to the point on the disk that is arc-distance counter-clockwise of the point that chose.

Helper robot right () will move to the point on the disk that is arc-distance clockwise of the point that chose.

The robots assume their positions.

1. The chooses a direction to move to (clockwise or counter-clockwise), conveys that information to the Helper robots and they all start moving in the same direction.
2. and ignore any interesting points they come across until finds one (allowing them to find themselves in a situation as in Figure 3.3). When finds an interesting point he stops and orders the Helper robots to stop as well.
3. , and communicate and learn what interesting point is at their respective locations.
4. The robot that found the treasure picks it up and heads to the exit to evacuate.

The time required for the above algorithm is roughly broken down below (Time having the same meaning as the time defined in Section 2.1.1):

* Time to move from the centre of the disk to their starting position: 1
* Time for to find an interesting point: (= starting position of , =position of interesting point discovered by )
* Time required for evacuation: (= )

Total time = which is less than the time suggested by the paper, which is , where .

However there is a problem. The value of , the time needed for a robot to find , differs between our algorithm and the paper’s.

In the paper’s algorithm the 2 robots start from the same starting point and move in opposite directions or start in antipodal starting points and move in the same direction, essentially, in the worst case, scanning half a disk each.

At one point either one will find an interesting point and the algorithm will continue to its next stage. In our algorithm the robots move in the same direction and will, at worst, scan the whole disk granting a value between .

The direction chosen determines the value of for our algorithm. Let us run through both algorithms together step by step.

* Our robot and the paper’s robots and take the same starting position on arbitrary point on the disk.
* will move counter-clockwise and will move clockwise and will discover an interesting point first. (see Figure 3.4)

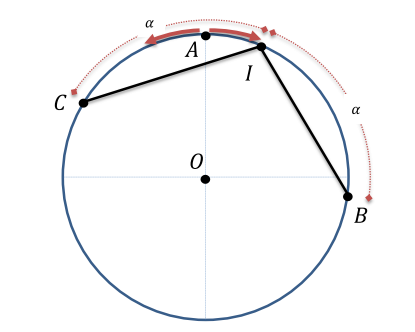


Figure 3.4: Execution example of suggested algorithm for

Now the matter of “right” direction of our algorithm comes into play. Should decide to move clockwise, it will essentially move the same as and in this case the of the paper’s algorithm and the of our algorithm are the same. However, should it move counter-clockwise, will discover an interesting point at either or , arriving to it in time which is intuitively greater than . So our is either the same as the paper’s, the optimal , or it is equal to a whose value can vary. The value of will vary depending on where the starting point is, inside or outside the arc with distance created by the interesting points.

**Theorem 3.1.** *The expected time complexity of Algorithm 1 is*

**Proof:**

The proof considers these two cases:

**Case 1: Value of when is inside the arc**

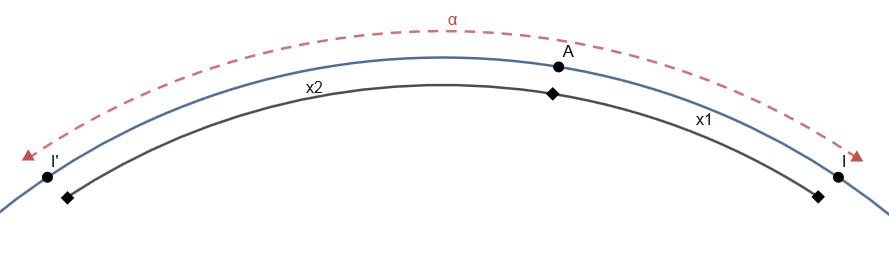


Figure 3.5: Starting point inside arc of interesting points

In this case, the starting point of robot , point , is inside the arc with arc-distance created by interesting points and . Point splits this arc into two parts, and . Intuitively is smaller than making the optimal and the in the paper’s algorithm. then becomes out , which in this case is equal to . In this case the value of has an upper bound of and has an upper bound of .

**Case 2: Value of when is outside the arc**

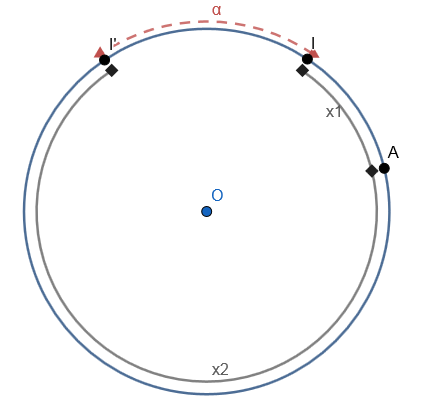
****

Figure 3.6: Starting point outside of arc of interesting points

In this case, the starting point of robot , point , is outside the arc with arc-distance created by interesting points and . also splits this arc into two parts, and , with being smaller than and being the optimal .

We can see however that , and are the whole disk and so.

So in this case .

In this case the value of has an upper bound of and of .

Putting the above together, we get that our algorithm’s average total time is:

Since the robots have no information on what the correct direction will be, the probability of picking the correct direction will be 50%.

Note that the time needed for the robots to reach their starting position on the perimeter is 1 and is omitted in further calculations aimed to analyze the time of the “search and evacuate” part of the algorithm.

The time equation becomes the following:

Since the time for the wrong dimension uses for its calculations and since and varies between cases ( varies as well but will always be the same as the used in the paper’s algorithm) we must find the maximum time between cases in order to find a lower bound.

The time equation changes accordingly:

Which is:

≡

This time is greater than which validates Theorem 3.1.

## Improvement

We can try to improve the total time of the algorithm and there are two ways we can do that and both involve trying to reduce the value of . The first is to try and reduce the value of by allowing the Helper robots to find interesting points. The second one is to try and combine our algorithm with the suggested one as in the suggested algorithm the robots will scan a maximum of half a disk each instead of the whole disk of our algorithm.

