Monitoring and behavior assessment of advisory robotic agents using low-computing techniques

Thesis submitted by

Bappaditya Das

Doctor of Philosophy (Engineering)

Department of Computer Science and Engineering
Faculty Council of Engineering And Technology
Jadavpur University
Kolkata, India

2024

Index No: 177/17/E

1. **Title of the Thesis:**

   MONITORING AND BEHAVIOR ASSESSMENT OF ADVISORY ROBOTIC AGENTS USING LOW-COMPUTING TECHNIQUES

2. **Name, Designation and Institution of the Supervisor:**

   Dr. Chintan Kumar Mandal
   Assistant Professor
   Department of Computer Science and Engineering
   Jadavpur University
   Kolkata-700032

3. **List Of Publications:**

   Journals:

   (a) **Das, B.**, Mandal, C. K. (2023). Prediction of behavior of a noncommunicative robot based on observations of its movement. Concurrency and Computation: Practice and Experience, 35(22), e7725.

   (DOI: https://doi.org/10.1002/cpe.7725)

   (b) **Das, B.**, Mandal, C. A revisit to the planar subdivision graph: free space detection in a dynamic environment with polygonal obstacles. Int. j. inf. tecnol. 16, 779–791 (2024).

   (DOI: https://doi.org/10.1007/s41870-023-01646-4)

   (c) **Das, B.**, Mandal, C. (2024). An Overlapping Circle Covering Problem For a Box With Polygons. (Communicated)

4. **List Of Patents:** None

5. **List of Presentations in National/International/Conferences/Workshops:**

   (a) A. Basu, **B. Das** and C. K. Mandal, "A Method for Finding Multiple Large Rectangular Free Spaces in a Map with Convex and Concave Obstacles," 2021 IEEE 18th India Council International Conference (INDICON), Guwahati, India, 2021, pp. 1-6,

   (DOI: https://doi.org/10.1109/INDICON52576.2021.9691569)

   (b) **Das, B.**, Mandal, C. (2024). A Genetic Algorithm-Based Heuristic Approach for Strategically Placing Circles Amid Convex Polygons. (Communicated)

Statement of Originality

I, **Bappaditya Das** registered on **21st July 2017**, do hereby declare that this thesis entitled "**Monitoring and behavior assessment of advisory robotic agents using low-computing techniques**" contains literature survey and original research work done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the "Policy on Anti Plagiarism, Jadavpur University, 2019", and the level of similarity as checked by iThenticate software is 2%.

Bappaditya Das 26/06/2024

Bappaditya Das
Registration No: D-7/e/500/17 of 17-18
Roll No: 101710505021

26/06/24

ASSISTANT PROFESSOR
Dept. of Computer Sc. & Engg.
JADAVPUR UNIVERSITY
Kolkata - 700 032

Dr. Chintan Kumar Mandal,
Assistant Professor,
Department of Computer Science and Engineering,
Jadavpur University
(Supervisor)

Certificate from the Supervisor

*This is to certify that the thesis entitled "***Monitoring and behavior assessment of advisory robotic agents using low-computing techniques***" submitted by* ***Bappaditya Das****, who got his name registered on 21st July 201l7, for the award of Ph.D. (Engineering) degree of Jadavpur University, is absolutely based upon his own work under the supervision of Dr. Chintan Kumar Mandal and that neither his thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.*

26/06/24

ASSISTANT PROFESSOR
Dept. of Computer Sc. & Engg.
JADAVPUR UNIVERSITY
Kolkata - 700 032

Dr. Chintan Kumar Mandal,
Assistant Professor,
Department of Computer Science and Engineering,
Jadavpur University
(Supervisor)

Acknowledgments

I would like to thank my family and my supervisor Dr. Chintan Mandal for all the help and support I have received during this work. Without their help, this work would not have been possible.

I am indebted to Convenor, members of PhD. Committee, Head Of The Computer Science and Engineering Dept, Jadavpur University for their valuable feedback during my research work. Further, I take this opportunity to acknowledge that my academic enrichment through coursework at the Computer Science and Engineering Dept, Jadavpur University developed a positive mindset which proved to be an essential component in the research work.

Bappaditya Das
26/06/2024

Bappaditya Das
Computer Science and Engineering Dept
Jadavpur University
Kolkata - 700032
West Bengal, India

Abstract

When a hostile robot or agent navigates through an unobstructed area having obstacles, the ability to forecast its forthcoming movements or *"behaviour"* becomes crucial in mitigating potential security risks or other forms of threats posed by these entities. In order to forecast future movements, it is essential to monitor the open space through cameras equipped with a 360-degree or panoramic view, enabling vigilant observation of the agents' activities and facilitating strategic planning in response. Traditional complex stochastic or machine learning models employed for predicting future states demand significant computational resources. This study introduces alternative techniques aimed at forecasting the *"behaviour"* of adversarial agents traversing freely within an environment, leveraging their current and past behavioral patterns. These techniques are tailored for implementation in low-power robots or devices, where deploying costly stochastic or machine-learning models may not be feasible. Diverging from the prevalent literature that relies on temporal predictions, these methods forecast future states based on the system's current states. Moreover, they are adaptable to scenarios involving both singular and multiple-looped paths followed by the adversary robot.

Considering the 360^0/panoramic perspective of camera coverage resembling circles, this study demonstrates that positioning circles to encompass the open space within environments containing convex and concave polygons presents an NP-complete problem. The expanse of the open space requiring coverage far surpasses the area that a single circle can encompass. Hence, the strategic arrangement of circles within the open space becomes crucial. This work offers an approximate solution for strategically situating overlapping circles within the open space. The methodology aims to minimize the overlap between polygons and these circles as much as possible. A heuristic method based on a Genetic Algorithm is also proposed to cover the free space of an environment containing convex polygons with circles, ensuring that the overlap between placed circles and polygons stays within a predetermined value and

that no overlap occurs between the placed circles.

In this proposed approach, following the placement of a circle, the environment undergoes adjustments to designate the circle's area as occupied space, thus preventing redundant coverage by subsequent circles. This approach hinges on an additional proposed method for generating a planar subdivision graph in a given environment. This method involves dynamically adding or removing polygonal obstacles, akin to the regeneration of a map following a natural disaster.

Contents

1 Introduction **11**

2 Free Space Detection in a Dynamic Environment with Polygonal Obstacles **16**

2.1 Literature Survey 16
2.2 Offline Algorithm for detecting Free Space in a Map 19
2.2.1 The Initial Root Trapezoids (IRT) 20
2.2.2 Design of the Hash Table 21
2.2.3 Construct Trees With New Trapezoids 22
2.3 Dynamic Algorithm for Detecting Free Space due to the Insertion/Deletion of Polygons 26
2.3.1 Insertion of a Polygon in a Map 27
2.3.2 Deletion of a Polygon from the Map 30
2.4 Analysis for the Dynamic Algorithm 33
2.5 Comparison with Some Similar Algorithms 36
2.6 Conclusion 37

3 Placement Of Vision Circles **38**

3.1 Literature Survey 39
3.2 Problem Classification For Circle Placements In Map With Polygons . 43
3.3 A Discrete Method for Strategic Placement of Vision Circles In Presence of Convex and Concave 48
3.3.1 Finding Free Rectangular Spaces 49
3.3.2 Placement of the Vision Circles 55
3.3.3 Revising the Map with the Vision Circles 59
3.4 A Heuristic Method Using Genetic Algorithm for Strategic Placement of Vision Circles In Presence Of Convex Polygons 65
3.4.1 Genetic Operators To Be Applied for Circle Packing 67

3.4.2 Handling Overlaps with Convex Polygons 68
3.4.3 Objective Function Formulation 69
3.4.4 Implementation of Genetic Algorithm 71
3.4.5 Experimental Results 72
3.5 Complexity Analysis Of Proposed Discrete Method For Placement Of Circles 74
3.5.1 Complexity Of algorithm to construct Map_R 75
3.5.2 Complexity Of algorithm to place circles in the map 78
3.5.3 Review And Analysis Of Some Related Algorithms 79
3.6 Conclusion 79

4 Prediction of Behaviour of a Non-communicative Robot 81
4.1 Literature Survey 82
4.2 Theoretical Basis 84
4.3 List Based Approach for the Behaviour Analysis 86
4.3.1 Behaviour Analysis With List $C_1(\theta)$ 87
4.3.2 Update of $C_2(R, d)$ 89
4.3.3 Prediction using the Behaviour Analysis 90
4.3.4 Complexity 92
4.4 A Graph-Based Approach: Alternative Solution for the Behaviour Analysis for 2-DOF 93
4.4.1 Data Representation and Construction of the "Behavioural Network" 93
4.4.2 Prediction using the Behavioural Network $G(V, E)$ 95
4.4.3 A "Markov Chain" equivalent of "Behavioral Network" 97
4.5 Comparative study between the proposed algorithms 98
4.5.1 Analysis of Update Activity in List-Based Approach 99
4.5.2 Analysis of Update Activity in "Behavioral Network" 101
4.6 Conclusion 102