### Improving the value of by allowing robots to find interesting points

We can try to improve the total time of the algorithm. The only time that can be improved is , the time needed for a robot to find an interesting point. We can try to improve it by allowing all robots, and not only , to find interesting points. If we do there is a predicted decrease in time by , the arc-distance between and the robots.

This however introduces one extra case in the two above, the case of an robot being inside the arc created by the exit and the treasure, while case 2 changes and means that no robot, not nor any of the robots, start inside the arc with arc-distance created by interesting points and .

### Case 3: Value of when is outside but is inside the arc

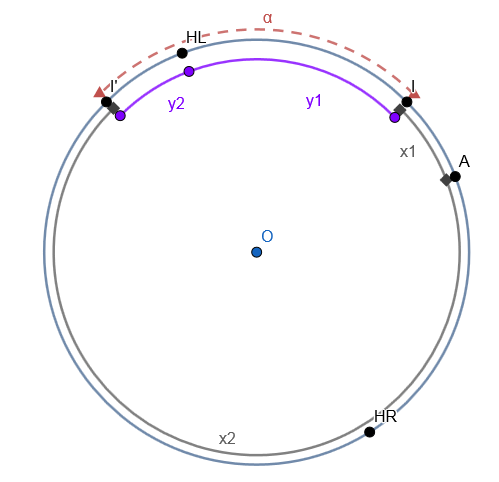


Figure 3.7:.Starting point outside and Helper robot inside of arc of interesting point

In this case one of the robots in the arc with arc-distance . (see Figure 3.7) Even though both and robots can find interesting points, our is still the distance of to an interesting point in order to compare this case to the other two.

is the arc which is also the optimal while is the arc. Should the robots move in the correct direction, then it is the optimal case. Should they pick the wrong direction of then would take time to reach but since the robots can now find interesting points and will reach either interesting point before reducing the time needed. will move towards on the arc with arc-distance . Let the arc be . As , then .

is the distance will travel if travels the arc so . As such, becomes and .

The total time needed become as follows (Ignoring the time needed to reach the starting position):

1. picks correct direction of :

Time for to find :

Total time:

1. picks wrong direction of :

Time for to find :

Total time:

Time for to find :

Total time:

The time needed for evacuation in this case changes and it can either be as the previous cases or. This is because if the interesting point that discovers is the exit then it would have to go pick up the treasure than head back to the exit doubling the time needed instead of discovering the treasure and then heading to the exit with it.

### How Case 2 is affected

Case 2 now changes and means than no robot starts inside the arc with arc-distance created by interesting points and . Since the robots can also detect interesting points, Case 2 changes as follows:

1. chooses correct direction of

Time needed for to find :

Total time:

Time for to find :

Total time:

1. chooses wrong direction of

Time needed for to find :

Total time:

Time for to find :

Total time:

The time needed for evacuation is either or for the reason described above.

## Combing both algorithms to improve time

Another way to improve time is to combine the two algorithms. Let the algorithm follow the suggested algorithm but with one change. Instead of ignoring the third robot, assign it as a “helper” to one of the other two robots. If and when the robot with the partner finds an interesting point first, the partner will be at the location of a possible interesting point. Because of that the robots will learn the final location of the interesting points before the other robot reaches his point. Because of this evacuation can be begin sooner than normally depending on the interesting point discovered.

The combined algorithm is as follows with the additions in an italicized font:

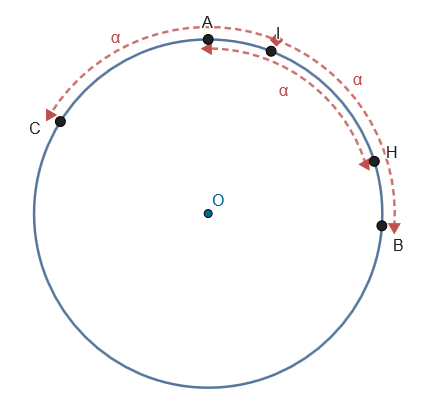


Figure 3.8: Starting positions of combined algorithm when

**Combined Algorithm**

1. If , then the two robots move together to an arbitrary point on the ring and start moving in opposing directions, else they move to arbitrary antipodal points , on the cycle and start moving in the same direction. *Helper robot arbitrarily chooses one of the other robots to help and moves to a point that is arc-distance α clockwise or counter-clockwise from its chosen robot’s location depending on the direction the chosen robot will move, as to be “ahead” of it. (see Figure 3.8)*
2. Let be the ﬁrst interesting point discovered by , at time . Let , be the points that are at clockwise and counter-clockwise arc-distance from respectively. *If the Helper robot is partnered with then it will be at point , giving the rest of the robots knowledge about the interesting points allowing them to move to the evacuation step. If the Helper robot is not partnered with then the algorithm continuous as normal but can help with evacuation if it is closer to than .*
3. If then robots learn that the other interesting point is in , else moves to , moves to .
4. Evacuate.

### Time of combined algorithm

The time needed for the combined algorithm differs from the suggested algorithm’s time depending on if the Helper robot is partnered with the robot that finds an interesting point first and even then it depends on the position of the interesting points.

#### Helper robot partnered with

Our cases are the same as the suggested algorithms with the single difference that when reaches point the Helper robot will be at point and will transmit to the other robots that .