5 Conclusion and Future Works 104

List of Symbols

Symbol	Description
$\mathbf{PSM}$	Planar Subdivision Map
$\mathbf{P_{pi}}$	Vertex i of polygon p
$(\mathbf{P^x_{pi}}, \mathbf{P^y_{pi}})$	x and y coordinates of polygon vertex P_{pi}
$\mathbf{IRT}$	Initial Root Trapezoid
$\mathbf{G_i}$	Group Number i
$\mathbf{GL(G_i)}$	Group Leader of Group Number i
$\mathbf{\Lambda}$	Mapping between vertex and its corresponding group leader
$\mathbf{SVtx}, \mathbf{TVtx}$	Left and right endpoint of line segment
$\mathbf{\Omega}$	Mapping between the group leader and line segments in the corresponding slab arranged from top to bottom
$\mathbf{IHN}$	Insertion Hit Node, a leaf node of the tree which will be updated due to an edge of an inserted polygon
$\mathbf{DHN}$	Deletion Hit Node, a leaf node of the tree which will be deleted due to an edge of a deleted polygon
$\mathbf{L_p}$	List of convex and concave polygons inside bounding box
$\mathbf{S}$	Slabs of planar subdivision
$\mathbf{G_u(V, E)}$	Graph constructed from free trapezoids of planar subdivision
$\mathbf{L_P Trap}$	Mapping between polygons in L_p and list of trapezoids covering each polygon
$\mathbf{Map_R}$	Mapping between the free trapezoid and rectangular free space
$\mathbf{E_w}$	Mapping between edges and corresponding edge weights
$\mathbf{map}$	Rectangular bounding box with polygons within it
$\mathbf{N_v}$	Total number of vertices in the map
R_{Gap}	Unit step length in a translation movement
R	Multiple of R_{Gap}
θ_{Gap}	One unit of rotational angle
θ	Multiple of θ_{Gap}
d	Linear direction of translation movement, reverse($R/-1$), forward($F/+1$) or pause($S/0$)

R_{Max}	Wheel circumference
R_{Obs}	An observed translation movement
θ_{Obs}	An observed rotational movement
s_{Obs}	An observed state represented as tuple $\theta_{Obs}, (R_{Obs}, d)$
$C_1(\theta)$	List of θ
$C_2(R, d)$	List of tuples $< R, d >$
$G(V, E)$	Graph representing motion behaviour, where V is a set of vertices, each vertex represents a state and E is a set of weighted directed edges between the two states

Chapter 1

Introduction

Robotic systems are mechanical systems that can perform tasks under human instructions or be programmed to make decisions using artificial intelligence techniques. In today's time, robotic systems are used in defence, exploration, agriculture, hazardous environments, factories etc. Robots that can learn from their surroundings and make their own decision while performing actions are considered autonomous.

Primary research areas for autonomous robots include motion planning [47], obstacle avoidance using image processing [27], map generation using SLAM [43] etc. This work studies the prediction of robot movements by an autonomous robot/agent in an enclosed region. A robot or human agent possesses the ability to navigate a given area based on its instinctual behaviours. Foreseeing the future movements of such an entity proves to be quite difficult for an observer. However, in scenarios like a war zone or a multi-robot soccer match, accurately predicting the imminent actions of opponents becomes essential for devising effective countermeasures. This task often requires significant computational resources, whether through a dedicated server or cloud computing, to convert predictions of the entity's movements into actionable paths within the designated area.

In a designated area, when a surveillance robot must observe agents from an opposing faction, it faces constraints on carrying extensive computational resources and power due to the necessity of staying hidden from adversary agents. This challenge is especially pronounced in war, combat, or disaster zones where GPS or internet connectivity is unavailable, making it significantly harder to track the movements of these adversaries. Without the ability to rely on receiving extensive historical movement data over the network, the surveillance robot's mission becomes even more

complex.

In a multi-robot soccer game, [11], each team member can receive commands for overall team strategy through global vision and mix that with its local vision for motion planning and obstacle avoidance. To make the robots truly autonomous, each robot must use only its local vision to plan its motion without any suggestion from off-field computers which have access to the information related to the motions of all robots in the field. These moving autonomous robots also need to be small as they cannot afford to carry large computational resources and battery power while playing.

While the robots' movements may appear random to observers, there may be concealed patterns within these observed movements. These seemingly random patterns can be designated as the robot's "*behaviour*". The finite sequence of these observations can be considered the state of the system for which the observations were conducted. The robot's "*behaviour*" is intrinsically linked to its state.

A surveillance robot can utilize the "*behaviours*" of an adversary robot to determine its future steps in this constrained environment with a lack of connectivity and low computing power. In the case of a multi-robot soccer game, an individual robot needs to observe the "*behaviour*" of other robots to predict their future states and plan accordingly. An observatory camera can record the pattern/behaviour of an adversary robot's movements to predict its future steps. These future steps when combined into a sequence will aid in giving a "predicted path".

Stochastic or machine learning techniques, which can be used to predict the future states of an adversary robot/agent demand high computational resources and power. Advanced "Recurrent Neural Network" techniques like "Long Short-Term Memory", which can make predictions using long-term dependencies of the sequence data, also have extremely high computing and memory requirements. Thus, the systems involved in using these techniques are not appropriate to be deployed in fields requiring small robots' inconspicuous monitoring. Some techniques inspired by human behaviour use sophisticated motion models to track people. Belhadi et.al. [6] have shown that normal and abnormal behaviours of human beings can be separated after finding outliers in a crowd.

This work proposes two novel methods to predict the future states of an adversary robot using the current and past *"behaviour"*. The proposed methods can be deployed in a low-computing device/robot. Low-computing devices are usually charac-

terized as low-power or having low performance. These characteristics meet specific needs, such as minimal power consumption, cost efficiency, and basic functionality. These devices typically include microcontrollers, single-board computers (such as the Raspberry Pi), and FPGAs. They are often compact and miniaturized, designed for low power consumption to extend battery life. Their processing capabilities are limited, featuring 8-bit, 16-bit, or low-end 32-bit processors, and they have constrained RAM and storage capacity, with RAM ranging from a few kilobytes (KB) to megabytes(MB), and storage ranging from megabytes(MB) to a few gigabytes(GB) [28], [22].

The proposed methods are based on a list and a graph respectively. Unlike most available literature, where the prediction is based on "time", these models predict the future state using the states of the system. These methods work for both single and multiple-looped paths the adversary robot takes. The experiments were conducted in a simulated environment where an adversary robot moves in a path with and without self-loops. The results show that the prediction error which depends on the predicted movement and the actual move taken by the robot decreases as a logarithmic function. The experiment also shows that both methods could effectively analyze the pattern in the movement. These methods show that error convergence occurs faster in maps with multiple loops than maps with single loops.

The free space regions of a map, where adversary robots are moving, need to be monitored continuously using cameras. As these proposed systems require deployment in places devoid of internet connection and GPS, there can be no centralized system to monitor the movements of the adversary robots holistically. Thus, these robots have to be observed in a local mode. The movement of a single adversary robot is the "behaviour" itself, which is observed through the "vision circles" and its movement is predicted. Thus, it is necessary that the "vision circles", which can be cameras having 360^0/panoramic view with a bounded vision be placed strategically such that the "behaviour" and the movements predicted of the adversary robots be observed efficiently. Instead of being a completely open space, a territory can have obstacles, already present within its boundary. In that case, cameras must be placed only in open spaces as there is no point in watching spaces occupied by obstacles. In a practical scenario, the open space to be covered is much larger compared to the area that can be covered by an individual circle. This gives rise to the problem of the strategic placement of the circles inside the free space. This work proposes methods to find strategic locations where these cameras can be placed to provide maximum coverage of the free space. It is also shown in this work that the problem

related to the placement of circles in the free space of a map with convex and/or concave polygons while ensuring minimum intersection among existing circles and these polygons is an NP-complete problem.

The algorithm proposes strategic placement of vision circles based on planar subdivision [7], where vertical slabs maintain sets of free and occupied trapezoids from the map. It employs an objective function considering centrality scores of free trapezoids and the area of enclosing rectangular spaces to identify the best-suited rectangle for vision circle placement. Centrality scores measure the importance of trapezoids based on their potential paths, derived from the graph representing the planar subdivision.

During strategic circle placement, overlap between polygons and circles is avoided. Successive circles are positioned in the remaining free map space to prevent redundant coverage. The circle, original polygons, and intersecting trapezoids are combined into a new polygon, updating the map. Trapezoids from the planar subdivision are examined during circle merging with other polygons. If polygon edges extend beyond the circle, external vertices are added to the merged polygon. Existing polygons' concavity may create holes, which are marked within the final merged polygon.

The most common way to find the free spaces among polygons is to divide the surface into smaller subdivisions and then find the unoccupied spaces. The subdivision of the space can be done in various ways e.g. dividing the space into equally/unequally spaced grids or polygonal space e.g. trapezoids. LaValle(2006) [38] demonstrated how the free space of a map can be decomposed into vertical cells, where each of the cells can be either a trapezoid that has vertical sides or a triangle (which is a degenerate trapezoid). Other litterateurs decompose the map into square planar regions/grids of different sizes or triangles. However, all of these litterateurs do not detect free space when polygons are deleted from the map. There are some popular data structures like the "slab structure" and "trapezoidal map" [7] which have been used to find the free space, but these data structures are not suitable when polygons are deleted. In this work, a simple data structure has been proposed that is adapted from the "slab structure" for detecting the free space in a map due to the insertion/deletion of concave/convex polygons in a given bounded rectangular space.

This algorithm represents each vertical slab as a specially designed tree containing trapezoids. These trees can be easily updated or merged with other trees while

inserting or deleting a polygon. Free and occupied trapezoids can be easily detected while traversing these trees for each vertical slab. It has been shown that for specific arrangements of concave polygons, the expected time complexity for updating the relevant trees is $O(NH)$ where N is the number of existing polygons and H is the number of concave angles of the polygon which is inserted or deleted.

The rest of the thesis is organized as follows: chapter 2 discusses free space detection in a dynamic environment where convex and concave polygons are added on demand, chapter 3 describes the placement of vision circles in free space and updating the environment to avoid redundant coverage of same free space by more than one vision circles, chapter 4 talks about a method for predicting the future behaviour of adversary robots/agents and chapter 5 concludes the research work.

Chapter 2

Free Space Detection in a Dynamic Environment with Polygonal Obstacles

A common necessity in the realm of robot motion planning involves identifying the unobstructed areas within a given map containing polygonal obstacles. Such maps can either remain static, with no alterations to existing obstacles or additions of new ones, or dynamic, where obstacles are frequently added or removed. In static maps, once the free spaces are determined, they remain unchanged. However, in dynamic maps, these free spaces must be updated alongside map modifications. If map updates occur frequently, the algorithm responsible for updating the existing free space must be efficient in terms of time complexity. In this chapter, highly effective data structures are presented alongside a simple algorithm tailored to leverage it. Specifically designed for a closed rectangular surface undergoing frequent insertion and deletion of convex and concave polygonal voids, this algorithm addresses these dynamic conditions adeptly.

2.1 Literature Survey

The challenge of locating free space is relevant in the areas of autonomous car navigation, motion planning in robots, military applications, crowd simulation, VLSI, and gaming applications. Autonomous cars must navigate safely through their environment, avoiding collisions with obstacles such as other vehicles, pedestrians, and

stationary objects like buildings or road signs. Identifying free space allows the car to plan a path that avoids these obstacles. Determining the free space enables a robot to plan an optimal path from its current location to its destination. By considering only areas where the robot can safely travel, the robot can generate a route that minimizes travel time and maximizes safety. Military vehicles, such as tanks, aircraft, and drones, need to navigate through complex and potentially hostile environments while avoiding obstacles and minimizing exposure to threats. Free space calculation allows these vehicles to plan routes that maximize cover and concealment, reducing the risk of detection and attack. Free space calculation is a fundamental aspect of crowd simulation, enabling the modeling and analysis of pedestrian behavior in various environments. By accurately simulating crowd movement and interaction, these systems can inform decision-making processes related to safety, efficiency, and urban design. Free space calculation enables designers to optimize chip layout, ensure compliance with design rules, and achieve high performance, reliability, and manufacturability. In gaming contexts, the computation of free space is a foundational element essential for facilitating player movement, guiding enemy pathfinding, detecting collisions among game objects, crafting captivating and demanding levels, and dynamically altering environments in response to player actions.

Free space can have multiple applications in road map-based motion planning for autonomous robots as shown by Li et. al.(2020) [41] using a bee colony heuristic. In this context, Hoang et al. (2015) [29] used shortest path planning strategy to plan motions for robots. They both have used a combination of Dikjstra's algorithm along with a greedy breadth-first search technique to evaluate the shortest path.

Pradhan et al. (2021) [54] combined the visibility graph algorithm and RRT algorithm for the path planning of a biped robot in a static environment. They used both normal and multi-node RRT approaches for searching the path and then optimized the path using a suitable visibility graph algorithm. Teli, Wani [64] proposed a path-planning algorithm based on a fuzzy-based Artificial potential field(APF).

In VLSI, the placement of rectangular modules for reconfigurable computing devices such that connecting with the present modules requires minimum cost requires one to find free space where the modules can be connected to as shown by Ahmadinia (2007) [2].

The most common way to find free space is to divide the surface into smaller subdivisions and find unoccupied spaces. The subdivision of the space can be done in various ways e.g. dividing the space into equally/unequally spaced grids or polyg-

onal space e.g. trapezoids [7]. The subdivision of the polygonal environment has been used to locate points.

LaValle(2006) [38] demonstrated how the free space of a map can be decomposed into vertical cells, where each of the cells can be either a trapezoid that has vertical sides or a triangle (which is a degenerate trapezoid). The free space of the map can also be decomposed into triangles as shown by Boissonnat et al.(1998) [10]. A map with polygonal obstacles can also be recursively decomposed among square planar regions/grids of different sizes until there is a single line segment per grid as shown in Samet et al.(1985) [58].

After completing the computation of free space, these areas are additionally utilized for pinpointing locations within the map. Point location, a extensively studied subject in computational geometry, is typically addressed in existing literature by positioning points around a planar straight line graph, as exemplified by Kirkpatrick(1983) [35], Lee et al.(1979) [39]. Berg et al.(2008) [7] introduced a randomized incremental trapezoidal map algorithm capable of both identifying free spaces and pinpointing points.

The algorithm introduced by Berg et al. (2008) [7] is amenable to modification, allowing for the detection of free space within a given map even after polygons are added in the form of a sequence of line segments. In this alternate approach, subsequent to constructing the trapezoidal map, the trapezoids situated inside the polygons are eliminated to reveal the trapezoidal map corresponding to the free space. The incremental algorithm discerns the affected trapezoids as line segments are inserted individually. Newly intersected trapezoids are computed and substituted with freshly constructed ones, considering both the intersecting trapezoids and the line segments. Nonetheless, Berg et al. do not explicitly provide a methodology for updating the trapezoidal map data structure and computing free space upon deletion of a line segment.

Existing literature on the detection of free space among polygons lacks exploration into detecting free space after polygons are removed. This chapter introduces simple data structures that draw from the ”slab structure” method outlined by Berg et al. (2008) [7]. Additionally, an algorithm is presented that utilizes these data structures to determine free space within a map following the insertion or deletion of concave/convex polygons within a bounded rectangular region. Furthermore, these adaptable data structures facilitate point location within the map.

The remaining sections of this chapter have been organized in the following way: Section 2.2 proposes the data structures for organizing the concave/convex polygons and an offline algorithm using the same data structures for detecting the free space; Section 2.3 uses the same data structures to update the free space in the map when polygons are inserted or deleted; Section 2.4 analyses the complexity of the algorithm for an arrangement of convex and concave polygons; Section 2.5 reviews and analyses the proposed algorithm with some similar state-of-the-art algorithms.

2.2 Offline Algorithm for detecting Free Space in a Map

In this section, an offline algorithm is proposed that constructs a Planar Subdivision Map (PSM) for a given set of convex and/or concave polygons enclosed in a large axis-parallel rectangle box. It is assumed that there are K number of polygons indexed from 0 to $K-1$. A vertex of a polygon is identified as $P_{pi} = (P_{pi}^x, P_{pi}^y)$, where $p \in \{0, \ldots, K-1\}$ is the polygon index and i is the vertex index of the pth polygon.

A PSM, Γ can be represented as groups of trapezoids, T, where each group comprises the partitioned planar subdivision of a vertical slab obtained by extending vertical lines through all the vertices. The PSM is the free space around the polygons inside the bounded box comprised of the vertically partitioned trapezoids. In each vertical slab, intersecting lines of the polygons act as the non-parallel edges of the trapezoids, and the left and right vertical lines are the other two sides of the trapezoids. For example, in Figure 2.1, the polygon line segments $< P_{13}P_{14} >, < P_{13}P_{12} >, < P_{12}P_{11} >, < P_{11}P_{16} >, < P_{21}P_{22} >, < P_{21}P_{24} >$ intersect the vertical slab $(c_2c_3d_3d_2)$, the trapezoids formed are $(c_2c_3k_1P_{13})$, $(P_{13}k_1P_{12})$, $(P_{13}P_{12}j_2)$, $(j_2P_{12}k_2j_3)$, $(j_3k_2k_3j_4)$, $(j_4k_3k_4j_5)$, $(j_5k_4d_3d_2)$.

It can be observed that inside each group, trapezoids are ordered vertically from top to bottom. Hence, Γ is $\{(c_0c_1d_1d_0)\}$; $\{(c_1c_2j_2P_{11})$, $(P_{11}j_2j_3)$, $(P_{11}j_3j_4P_{21})$, $(P_{21}j_4j_5)$, $(P_{21}j_5d_2d_1)\}$; $\{(c_2c_3k_1P_{13})$, $(P_{13}k_1P_{12})$, $(P_{13}P_{12}j_2)$, $(j_2P_{12}k_2j_3)$, $(j_3k_2k_3j_4)$, $(j_4k_3k_4j_5)$, $(j_5k_4d_3d_2)\}$; $\{(c_3c_4P_{14}k_1)$, $(k_1P_{14}l_1k_2)$, $(k_2l_1P_{22}k_3)$, $(k_3P_{22}P_{24}k_4)$, $(k_4P_{24}d_4d_3)\}$; $\{(c_4c_5m_1P_{14})$, $(P_{14}m_1P_{16}l_1)$, $(l_1P_{16}m_2P_{22})$, $(P_{22}m_2m_3P_{24})$, $(P_{24}m_3d_5d_4)\}$; $\{(c_5c_6P_{15}m_1)$, $(m_1P_{15}P_{16})$, $(P_{16}P_{15}n_1m_2)$, $(m_2n_1n_2m_3)$, $(m_3n_2d_6d_5)\}$; $\{(c_6c_7P_{23}n_1)$, $(n_1P_{23}n_2)$, $(n_2P_{23}d_7d_6)\}$; $\{(c_7c_8d_8d_7)\}$.

In the following subsections, the data structures and methods for constructing the PSM, (Γ) are discussed.