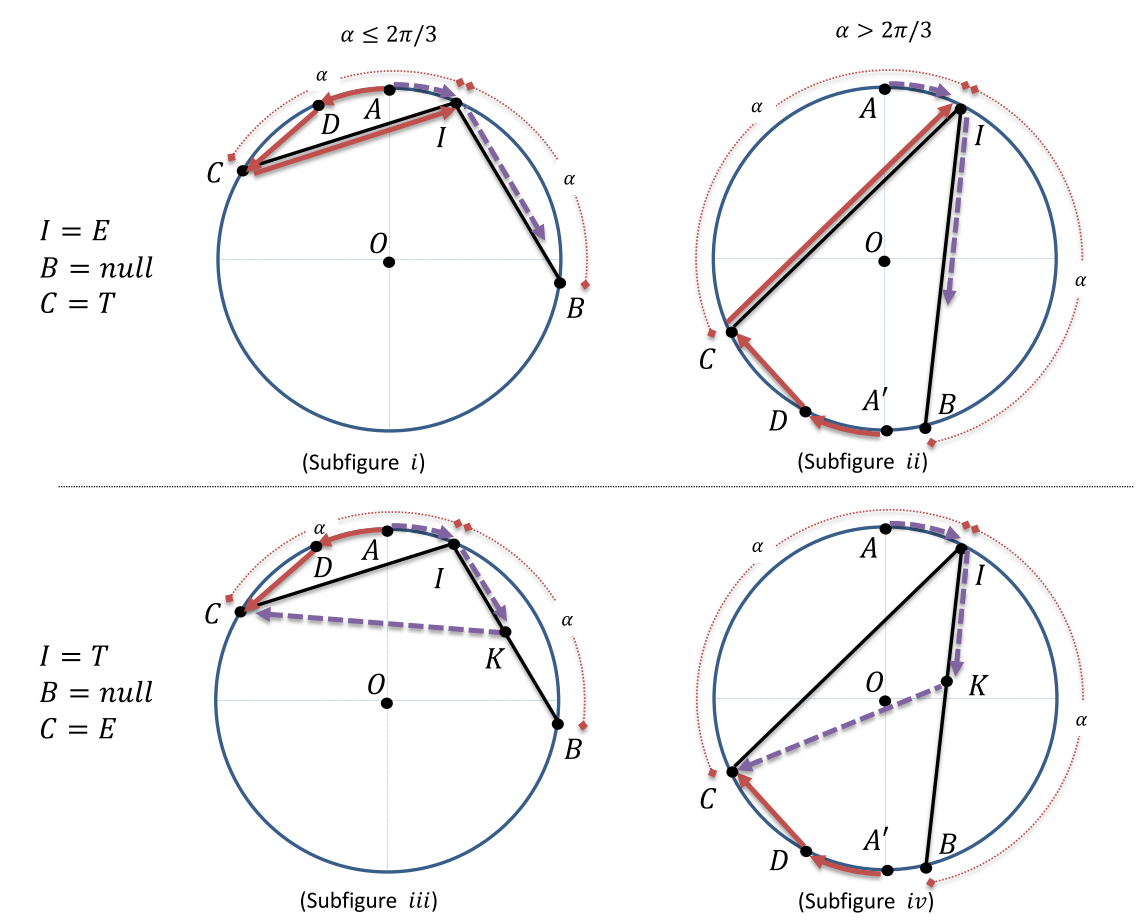


Figure 3.9: Case analysis of suggested algorithm when

**Case (i):** (, , & ): Let be at point , . the moves along chord , locates the treasure at point and head at exit to evacuate. In this case the helper robot does not provide an advantage as even if we know that , will still follow the same path in order to pick up the treasure and evacuate. The time will be the same as the suggested algorithm’s:

**Case (ii):** (, , & ): Let be at point ,. the moves along chord , locates the treasure at point and head at exit to evacuate. As in the case above the helper robot does not provide an advantage for the same reason. The time will be the same as the suggested algorithm’s:

**Case (iii):** (, , & ): When finds the treasure and picks it up, will be at point . In the suggested algorithm, will start heading towards while starts moving towards point on chord . When reaches , it transmits the information that the exit is at so starting from point heads to the exit on the line segment and evacuates. In the combined algorithm, when finds the treasure the helper robot will be at point and will transmit that . Because the exit must be at point and so will immediately head towards point . The time needed is:

**Case (iv):** (, , & ): When finds the treasure and picks it up, will be at point . Using the same logic as the case above, in the combined algorithm, picks up the treasure, at the same time it learns from the helper robot that and head towards exit without further delay. The time needed is:

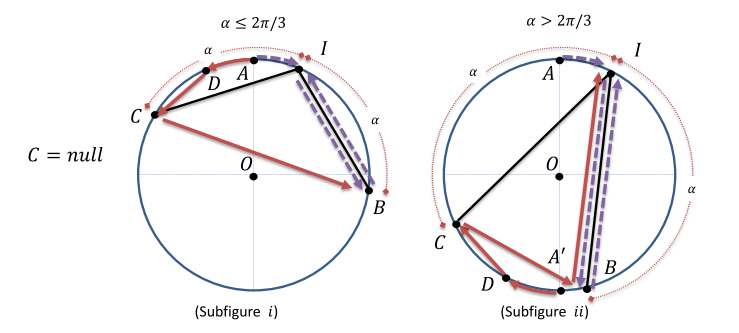


Figure 3.10: Case analysis of suggested algorithm when

In the case that and there is an interesting point at point the cases change as follows. finds interesting point and at the same time the helper robot finds an interesting point at point . The robot that finds the treasure picks it up and heads to the exit to evacuate. If finds the treasure it will head towards exit to evacuate and if the helper robot finds the treasure at point it will head towards exit to evacuate. In either case the time will be:

#### Helper robot partnered with

In the case that the helper robot is partnered with the robot that is not the first to discover an interesting point, the assistance that the helper robot provides is less than what it would provide by being partnered with . As shown below, when partnered with the helper robot will “overshoot” point and as such cannot check possible interesting point at the same time when locates point , similar to what it does with point when partnered with . (see Figure 3.11)