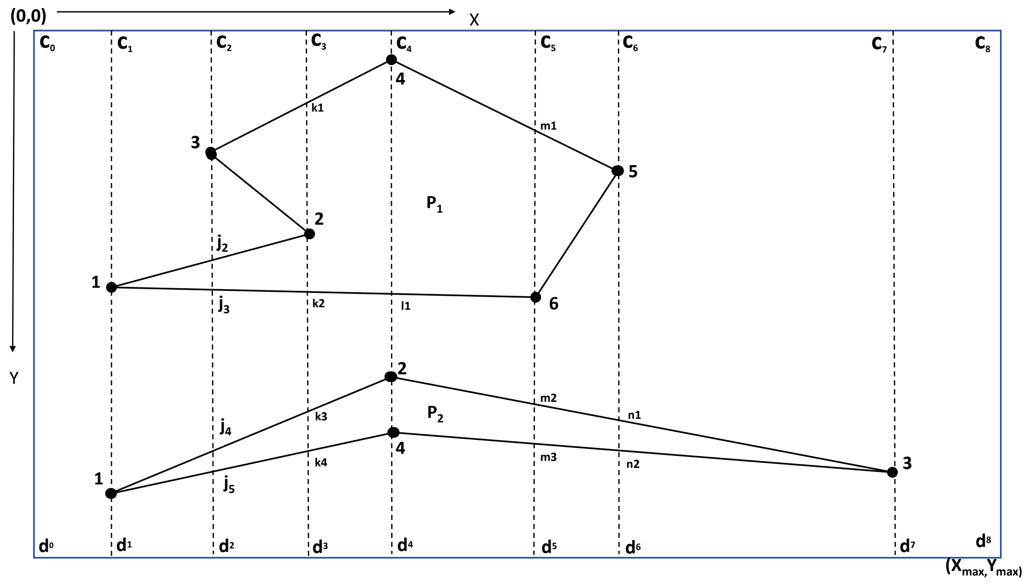


Figure 2.1: Example of a Planar Subdivision

The Γ structure comprises lists, with each list representing partitioned vertical planar subdivisions. These subdivisions are organized hierarchically under a root, which corresponds to the unpartitioned plane intersected by vertical lines passing through the vertices.

In the following section, a data structure is described that is used for maintaining the PSM. This data structure will be further used for detecting free spaces when polygons are inserted or deleted in the map.

2.2.1 The Initial Root Trapezoids (IRT)

Each vertical planar subdivision is depicted as the root node of a tree, with all child nodes representing subdivisions formed by polygon edges intersecting the vertical planar subdivision. This root node is known as the Initial Root Trapezoid, characterized by vertical edges aligned with the vertical straight lines drawn through polygon vertices, while its horizontal edges comprise sections of the upper and lower boundaries of the rectangular map.

At first, the polygon vertices are lexicographically sorted in an ascending order w.r.t. their x-coordinate. All vertices having the same x coordinate are kept inside the same group. For a group G_i, where i is the group index, the vertex having the minimum y coordinate value be called the *Group Leader* and represented by $GL(G_i)$. A "Vertex-Group Leader" hash table, Λ maintains the mapping of a polygon vertex

and its corresponding group leader. In Figure 2.1, the groups are : G_1={P_{11},P_{21}}, G_2={P_{13}}, G_3={P_{12}}, G_4={P_{14},P_{22}, P_{24}},G_5={P_{16}}, G_6={P_{15}}, G_7={P_{23}}.

Vertical lines through each $GL(G_i)$ generate ordered left to right vertical slabs, which are referred to as the *Initial Root Trapezoids*(IRT). The first group corresponds to the second IRT, the second group corresponds to the third IRT, and so on. In Figure 2.1, the seven(7) IRTs are $T_1 = (c_1c_2d_2d_1)$, ..., $T_7 = (c_7c_8d_8d_7)$, each corresponding to a group G_1, G_2, G_3, ..., G_7. The IRT $T_0 = (c_0c_1d_1d_0)$ is not accounted for as it does not have any group. The edges of the polygons might intersect these vertical lines and create smaller trapezoids inside each IRT.

In the upcoming subsection, a hash table-based data structure is developed to facilitate the identification of how polygonal line segments are used to divide the Initial Root Trapezoids (IRTs) into smaller trapezoids. This data structure, initially named the "Line Segment Hash Table," will be simplified and referred to as the "Hash Table" for ease of understanding.

2.2.2 Design of the Hash Table

This data structure contains information concerning polygon edges or their segments that intersect the vertical lines of Initial Root Trapezoids (IRTs). All trees associated with IRTs will be generated from this structure. Represented as a sorted hash table denoted by Ω, it manages key elements $GL(G_i)$ and segments of polygon edges intersecting corresponding IRTs horizontally, arranged vertically from top to bottom. Each IRT will have a single entry in this hash table. Once constructed, all IRTs can be easily created by inspecting entries in this hash table.

For any group, G_i, each vertex corresponds to two line segments (as two polygon line segments are incident on each vertex) which intersect the IRT of $GL(G_i)$. A polygon line segment starts at a source vertex ($SVtx$) on a Left Vertical Line (LVL) of an IRT and ends at a target vertex ($TVtx$) on a Right Vertical Line (RVL) of the same/different IRT. The line segments $< SVtx, TVtx >$ are maintained as a list for each $GL(G_i)$, sorted in a lexicographical order w.r.t. $SVtx_y$ and $TVtx_y$. a

Smaller trapezoids are formed by the polygon line segments when they intersect the IRTs. These lines are again sorted in an ascending order w.r.t. the y-coordinate of $SVtx$. The first line segment of the sorted list partitions an IRT into an upper and a lower trapezoid; the second line segment partitions the latter lower trapezoid into

(1) $P_{11} \rightarrow$	P_{11}	P_{12}	TRUE	$Poly_1$
	P_{11}	P_{16}	TRUE	$Poly_1$
	P_{21}	P_{22}	TRUE	$Poly_2$
	P_{21}	P_{24}	TRUE	$Poly_2$
(2) $P_{13} \rightarrow$	P_{13}	P_{14}	TRUE	$Poly_1$
	P_{13}	P_{12}	TRUE	$Poly_1$
(3) $P_{12} \rightarrow$	P_{12}	P_{13}	FALSE	$Poly_1$
	P_{12}	P_{11}	FALSE	$Poly_1$
(4) $P_{14} \rightarrow$	P_{14}	P_{13}	FALSE	$Poly_1$
	P_{14}	P_{15}	TRUE	$Poly_1$
	P_{22}	P_{21}	FALSE	$Poly_2$
	P_{22}	P_{23}	TRUE	$Poly_2$
	P_{24}	P_{21}	FALSE	$Poly_2$
	P_{24}	P_{23}	TRUE	$Poly_2$
(5) $P_{16} \rightarrow$	P_{16}	P_{11}	FALSE	$Poly_1$
	P_{16}	P_{15}	TRUE	$Poly_1$
(6) $P_{15} \rightarrow$	P_{15}	P_{14}	FALSE	$Poly_1$
	P_{15}	P_{16}	FALSE	$Poly_1$
(7) $P_{23} \rightarrow$	P_{23}	P_{22}	FALSE	$Poly_2$
	P_{23}	P_{24}	FALSE	$Poly_2$

Table 2.1: Sorted Line Segments Hash Table : Ω

two smaller trapezoids and subsequently, the other line segments partition the previously obtained lower trapezoid into two smaller trapezoids. Thus, all the trapezoids of an IRT can be obtained from top to bottom using the sorted list of line segments w.r.t. a $GL(G_i)$.

A marker, $isProcReq = TRUE$ for all unique pairs and $isProcReq = FALSE$ for duplicate pairs are maintained. For each pair, $< SVtx, TVtx >$, a $PolyId$ is also maintained indicating the affiliated polygon. $LnSegLst$ is a list containing the tuples $< SVtx, TVtx, isProcReq, PolyId >$.

Table 2.1 gives the order of the keys of Ω and their corresponding list of line segments (as per Fig. 2.1) due to the intersection of the vertical lines LVL, RVL and $< SVtx, TVtx >$ pairs.

In the next section, trees are constructed for all the IRTs using Ω due to the intersection of the vertical lines LVL, RVL, and $< SVtx, TVtx >$ pairs. These trees are used for maintaining and updating trapezoids corresponding to the PSM.

2.2.3 Construct Trees With New Trapezoids

In this section, a method is proposed for the identification of trapezoids within vertical slabs and their arrangement as trees under IRTs using Ω. The outcome comprises

a list of IRTs, with each IRT serving as the root of a binary tree for its respective vertical slab, and all the partitioned smaller trapezoids are nodes within that tree.

Each trapezoid can be identified by four vertices where each vertex has information related to its

1 current polygon and

2 a marker representing whether it is an actual polygon vertex or an intersection point with the vertical lines

Each trapezoid, T is represented with

1 T_{tl}: Top left vertex of trapezoid

2 T_{tr}: Top right vertex of trapezoid

3 T_{bl}: Bottom left vertex of trapezoid

4 T_{br}: Bottom right vertex of trapezoid

5 LCT: Reference To Left Child Trapezoid

6 RCT: Reference To Right Child Trapezoid

7 AdT: Adjacent Right IRT

8 $ParentT$: Parent Trapezoid

9 $TEId$: Polygon Identifier of the top edge; for the Root Node, it will be marked as $-$ for $NULL$

The AdT represents the adjacent right IRT due to the creation of the vertical lines; the extreme right AdT will be $-$, as there exists no IRT right of it. Figure 2.1 represents the initial IRTs constructed by the vertical lines through each vertex and attribute values for some of the IRTs are

(1) $T_0 : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [c_0, c_1, d_0, d_1, -, -, T_1, -, -]$

(2) $T_1 : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [c_1, c_2, d_1, d_2, -, -, T_2, -, -]$

(3) $T_7 : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [c_7, c_8, d_7, d_8, -, -, -, -, -]$

In the beginning, the IRTs are not partitioned, the LCT and the RCT are kept as $-$.