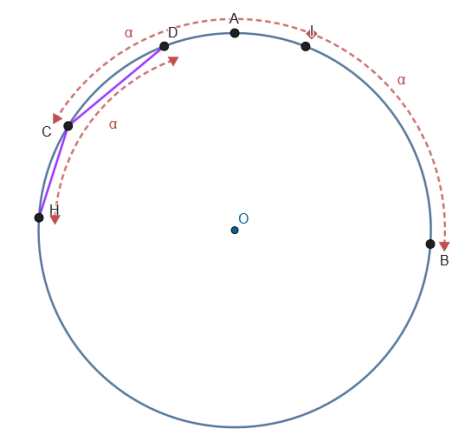


Figure 3.11: Helper robot "overshoots" when partner is at

In this case the helper robot can only help reduce the total time in a specific way. If it can reach point faster than in the cases when chord distance matters. When starts moving on chord in order to reach point , the helper robot can also start moving towards point on chord . Should the helper robot reach point faster than then we can achieve as small reduction in time.

This occurs when. The closer is to the starting point the further it is from point and so is closer to . If is closer to then reaches it first and if is closer to then the helper robot reaches it first. As such the greatest value of time needed to reach is when and are an equal distance away from . That means which happens when is the middle point of arc . Let be the middle of arc . That means that .When then . The time needed to reach , in the suggested algorithm, now has an upper bound of However distance plays a role in time calculation only when the other interesting point is at point .

### Total time of combined algorithm

The total time of the combined algorithm varies depending on whether the Helper robot is partnered with the robot that finds an interesting point first. Because we do not know which robot that will be, there is a 50-50 chance for the Helper robot to be partnered with it, or not. As such the total expected time becomes as following:

The total time for when the Helper robot is partnered with the robot that finds interesting point first is further depends on whether point or point are the other interesting point. If the other interesting point is at point then the time is:

This time has a probability of being less than the suggested algorithm’s in the same case.

If the other interesting point is at point then there is a further split in possible times. In 2 out of 4 cases when the other interesting point is at , when (, , & ) or (, , & ), the time is:

In the case when (, , & ) (Section 3.5.1.1 case (i)) the time is:

And in the case when (, , & ) (Section 3.5.1.1 case (ii)):

As we do not know whether the other interesting point will be at either point or , these two cases have an equal probability of occurring. The same applies to the 4 cases of when the other interesting point is at point . The total time for when the Helper robot is partnered with the robot that finds interesting point first becomes the following:

The total time for when the Helper robot is not partnered with the robot that finds interesting point first is the same as the suggested algorithm’s but the difference is that when the distance matters, that distance now has an upper bound to half of what it used to be, . Distance matters when the interesting point is at point while when the time needed is independent of the distance and the time is the same as the suggested algorithm’s. As such there is 50% chance of having some time reduction.

Because of cases like (i) and (ii) in Section 3.5.1.1 which are the same time as the suggested algorithms the upper bound of the combined algorithm’s time is the same as the suggested algorithm’s. However the probability of the combined algorithm’s time being the same as the suggested algorithm’s is:

That translates to

In all other cases we achieve the reduction of the total time needed for evacuation.

From all of the above, we have the following result:

**Theorem 3.2.** *The time needed for the combined algorithm has an upper bound of . The combined algorithm has a probability of of needing the same time as the suggested algorithm leading to the same upper bound. The combined algorithm has a probability of of needing less time than the suggested algorithm. The amount of the time reduction being,*, or *depending on the case.*

# Optimally Gathering Three Robots

4.1 Objective 30

4.2 Our algorithm 30

4.3 Analysis 33

4.3.1 Robot , case (a) 34

4.3.1.1 Robot activated before completes its COMPUTE phase 35

4.3.1.2 Robot not activated before completes its COMPUTE phase 36

4.3.2 Robot , case (b) 36

4.3.2.1 A robot is activated 37

4.3.3 Robot , case (c) 37

4.3.3.1 A robot is activated 37

4.3.3.2 A robot is not activated 37

4.3.4 Irregular case 38

## Objective

The objective of this part of our study is to design an algorithm that solves the gathering problem for three robots in the non-rigid ASYNC model as it is described in Section 2.2.1. While our robots are anonymous, for practical reasons they will be called , and in the rest of the paper.

## Our algorithm

For the robots to gather they need to head towards a common point. Because they cannot communicate except by using their light and because they are anonymous they cannot distinguish each other and agree to move towards any one of them, , or , while that one stays still. However as they are points on a two dimensional plane they form a shape and so they can aim to gather at a point of that shape that is common to all of them.

As there are three points the shape produced is a triangle. A common point that is as far from one robot as from the other, in an attempt to equalize the distance each robot travels, is the centroid. The centroid is the arithmetic mean position of all the points in the shape. In a triangle it is the point where the medians intersect (Medians are the lines joining each vertex with the midpoint of the opposite side). Even though each robot has its own coordinate system it obtains the location of each robot in it. When a robot takes a snapshot of each robots location in its local coordinate system, if the coordinates of each robot in are , , then the centroid is:

This centroid will be the destination that the robot that took the particular snapshot will aim for. A notable problem arises when by any chance the robots find themselves in a line. While the common point can become the centre of the line segment, the robots at the ends may not see each other as the one in the centre blocks their view. When this happens a special subroutine is executed that will make the robot move just enough in order to turn the line back into a triangle and the robots at the end can view each other.

Our algorithm follows:

|  |
| --- |
| **ASYNC 3 robot gathering with two colours (COMPUTE phase)** |
| **if** me.colour = Black **then**  **if** (me.position = other1.position) **and** (me.position = other2.position) **then**  do nothing  **if** (other1.colour = White) **or (**other2.colour = White) **then**  do nothing  **if** (other1.colour = Black) **and** (other2.colour = Black) **then**  me.colour = White  me.destination = Centroid  **if** me.colour = White **then**  me.colour = Black |

A robot takes a snapshot of the location and colour of each robot at the LOOK phase. Immediately after it enters the COMPUTE phase. If the other robots’ lights are Black then it sets the centroid as its destination and at the end of the COMPUTE phase it will switch its light’s colour to White. It then enters the MOVE phase and moves towards its destination but only if it has a destination set.

Because the goal point of the centroid will change and move depending on the movements the robots make, the algorithm makes it so that a robot will take its chance to move only when the other robots are not moving as their movements will affect the position of the centroid. In an attempt to coordinate the robots in order to gather, despite their asynchronous operation, we assign meaning to the two colours the lights can take. “Black” means that a robot is not moving and “White” means that a robot is moving. However because a robot can only “think” and change colour during the end of the COMPUTE phase it is not possible for a robot to change colour when it stops moving as that occurs during the MOVE phase. As such White now has the meaning of “moving or has moved during its previous activation”.

Below is a table showing the colour combinations the robots can take and the desired result that a robot will compute to based on the snapshot it took.

Table 4.1: Colour combinations of robots and result of operation

|  |  |  |  |
| --- | --- | --- | --- |
| Self | Other 1 | Other 2 | Result |
| Black | Black | Black | Turn White & Move to Centroid |
| Black | White | Black | Stay |
| Black | Black | White | Stay |
| Black | White | White | Stay |
| White | Black | Black | Turn Black |
| White | White | Black | Turn Black |
| White | Black | White | Turn Black |
| White | White | White | Turn Black |

After a robot has moved its light’s colour stays White. Any robots that activate from their WAIT phase after this point will see that a robot’s light’s colour is White and will COMPUTE to go back to waiting. It may be possible that the adversary will keep on activating the robots whose light’s colour is not White and the above scenario will keep on repeating when what we need is for the adversary to activate the robots whose light’s colour is White in order to turn it back to Black and this will result in a deadlock.

This is the other liveness property of our system, one that makes the adversary fair using fair scheduling. Even if the adversary keeps on activating the robots according to the scenario above, fair scheduling dictates that after a number of times the robot whose light is White will activate and so the deadlock that would arise from this scenario will be avoided.

However, the model is ASYNC so our attempts at coordinating the robots might not work. For example, let us suppose that all robots are Black and waiting for activation. The adversary then wakes one of them. That robot looks and sees that his and all other robots lights are Black. It will then compute that it will switch to White and start moving. Before it can change its light’s colour another robot is activated and enters its LOOK phase taking a snapshot and according to that snapshot the robot that will change to White appears to be Black. As such we can have two robots moving at once when we attempted to only have one robot moving at a time. A case analysis of the algorithm is needed in order to test its correctness.

**Theorem 4.1.** *After every MOVE phase, the distance between the robot, that executes the MOVE phase, and the other robots is reduced.*

**Proof:** The proof considers the cases studied in Section 4.3.

## Analysis

Below is a diagram showing how a robot operates after being activated from its WAIT phase and how it transits from one phase to another.

When a robot is activated from its WAIT phase it executes the flowchart shown in Figure 4.2 with the MOVE phase being optional, depending on what the algorithm tell it to do. The LOOK phase is assumed to be instantaneous and the robot immediately enters the COMPUTE phase after it. The COMPUTE phase (a), the MOVE phase (c) and the delay between them (b) are chosen by the adversary and can be arbitrary long but are finite. It is in these three cases when a robot, other than the one executing the cycle, can be activated and mess up our attempts at coordinating the robots and having the system operate linearly.

LOOK

COMPUTE

COMPUTE END

MOVE

MOVE END

(a)

(b)

(c)

Figure 4.2: Time durations and pauses during a robot's operation

However, the only problematic phase is phase a), for the following reason. At the end of the COMPUTE phase a robot may change its colour to White which will shut down the other robots attempts at moving. For any problems to occur, the robots must be activated from their WAIT phase and enter their LOOK phase while another robot is in its COMPUTE phase.

It does not matter which robot is activated first but all three robots must be activated and execute one cycle in order to see if they behave outside of the algorithm’s expectations. In the following example, is activated first followed by and then by .

### Robot , case (a)

When all three robots are in their WAIT phase, the adversary wakes up robot . Robot enters its LOOK phase and takes a snapshot of its surroundings. Immediately after it moves to its COMPUTE phase. While in the middle of its COMPUTE phase (a), two things can happen:

1. Robot is activated from its WAIT phase.
2. Robot is not activated and completes its COMPUTE phase.

Between these two options, number 2 is the one desired, because when a robot is activated after completes its COMPUTE phase, it will take a correct snapshot of its surroundings. If a robot is activated before robot completes its COMPUTE phase, the robot will have a wrong snapshot of its surroundings because the state of robot in that snapshot will be outdated when robot completes its COMPUTE phase

#### Robot activated before completes its COMPUTE phase

In the case where robot is activated while robot is in the middle of its COMPUTE phase, robot will take a snapshot of its surroundings that will become wrong once robot finishes its COMPUTE phase. In this example, robot will also take the same snapshot as robot , in which every robots’ colour is Black and as such it will change its light’s colour to White and start moving towards the destination it calculated. Before it finishes its COMPUTE phase there are two that can happen:

1. Robot is activated from its WAIT phase.
2. Robot is not activated and completes its COMPUTE phase

Option number 2 is the optimal one as robot will avoid taking a wrong snapshot and operating incorrectly.

* **Robot activated before completes its COMPUTE phase**

In this case, robot is activated from its WAIT phase while robots and are in their COMPUTE phase and robot has taken a wrong snapshot. Just like robot , robot will take a wrong snapshot and think that since all robots’ colours are Black it will change its light to White and begin moving towards the destination it calculates.

This case will have all robots deciding to change their colours to White and moving towards the centroid of the triangle in their snapshots.