For each group, G_i, the corresponding IRT is partitioned by each line segment $< SVtx, TVtx >$ of the list for $GL(G_i)$ in Ω. Each line segment divides the current right-most node T_d (or the IRT at the beginning) of the tree for G_i into an upper and a lower trapezoid. The upper and lower trapezoids are added as the LCT and RCT of T_d respectively in the tree. If any line segment $< SVtx, TVtx >$ crosses over to the RVL of current IRT, then a "Derived segment" $< SVtx_d, TVtx_d >$ is added to $GL(G_{i+1}) \in \Omega$; $SVtx_d$ and $TVtx_d$ are the intersections of the line $< SVtx, TVtx >$ with LVL and RVL of the corresponding IRT of G_{i+1}.

In Table 2.1, the group leader of the first group in Ω is P_{11} and its first pair is $< P_{11}, P_{12} >$. The group leader for the next group is P_{13}. As $(P_{12})_x > (P_{13})_x$, line segment $< P_{11}, P_{12} >$ intersects the vertical line drawn through P_{13}. Thus, it creates a new trapezoid in the next IRT, T_2. The intersection point j_2 creates a "Derived segment", $< j_2, P_{12} >$, which is added to the next group P_{13} retaining the sorted order.

Thus, the table represented by key P_{13} in Ω is $P_{13} \rightarrow$

P_{13}	P_{14}	TRUE
P_{13}	P_{12}	TRUE
j_2	P_{12}	TRUE

The line segment $< P_{11}, P_{12} >$ divides T_1 into two new trapezoids, $T_{1.1}$ and $T_{1.2}$. Thus, $T_{1.1}$ becomes the LCT and $T_{1.2}$ is the RCT of T_1. Thus, the updated Ω is

(1) $T_1 : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [c_1, c_2, d_1, d_2, T_{1.1}, T_{1.2}, T_2, -, -]$

(2) $T_{1.1} : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [c_1, c_2, P_{11}, j_2, -, -, T_2, T_1, -]$

(3) $T_{1.2} : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [P_{11}, j_2, d_1, d_2, -, -, T_2, T_1, Poly_1]$

The next pair in the group P_{11} is $< P_{11}, P_{16} >$, which divides $T_{1.2}$. Thus, $T_d = T_{1.2}$. As $(P_{16})^x > (P_{13})^x$, $< P_{11}, P_{16} >$ intersects the vertical line drawn through P_{13}. The intersection point j_3 creates a "Derived segment", $< j_3, P_{16} >$, which is added to the next group P_{13} retaining its sorted order. This "Derived Line" increases the level of the tree corresponding to the next slab, T_2 by adding new LCT and RCT. The updated group P_{13} becomes $P_{13} \rightarrow$

P_{13}	P_{14}	True
P_{13}	P_{12}	True
j_2	P_{12}	True
j_3	P_{16}	True

$< P_{11}, P_{16} >$ divides the trapezoid $T_{1.2}$ into two new trapezoids, $T_{1.2.1}$ and $T_{1.2.2}$. $T_{1.2.1}$ is the LCT and $T_{1.2.2}$ is the RCT of $T_{1.2}$. Thus, the updated trapezoid $T_{1.2}$ with its LCT and RCT is

(1) $T_{1.2} : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [P_{11}, j_2, d_1, d_2, T_{1.2.1}, T_{1.2.2}, T_2, T_1, Poly_1]$

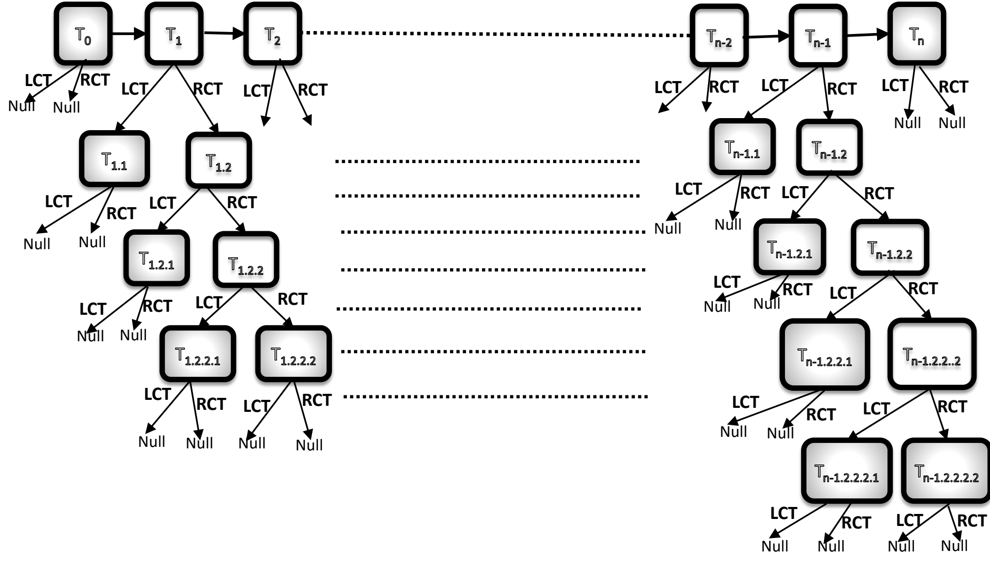


Figure 2.2: Initial Trapezoids and Generated Trees

(2) $T_{1.2.1} : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [P_{11}, j_2, P_{11}, j_3, -, -, T_2, T_{1.2}, Poly_1]$

(3) $T_{1.2.2} : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [P_{11}, j_3, d_1, d_2, -, -, T_2, T_{1.2}, Poly_1]$

After all the pairs have been processed sequentially for a $GL(G_i)$, it is to be noted that the smaller trapezoids are created inside their corresponding IRT. In this process, a tree is created for every IRT and all its leaf trapezoids correspond to the smaller trapezoids of the IRT. Leaf trapezoids are either at the left side of each intermediate level or at the lowest level of the tree. For T_0, the tree consists of only one root node i.e. T_0 itself. Figure 3.13 shows the generation of the trees from all the IRT nodes. It is noted that the first and last IRTs do not have any LCT or RCT figures-planar-sub/and the grey trapezoids represent the PSM of Figure 2.1.

For a given set of polygons $Poly_1$, $Poly_2$, $Poly_3$, ..., $Poly_N$, where $Poly_i = (P_{i1}, P_{i2}, \ldots, P_{im})$, [Algorithm 1] constructs the PSM Γ. Using the list of polygons, IRTs Π, Groups G (Section 2.2.1) and Sorted Hash Table Ω (Section 2.2.2) are constructed respectively. These are then used by [Algorithm 2] for constructing trees for all the trapezoids (Section 2.2.3).

The nodes in the tree represent a trapezoid that is to be partitioned and the leaves are the trapezoids that cannot be further partitioned. The collected leaves starting from the left child of the IRT to the bottom left and right leaves respectively represent

the PSM for the corresponding group/vertical slab [Algorithm 1: Line 9]. The odd-numbered trapezoids in this collection are the unoccupied or free space of the PSM for the group, while the even-numbered trapezoid lies inside the polygons.

Algorithm 1 PSM For Polygonal Obstacles

Require: List of Polygons: $Poly_1$, $Poly_2$, $Poly_3$, ..., $Poly_N$, where $Poly_i = (P_{i1}, P_{i2}, \ldots, P_{im})$
Ensure: PSM Γ
1: Generate IRT Π : $(T_0, \ldots, T_n)$; Groups : $G = (G_1, \ldots G_n)$; Vertex-Group Leader Hash Table : Λ
2: Construct Sorted Hash Table : Ω from polygon vertices and edges
3: Construct Vertex-Group Leader Hash Table : Λ;
4: Mark $isProcReq = FALSE$ for Ω using Vertex-Group Leader Hash Table : Λ
5: $\forall$ duplicate $< SVtx, TVtx >$: $isProcReq = FALSE$
6: `ConsructTreesWithNewTrapezoids`(Π, G, Ω)
7: **for** i = 1 *to* ($|IRT|$) - 1) **do**
8: Trapezoid Group : $TG = \phi$
9: `ConsructPSMGroup`(T_i, TG)
10: Append TG to Γ.
11: **end for**

Algorithm 2 ConstructTreesWithNewTrapezoids

Require: List of IRT : Π : $(T_0, T_1, \ldots, T_n)$; Groups of Vertices : $G = (G_1, G_2, \ldots, G_n)$; Sorted Hash Table : Ω
Ensure: IRTs with Trees : Π : $(T_0, T_1, \ldots, T_n)$
for all $G_i \in G$ **do**
2: $Key = GL(G_i)$
$\triangleright$ $LnSegLst$ is a list of tuples, $< SVtx, TVtx, isProcReq >$ from Ω based on Key.
$LnSegLst = CollectLineSegments(\Omega, Key)$
4: $T_d = T_i$
for $tuple \in LnSegLst$ such that $isProcReq \equiv True$ **do**
6: $DerivedLine = \phi$
if $(TVtx)^x > (GL(G_{i+1}))^x$ **then**
8: $SVtx'$ = Intersection of vertical line through $(GL(G_{i+1}))$ and line segment $< SVtx, TVtx >$
$DerivedLine : [SVtx_d, TVtx_d, isProcReq] = [SVtx', TVtx, True]$
10: $Key' = GL(G_{i+1})$
$Insert\Omega(DerivedLine, Key')$
12: $Sort\Omega(GL(G_{i+1}))$
end if
14: $T_d.LCT : [T_{tl}, T_{tr}, T_{bl}, LCT, RCT, AdT, ParentT, TEId] = [T_d.T_{tl}, T_d.T_{tr}, SVtx, -, -, T_{i+1}, T_d, T_d.TEId]$
if $DerivedLine \neq \phi$ **then**
16: $T_d.LCT.T_{br} = SVtx'$
else
18: $T_d.LCT.T_{br} = TVtx$
end if
20: $T_d.RCT : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, AdT, ParentT, TEId] = [T_d.LCT.T_{bl}, T_d.LCT.T_{br}, T_d.T_{bl}, T_d.T_{br}, -, -, T_{i+1}, T_d, tuple.PolyId]$
$T_d = T_d.RCT$
22: **end for**
end for

2.3 Dynamic Algorithm for Detecting Free Space due to the Insertion/Deletion of Polygons

In this section, taking advantage of the data structures in the previous section, a dynamic algorithm is proposed that updates the IRTs due to the insertion or deletion of a polygon in the map. In this algorithm, it is assumed that there exists a PSM and subsequently its IRT for some polygons in a map.