When this happens, robots and are operating “incorrectly”, they deviate from our desired operation of one robot moving at a time. They think that they are the only ones that are active and are going to move towards their destination. The destination they calculate can become incorrect when the other robots move causing their location to change and the snapshot that the destination is based upon becomes outdated. However, this scenario occurs when robot is activated from its WAIT phase before robot completes its COMPUTE phase and when robot is activated before robot and complete their COMPUTE phase, since when a robot completes its COMPUTE phase and changes its light’s colour to White it will block other robots activating later from moving. Not completing the COMPUTE phase means that no movement has been done either and that means that the snapshot the robots took is the same and so they also have the same destination. As such, when the robots enter their MOVE phase, no matter the order, they will all move towards the **same destination** and even if anyone of them is stopped by the adversary before reaching it they will have still inched a little closer to each other. As such progress is being made even when the robots are operating “incorrectly” with multiple robots moving at the same time.

Even then though, it can lead into the **Irregular case** if any two of the three robots keep on activating and completing their cycles while the third one is still in the middle of its COMPUTE phase with the snapshot it took quite a few cycles back. This Irregular case leads to something that appears to be a deadlock but upon closer inspection it is not. This case is explained in Section 4.3.4.

* **Robot not activated before completes its COMPUTE phase**

In this case, robot completes its COMPUTE phase and changes its light’s colour to White. If robot awakens from its WAIT phase it will LOOK and see that robot ’s light is White and will COMPUTE to go back to waiting. Robot is blocked from making any movement that may desynchronize the system any further.

#### Robot not activated before completes its COMPUTE phase

Robot will complete its COMPUTE phase before robot awakens and changes its light to White. When robot awakens from its WAIT phase it will see that robot ’s light is White and will COMPUTE to go back to waiting. No robot will move before robot stops moving and no robot has taken a wrong snapshot. This is the most optimal case.

### Robot , case (b)

Robot has completed its COMPUTE phase and now it will either wait for some time before entering its move PHASE or will enter its WAIT phase. In this example, since robot is the first robot to awaken and take a snapshot, it will switch its lights colour to White and begin to move at its MOVE phase because in the snapshot it took, all the other robots lights are Black. In the delay between its COMPUTE and MOVE phases (b), two things can happen:

1. A robot is activated from its WAIT phase, if it was not activated in 4.4.1.
   1. If no robot was activated in 4.1 then robot will activate.
   2. If was activated in 4.4.1 then will activate, if it was not activated during ’s cycle.
2. A robot is not activated and robot enters its MOVE phase.
   1. A robot is not activated while and have not activated yet.
   2. A robot is not activated while has not activated yet.
   3. A robot is not activated because there are no more robots to activate.

From this list of options, option number 2.1 is the optimal one as it means that so far the robots are operating linearly which will lead to no mistakes.

#### A robot is activated

Following robot ’s COMPUTE phase another robot is activated. Regardless of which robot it is, if it is activated from its WAIT phase it will see that robot ’s light is white and will COMPUTE to go back to waiting.

### Robot , case (c)

Robot is now entering its MOVE phase and starts moving towards the destination it calculated in the COMPUTE phase based on the snapshot of the robots’ position that it took in its LOOK phase. Similarly to the case above, two things can happen:

1. A robot is activated from its WAIT phase while robot is moving.
2. Robot stops moving (because it has reached its destination or stopped by the adversary) before other robots are activated.

#### A robot is activated

If a robot is activated from its WAIT phase while robot is moving, it will see that robot ’s light is White and will COMPUTE to go back to waiting.

#### A robot is not activated

Robot completes its movement, either because it has reached its destination or because it was stopped by the adversary. Following activations of other robots from their WAIT phase will result in them COMPUTING back to waiting as robot ’s light is still White even after stopping. Robot must awaken again in order to change its colour to Black, regardless of what the other robots’ colour are, in order to give the other robots the chance to move. Fair scheduling ensures that at some point robot will awaken again and change its colour to Black which makes us avoid the deadlock of robots and awakening only to go back to waiting upon seeing that robot ’s colour is White.

### Irregular case

This case occurs when a robot is stuck in the middle of its COMPUTE phase while the remaining two continue operating normally and going through LOOK-COMPUTE-MOVE cycles. For example, robot awakens from its WAIT phase while all the other robots’ colours are Black and takes a snapshot with the current centroid as its destination. As all other robots’ lights are Black, robot will change its light’s colour to White at the end of its COMPUTE phase and will then move. However, before it completes its COMPUTE phase and turns its light’s colour into White, something that will block the other robots from moving, the other robots awaken from their WAIT phase and will start moving following case 4.4.1.1.1. (also see Figure 4.3)

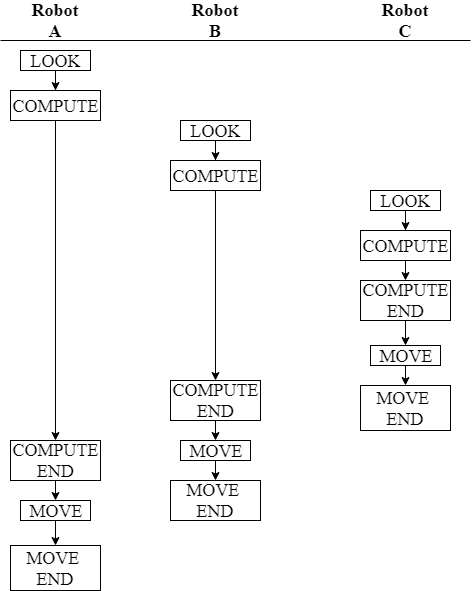


Figure 4.3: Example of Robot B and C being activated before Robot A finishes its COMPUTE phase

Without going into any details on how they operate, the point is that robots and will move and complete their cycles before robot completes its COMPUTE phase. Following this, they will then awaken and move again, while robot is stuck in its COMPUTE phase from long ago.