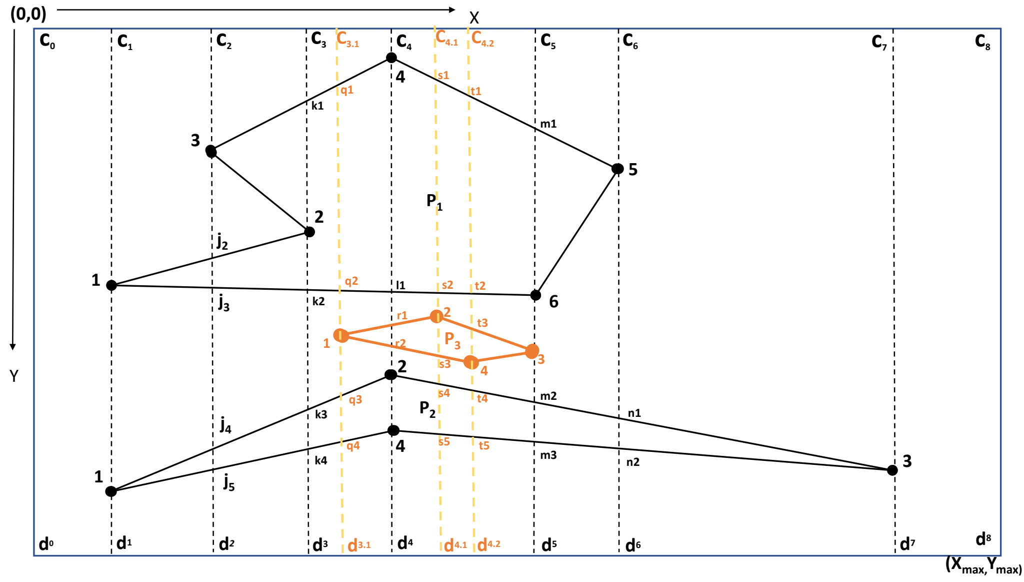


Figure 2.3: New Polygon P_3 inserted In Environment

2.3.1 Insertion of a Polygon in a Map

In this section, a method is discussed that detects free space due to the insertion of a polygon, $Poly_i(P_{i1}, \ldots, P_{ij}, \ldots, P_{in})$ into the map. Due to this insertion, the existing vertical slabs (IRTs and their associate trees) are split and their corresponding data structures are updated.

This method has two parts: the first part where the IRTs are split along with their corresponding trees and the second part where the trees are updated.

Splitting of the IRTs and their Associated Trees

The vertices of the new polygon are sorted in an ascending order w.r.t. their x-coordinate. The IRT in which P^x_{ij} lies is identified. This IRT is partitioned into two IRTs representing a line passing through P^x_{ij}. This is done by replacing the present IRT with a pair of duplicates renamed as IRT_1 and IRT_2. The vertices $\{T_{tr}, T_{br}\}$ and $\{T_{tl}, T_{bl}\}$ for each node of IRT_1 and IRT_2 respectively are updated with the intersection of the vertical line and the node's top and bottom edges.

In Figure 2.3, due to the insertion of P_{31} of $Poly_3$, the intersection between the vertical line passing through vertex P_{31} and top, bottom edges of $IRT : [c_3, c_4, d_4, d_3]$ gives the $\{T_{tr}, T_{br}\}$ and $\{T_{tl}, T_{bl}\}$ of IRT_1 and IRT_2 respectively. Thus, the updated IRTs are $IRT_1 : [c_3, c_{3.1}, d_{3.1}, d_3]$ and $IRT_2 : [c_{3.1}, c_4, d_4, d_{3.1}]$. Similarly, the intersection between the vertical line passing through vertex P_{31} and top, bottom edges of The

trapezoid $[k_2, l_1, P_{22}, k_3]$ gives the updated $\{T_{tr}, T_{br}\}$ and $\{T_{tl}, T_{bl}\}$ of the trapezoid node $[k_2, l_1, P_{22}, k_3]$ in the tree of IRT_1 and IRT_2 respectively. Thus the trapezoid node $[k_2, l_1, P_{22}, k_3]$ in the tree of IRT_1 is updated to $[k_2, q_2, q_3, k_3]$ and in the tree of IRT_2, the same trapezoid node is updated to $[q_2, l_1, P_{22}, q_3]$.

Updating the Trapezoid Trees

Algorithm 3 Update the Trapezoid Trees Due To Insertion of Polygon

Require: New Polygon: $Poly_i \in \{Poly_1, Poly_2, \ldots, Poly_N\}$, where $Poly_i = (P_{i1}, P_{i2}, \ldots, P_{im})$; IRT: $\Pi : (T_0, T_1, \ldots, T_n)$, $G = G_0, G_1, \ldots, G_n$; Λ; Ω
Ensure: Updated Π'; G; Λ; Ω
    Update Groups : G with vertices of $Poly_i$
    Update: Λ for vertices of $Poly_i$ ▷ Λ : Vertex-Group Leader
3: Update: Ω for vertices of $Poly_i$ ▷ Ω : Sorted line segment Hash Table
    for all Duplicate $< SVtx, TVtx >$ || $(SVtx \notin Poly_i) of \Lambda$ **do**
        $isProcReq = FALSE$
6: **end for**
    for all $G_i \in G$ **do**
        $Key = GL(G_i)$
9:     $I = IRT\ T such\ that\ LVL_x = Key_x$
        $LnSegLst = CollectLineSegments(\Omega, Key)$ ▷ $LnSegLst$ is a list of tuples $< SVtx, TVtx, isProcReq > \in \Omega$ based on Key
        $IHN = -$ ▷ Insertion Hit Node
12:     **for** $< SVtx, TVtx > \in LnSegLst$ such that $isProcReq = True$ **do**
            if $(TVtx)_x > (GL(G_{i+1}))_x$ **then**
                $SVtx'$ = Intersection of vertical line from $(GL(G_{i+1}))$ and line segment $< SVtx, TVtx >$
15:             $DerivedLine : [SVtx_d, TVtx_d, isProcessingRequired] = [SVtx', TVtx, True]$
                $Key' = GL(G_{i+1})$
                $Insert\Omega(DerivedLine, Key')$
18:             $Sort\Omega(GL(G_{i+1}))$
            end if
            if $IHN \equiv -$ **then**
21:             Find the leaf Node, $T_r \in I | (T_{tl})_y < SVtx_y < (T_{bl})_y)$
                $IHN = T_r$
            end if
24:         If IHN is found at the lowest level: Add the LCT and RCT of this IHN.
            If IHN is found in an intermediate level: Update the IHN and its position, parent and sibling as described in Section 2.3.1.
            Set $isProcReq = True$ for current line segment
27:     **end for**
    end for

In section 2.3.1, the IRTs are vertically split due to the insertion of the new polygon, $Poly_i$. In this section, nodes in the trees of IRTs are further split due to the insertion of edges of the new polygon.

The vertices of $Poly_i$ along with existing polygons are categorized into groups: $G_0, G_1, \ldots, G_k$ based on their x-coordinates.

As in the offline algorithm, Λ and Ω are maintained and as the new polygon is introduced, both are updated keeping their characteristic features. For each unique line segment, $< SVTx, TVTx > \in LnSegLst$ belonging to $Poly_i$, the tree associated with its corresponding IRT is updated by first identifying the leaf trapezoids. These leaf trapezoids are searched in the tree associated with a group for $< SVTx, TVTx >$

by traversing the leaf trapezoid T_r for which $(T_{tl})^y < (Poly_i.SVTx)^y < (T_{bl})^y$. These identified leaf trapezoids are each referred to as an "Insertion Hit Node"(IHN).