The movement of robots and will bring them closer to each other as well as to robot . At one point, as they get closer to , this will result in the destination that robot will calculate, based on the snapshot it took before robots and moved, being outside of the current triangle (see Figure 4.4). When finally completes its COMPUTE phase, it will begin moving towards its calculated destination and at one point, if the adversary doesn’t stop it when it passes robots and , will actually be moving away from the other robots.

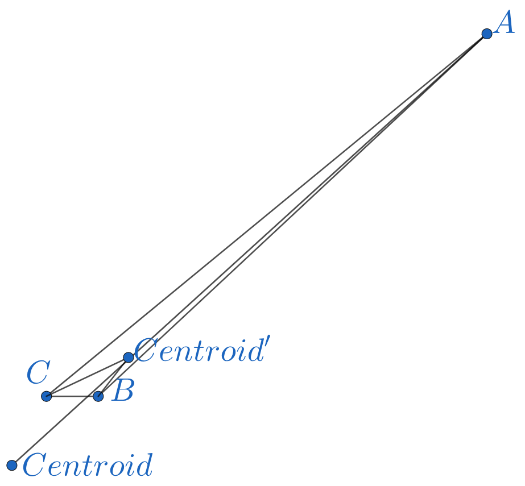
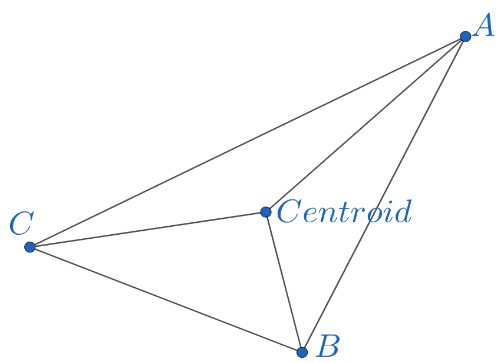
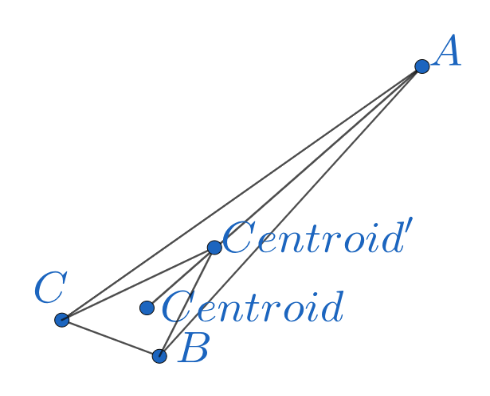


Figure 4.4: Example of Irregular case

An example of the Irregular case is shown in Figure 4.4. The shape on the top left is our initial configuration. The robots will operate as shown in Figure 4.3 and the destination they calculate is . Following that, robots and will move towards that destination and they will either reach it before stopping or will be stopped by the adversary before reaching it.

In the shape on the top right, robots and moved towards the centroid but were stopped before reaching it. Note that robot has not moved yet and its destination is . Before the COMPUTE phase of robot ends, robots and will be activated again, switching their light’s colour to Black as per Table 4.1. Once again they will activate again before completes its COMPUTE phase. The new destination they calculate is and they will begin to move towards it.

This is shown in the shape on the bottom. Robots and moved towards their new destination of but were stopped before reaching it. Here is where the irregular case shows itself. The destination that robot has is the one it calculated initially, that is . If it moves towards it, it may pass robots and and move away from them, the opposite of what we want.

While this irregular case does make it seem that the robots are working towards undoing any progress by distancing themselves there is a catch. Even if robot moves away from the other robots towards the destination it calculated long ago, that destination is still the centroid of the triangle in its snapshot. The maximum distance that a robot can move away from the other robots is the distance from its position during the LOOK phase to the centroid, this happens when the other two robots manage to reach robot ’s position. For negative or no progress to be done the distance that robot travels must put it outside of or on the boundaries of the original starting triangle. This distance is greater than the distance to the centroid making it impossible for no progress to be made. On the contrary progress will still be made as if and when this irregular case occurs again, the new destination of robot will be the centroid of a smaller triangle, with the area of the triangle decreasing meaning that the robots are closing in on each other. The distance that robot moves away will slowly be reduced which at one point will put it inside the range that the robots consider themselves to be gathered.

We can see that after every MOVE phase, the robot who moves will get closer to the other robots which leads to them gathering. This proves Theorem 4.2.

**Theorem 4.2.** *The algorithm presented in Section 4.2 solves the gathering problem for three robots.*

# Conclusion and Future Work

In this thesis, we developed an algorithm that solves the *search-and-fetch* treasure evacuation problem with three robots, instead of two, as it was introduced in [1]. Our objective was for our solution to have a lower upper bound of time needed, than the one proposed in [1]. With three robots we were able to use the morphology of the problem to radically reduce the time needed for the *evacuation* part of the problem. However, because all three of the robots are used to decrease the time needed for evacuation, the *search* part ended up suffering with the time needed for it increasing. Following that, we combined our algorithm with the one of [1] in order to get the best of both worlds. This combined algorithm has the same upper bound as the suggested algorithm but can reduce the time needed for either the search or the evacuation part of the algorithm. This algorithm has a probability of of needing the same time as the suggested algorithm, while the rest of the time it can reduce the amount of time needed, compared to the time needed for the suggested algorithm. There are a number of open problems related to this variation of the search-and-fetch problem. These include how the solutions changes when we have an even greater number of robots, multiple exits and/or treasures, a different geometric domain than a disk and differing robot speeds. Additional knowledge about the topology or the search space changes the problem and its solution.

Following that, we have presented an algorithm that solves the problem of gathering three robots in non-rigid ASYNC model with an optimal number of lights. Because of the asynchronous execution model, there is the possibility to start from any possible configuration but even if that happens our algorithm will always try to stabilize itself to only one robot moving at a time. We have also shown that even if more than one robots move at the same time, which goes against our normal mode of operation, they will still make progress towards gathering. Even the case, in which it looks like the robots are going against our algorithm’s logic and undoing its progress (see Chapter 4.3.4), is only a setback that will still lead to the robots gathering. This algorithmic solution is only possible due to the two liveness properties granted by the model, the first being fair scheduling and the second being the guarantee that when the robots move, they will move at least distance, with . In theory, this algorithm can be used to gather any number of robots as long as they can find a common point they can head to in order to gather. That point will most likely be the centroid again with the robots finding it using a method appropriate for the shape they take. Open problems to robot gathering include gathering in three dimensional space and the gathering of “fat” robots, in which robots have a large volume and may block each other’s movements, although we expect the premise of the solution to stay the same.

###### References

|  |  |
| --- | --- |
| [1] | Konstantinos Georgiou, George Karakostas and Evangelos Kranakis. *Search-and-Fetch with 2 Robots on a Disk: Wireless and Face-to-Face Communication Models*. 1 December 2016. [Online]. Available: arxiv.org/abs/1611.10208. |
| [2] | Ichiro Suzuki and Masafumi Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing,* 28(4), p. 1347–1363, 1999. |
| [3] | Paola Flocchini, Giuseppe Prencipe and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. |
| [4] | Adam Heriban, Xavier Défago and Sébastien Tixeuil. *Optimally Gathering Two Robots*. 21 August 2017. [Online]. Available: https://arxiv.org/abs/1708.06183. |
| [5] | R. Ahlswede and I. Wegener. *Search problems*, Wiley-Interscience, 1987. |
| [6] | S. Alpern and S. Gal. *The theory of search games and rendezvous*. Springer, 2003. |
| [7] | L. Stone. *Theory of optimal search.* Academic Press New York, 1975. |
| [8] | P. Berman. On-line searching and navigation. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, p. 232–241. Springer, 1998. |
| [9] | E. Koutsoupias, C. Papadimitriou and M. Yannakakis. *Searching a ﬁxed graph*. In *ICALP 96*, p. 280–289. Springer, 1996. |
| [10] | Vadim Bulitko, Yngvi Bjornsson, Nathan R. Sturtevant and Ramon Lawrence. *Real-time Heuristic Search for Pathﬁnding in Video Games*. 7 July 2010. [Online]. Available: http://www.ru.is/~yngvi/pdf/BulitkoBSL10.pdf. |
| [11] | F. Kolovský and J. Ježek. *Algorithm for searching a shortest path obeying restricted areas without preprocessing*. 3-6 November 2015. [Online]. Available: http://project.opentransportnet.eu/otn/sites/default/files/Sigspatial.pdf. |
| [12] | P. Nahin. *Chases and Escapes: The Mathematics of Pursuit and Evasion*. Princeton University Press, 2012. |
| [13] | J. Czyzowicz, L. Gasieniec, T. Gorry, E. Kranakis, R. Martin and D. Pajak. Evacuating robots from an unknown exit located on the perimeter of a disc. In *DISC 2014*, p. 122-136. Springer, Austin, Texas, 2014. |
| [14] | J. Czyzowicz, K. Georgiou, S. Dobrev, E. Kranakis and F. MacQuarrie. Evacuating two robots from multiple unknown exits in a circle. In *ICDCN 2016*, 2016. |
| [15] | J. Czyzowicz, K. Georgiou, E. Kranakis, L. Narayanan, J. Opatrny and B. Vogtenhuber. Evacuating robots from a disc using face to face communication. In *CIAC 2015*, p. 140-152. Springer, Paris, France, 2015. |
| [16] | J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny and S. Shende. Wireless autonomous robot evacuation from equilateral triangles and squares. In *ADHOC-NOW 2015, Athens, Greece, June 29-July 1, 2015, Proceedings*, p. 181-194, Athens, Greece, 2015. |
| [17] | B. Gluss. An alternative solution to the lost at sea problem. *Naval Reserch Logistics Quarterly,* 8(1), p. 117–122, 1961. |
| [18] | J. S. Jennings, G. Whelan and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *ICAR,* p. 193–200. IEEE, 1997. |
| [19] | Pierre Courtieu, Lionel Rieg, Sébastien Tixeuil and Xavier Urbain. Impossibility of gathering, a certification. *Information Processing Letters (IPL),* 115, p. 447–452, 2015. |
| [20] | Yoann Dieudonné and Franck Petit. Self-stabilizing gathering with strong multiplicity detection. *Theor. Comput. Sci.,* 428, p. 47–57, 2012. |
| [21] | Xavier Défago, Maria Gradinariu Potop-Butucaru, Julien Clément, Stéphane Messika and Philippe Raipin-Pervédy. Fault and byzantine tolerant self-stabilizing mobile robots gathering. IS-RR-2015-003, Japan Adv. Inst. of Science and Tech. (JAIST), Hokuriku, Japan, February 2015. |
| [22] | Xavier Defago, Maria Gradinariu Potop-Butucaru, Stéphane Messika and Philippe Raipin-Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering: Feasibility study. In *Proc. 20th Intl. Symp. Distributed Computing (DISC),* volume LNCS 4167, p. 46–60, September 2006. |
| [23] | Noa Agmon and David Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.,* 36(1), p. 56–82, 2006. |
| [24] | Zohir Bouzid, Shantanu Das and Sébastien Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS 2013)*, p. 337–346, Philadelphia, PA, USA. IEEE Press, July 2013. |
| [25] | Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science,* vol. 609, p. 171 – 184, 2016. |
|  |  |