If the new polygon is inserted into the bottom of the map, the IHN will be at the lowest level of the tree of the corresponding IRT. The new left and right child nodes will be added directly under this IHN. Attributes of the left and right child of IHN are updated as follows :

(1) $IHN.LCT : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, ParentT, AdT, TEId] =$
$[IHN.T_{tl}, IHN.T_{tr}, SVtx, TVtx, -, -, IHN,$ Adjacent $IRT, -]$

(2) $IHN.RCT : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, ParentT, AdT, TEId] =$
$[SVtx, TVtx, IHN.T_{bl}, IHN.T_{br}, -, -, IHN, IRT', PolyId']$

where IRT' is the Adjacent IRT; $PolyId'$ corresponding to $< SVtx, TVtx >$

If the new polygon is inserted at the top of all existing polygons or in between two existing polygons, then the IHN lies at the intermediate levels. If the IHN is found at an intermediate level, then the right sibling of IHN is brought down to the next level and a new node takes its place. A new left child node is also added for this new node. The IHN along with the new node and its left child nodes are updated accordingly. The update to the IHN and its position, parent, and sibling is done as follows:

(1) $IHN : [T_{bl}, T_{br}] = [SVtx, TVtx]$

(2) $S = IHN.ParentT.RCT$

(3) $N : [T_{tl}, T_{tr}, T_{bl}, T_{br}, ParentT, AdT, TEId] =$
$[SVtx, TVtx, S.T_{bl}, S.T_{br}, IHN.ParentT, IRT', PolyId']$

where IRT' is the Adjacent IRT; $PolyId'$ corresponding to $< SVtx, TVtx >$

(4) $IHN.ParentT.RCT = N$

(5) $N.RCT = S$

(6) $LN : [T_{tl}, T_{tr}, T_{bl}, T_{br}, LCT, RCT, ParentT, AdT, TEId] =$
$[SVtx, TVtx, S.T_{tl}, S.T_{tr}, -, -, N,$ Adjacent $IRT', PolyId']$

where IRT' is the Adjacent IRT; $PolyId'$ corresponding to $< SVtx, TVtx >$

(7) $N.LCT = LN$

(8) $IHN = LN$

For the same vertical slab, this new left child node acts as the new IHN for the next tuple in the $LnSegLst$.

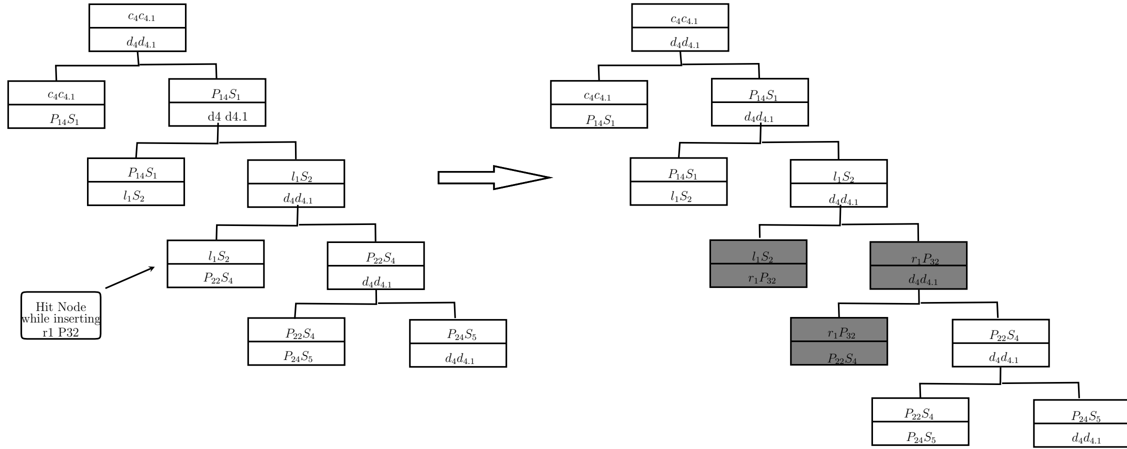


Figure 2.4: Update tree for r_1P_{32} insertion

In Figure 2.4, the left side shows the existing tree for the vertical slab $[c_4c_{4.1}d_4d_{4.1}]$. The right side shows the updated tree with three new nodes (grey colour) due to the insertion of $< r_1, P_{32} >$ in this tree. The IHN found for this line segment is $l_1S_2P_{22}S_4$. The current sibling node of this IHN is $[P_{22}S_4d_4d_{4.1}]$, which is pushed one level down the tree. The bottom edge of IHN is replaced with current line segment $< r_1, P_{32} >$. A new node $[r_1P_{32}d_4d_{4.1}]$ takes the place of displaced sibling node and $[r_1P_{32}P_{22}S_4]$ is added as its left child node. The node $[r_1P_{32}P_{22}S_4]$ is the new IHN for the next line segment $< r_2, s_3 >$ for the current group.

2.3.2 Deletion of a Polygon from the Map

In this section, a method is proposed for updating the data structure when a polygon is deleted from the existing map. To delete a polygon, the line segments of the polygon are first deleted from the vertical slabs, following which the vertical slabs are then merged.

Updating the Trapezoid Trees

The trapezoid trees for all vertical slabs will be updated [Algorithm 4] such that no nodes related to the edges of the deleted polygon are present in these trees.

The bottom-most line segment corresponding to the polygon which is to be deleted in a vertical slab is the bottom-most left child trapezoid in the tree for which $TEId$ is the same as the identifier of the polygon to be deleted. This left child trapezoid is referred to as the "Deletion Hit Node" (DHN). For a vertical slab, the line segments of the deleted polygon are deleted starting from the bottom to the top.

Algorithm 4 Update the Trapezoid Trees Due To Deletion Of Polygon

Require: Polygon Identifier to be deleted $TEId$; Polygon to be deleted: $Poly_i \in \{Poly_1, Poly_2, \ldots, Poly_N\}$, where $Poly_i = (P_{i1}, P_{i2}, \ldots, P_{im})$; IRT $\pi^{'}$: $(T_0^{'}, T_1^{'}, \ldots, T_j^{'})$; $G = G_0, G_1, \ldots, G_n$; Λ; Ω

Ensure: IRT with Trees : π : $(T_0^{'}, T_1^{'}, \ldots, T_j^{'})$; G; Λ ; Ω

```
    Delete groups in G corresponding to vertices of Poly_i
    Delete entries in Λ corresponding to vertices of Poly_i
    Delete key and corresponding LineSegment List in Ω for which key matches the vertices of Poly_i
 4: Delete entries in Ω for which < SVtx, TVtx > lies on an edge of Poly_i
    for IRT I ∈ π' do
        polygonEdgeExist = TRUE
        while (polygonEdgeExist) do
 8:         CurrNode = I.LCT; CurrLevel = 1; MaxLevel = CurrLevel
            polygonEdgeExist = FALSE
            repeat
                if CurrTrapezoid.TEId ≡ TEId then
12:                 DL = CurrLevel                          ▷ DL : Delete Hit Node level
                    DHN = CurrTrapezoid
                    CurrTrapezoid = CurrTrapezoid.LCT
                    CurrLevel = CurrLevel + 1
16:                 MaxLevel = CurrLevel
                    polygonEdgeExist = TRUE
                end if
            until CurrTrapezoid ≡ −
20:         if DL ≡ MaxLevel then
                PN = DHM.ParentT                            ▷ Parent Trapezoid
                PN.RCT = PN.LCT = −
            else
24:             RightSiblingTrapezoid = DHN.ParentT.RCT
                DHN.ParentT = RightSiblingTrapezoid
                DHN.ParentT.ParentT.LCT.T_bl = RightSiblingTrapezoid.T_tl
                DHN.ParentT.ParentT.RCT.T_br = RightSiblingTrapezoid.T_tr
28:         end if
        end while
    end for
```

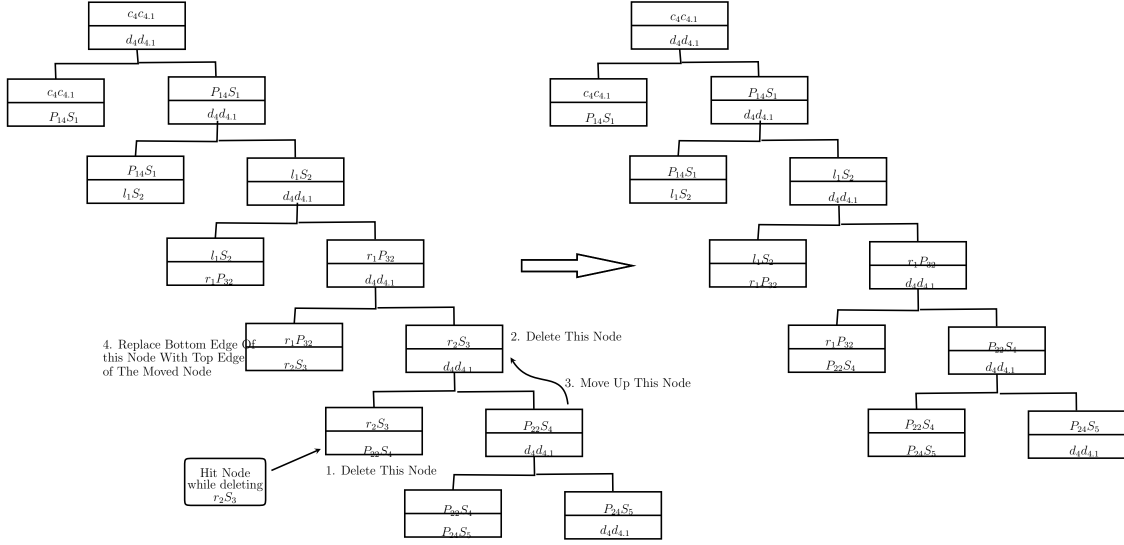


Figure 2.5: Update tree for r_2s_3 deletion

If the deleted polygon lies below all existing polygons, then the DHN will be at the lowest level of the tree, and the DHN and its sibling right child trapezoid nodes will be deleted. If the deleted polygon lies at the top of all polygons or in between polygons, then the DHN will be at the intermediate levels of the tree and, the DHN and its parent node are deleted. The former right sibling of DHN is moved up by one level and its current left child is updated accordingly, which becomes the next DHN for the current group.

Apart from updating the trees, groups in G, entries in Λ and Ω matching the vertices of the polygon are also deleted. Entries in Ω for which $< SVtx, TVtx >$ lies on an edge of $Poly_i$ are also deleted.

Figure 2.5 shows the existing tree for the vertical slab $[c_4c_{4.1}d_4d_{4.1}]$, DHN and the updated tree due to deletion of r_2s_3 corresponding to Figure 2.3.

Merging of IRTs and their Associated Trees

After the line segments of the deleted polygon are removed from the vertical slabs, these are merged with the neighbouring vertical slabs. Starting from the leftmost vertical slab, groups of vertical slabs that need to be merged are identified. A vertical slab is eligible for merging if no left child trapezoid is present in the corresponding tree for which the T_{tl} is an actual polygon vertex.

Two consecutive trees to be merged will have the same height because the vertical lines corresponding to the vertices of the deleted polygon had created these vertical

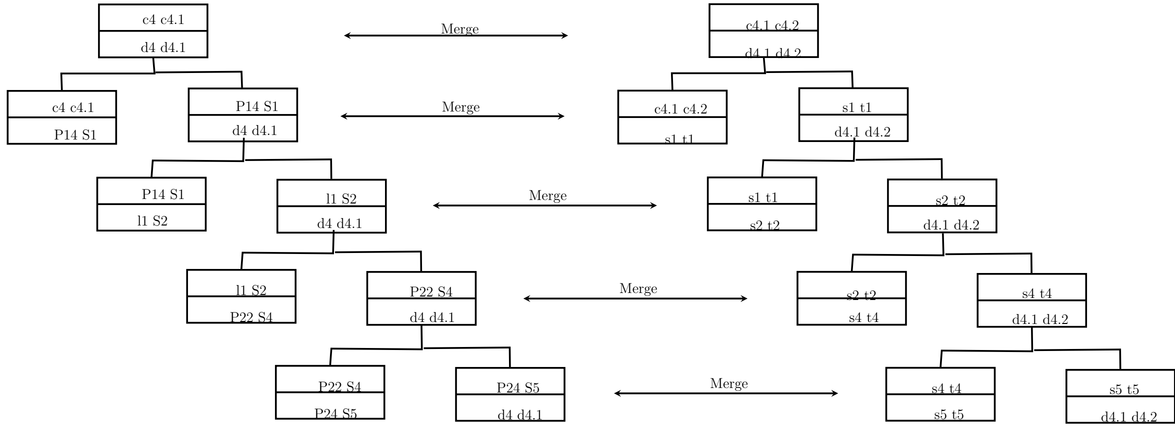


Figure 2.6: Merging of two consecutive trees after deletion of edge segments of the deleted polygon

slabs by dividing each trapezoid of the existing vertical slab into an equal number of trapezoids. Thus, the trapezoids which are in the same level and position are merged. When two consecutive trapezoids (T_1 and T_2) are merged, T_{tl} and T_{bl} of T_1 becomes T_{tl} and T_{bl}; T_{tr} and T_{br} of T_2 becomes T_{tr} and T_{br} of the merged trapezoid. LCT and RCT of the merged trapezoid are obtained by merging $T_1.LCT$, $T_2.LCT$ and $T_1.RCT$, $T_2.RCT$ respectively. The AdT of T_2 becomes the AdT of the merged trapezoid while $T_1.TEId$ is the $TEId$ of the merged trapezoid.

In example Figure 2.6, two existing trees for two consecutive vertical slabs $[c_4c_{4.1}d_4d_{4.1}]$ and $[c_{4.1}c_{4.2}d_{4.1}d_{4.2}]$ are shown. These two trees can be merged to create one tree for the merged vertical slab $[c_4c_{4.2}d_4d_{4.2}]$.

2.4 Analysis for the Dynamic Algorithm

The authors proposed a data structure by adapting the “slab structure” stated by Berg et.al.[7], as they observed that a deletion of a line segment or a polygon from the popular trapezoidal map data structure is demanding due to the combinatorial complexity of identifying and merging the associated trapezoids of the deleted line segments. In this section, the deletion process of a line segment in the trapezoidal map inspired by Berg et.al.[7] is analyzed first and then the analysis of the proposed system is conducted.

The proposed system for locating a point in a map by Berg et.al. consists of two data structures: “search structure” and “trapezoidal map”. When a new line segment is inserted, the leftmost trapezoid intersected by the newly inserted line segment is

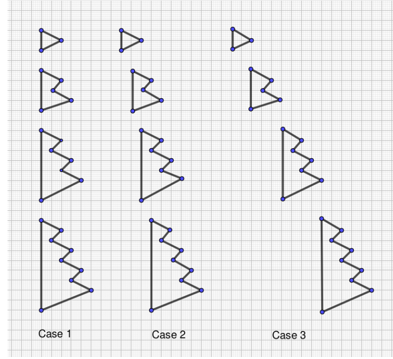


Figure 2.7: Arrangements of some convex and concave polygons

searched using both “search structure” and “trapezoidal map” with logarithm time. However, the deletion of a line segment from the map does not require locating the leftmost trapezoid containing the left endpoint of the line segment but the trapezoids that are “associated” with the line segment. The line segment passes through the vertices or overlaps with one of the edges partially or fully with the “associated trapezoids”. To locate these “associated trapezoids”, all the existing trapezoids in the trapezoidal map are scanned linearly with quadratic time complexity, while the existing “search structure” does not assist in locating the “associated trapezoids”. These “associated trapezoids” are merged and replaced in the trapezoidal map.

As the line segments may not be in general position, there could be an arbitrary number of adjacent trapezoids of the associated trapezoids. Each of these adjacent trapezoids is inspected to determine its adjacency with the merged “associated trapezoids”. So, in essence, due to the deletion of a line segment/polygon, the search and merging of the “associated trapezoids”, determining the top edge, bottom edge, left endpoint, right endpoint, and adjacent trapezoids of these merged trapezoids do not follow deterministic rules like splitting trapezoids while insertion of a new line segment. If the total number of trapezoids in the trapezoidal map, the number of associated trapezoids, and the number of adjacent trapezoids are N_t, N_a and N_{adj} respectively, the time complexity for deleting a line segment becomes $O(N_t + N_{adj} + N_a^2)$.

It can be observed that for every vertical slab (except for the first and last vertical slab), the level of the corresponding tree is equal to the number of line segments intersected on its RVL. As each level (except the root) of any such tree has a fixed (two) number of trapezoid nodes, the insertion or deletion of a polygon is dependent on the number of vertical slabs and line segments intersected in each vertical slab.

Let N polygons be arranged such that each polygon has $0, 1, 2, \ldots, N-1$ concave angles. These polygons can be arranged in many ways with no interleaves among themselves. These arrangements can be obtained by moving the polygons horizontally and/or vertically. In Figure 2.7, three arrangements of four polygons are considered. In these arrangements, the number of covered vertical slabs, N_{vb} lies between $MIN_{vb} = N + 1$ and $MAX_{vb} = 2 + 3 + 4 + \ldots + (N + 1) = \frac{(N^2+3N)}{2}$ having $\frac{(N^2+N)}{2}$ intermediate values, each being equally probable.

Thus, the total number of possible values of the vertical slabs is given by

$$SUM_{vb} = \quad (N+1) + (N+2) + \ldots + \frac{(N^2+3N)}{2}$$

and the expected number of vertical slabs is given by

$$E[N_{vb}] = \frac{SUM_{vb}}{\frac{N^2+N}{2}} \approx O(N^2)$$

The leftmost IRT, in which the leftmost vertex of the inserted/deleted polygon lies can be located with a logarithmic time of the number of vertical slabs. Similarly, the rightmost IRT can also be located with a logarithmic time. Thus, the expected time to locate either the leftmost or rightmost IRT during the insertion or deletion of a polygon is $O(\log N)$.

The number of line segments, N_{ls} intersected in any vertical slab for all arrangements lies between $MIN_{ls} = 2$ (Figure 2.7: Case 1) and $MAX_{ls} = (2 + 4N)$ (Figure 2.7: Case 3). It is to be noted that for all these arrangements, the sorted consecutive values of N_{ls} differ by 2 and thus, the number of possible values of N_{ls} is $2N + 1$. The sum of all these possible values

$$\begin{aligned} SUM_{ls} = & \quad 2 + 4 + \ldots + (2 + 4N) \\ = & \quad 2(2N+1)(N+1) \end{aligned}$$

. Thus, the expected value of the number of line segments intersected in each vertical slab is given by

$$E[N_{ls}] = \frac{2(2N+1)(N+1)}{(2N+1)} \approx O(N)$$

.

As each intersected line segment adds a new level in the tree corresponding to the vertical slab, the expected value of the height of a tree corresponding to a vertical

slab is also $O(N)$.

For any vertical slab, IHN or DHN can be equally found in any level between 2 to $O(N)$. Thus, the total number of searches is $N_{HN} = (2+3+4.....+O(N)) = O(N^2)$. Thus, the expected number of total searches for IHN or DHN in a vertical slab is

$$E[N_{HN}] = \frac{N_{HN}}{O(N)} \approx O(N)$$

If a polygon with H concave angles is inserted or deleted on the existing setup of N polygons, H vertical slabs will be updated. So the complexity for searching and updating the trees due to an insertion or deletion of a polygon with H concave angles is $O(NH)$.

It is to be noted that computing free spaces in the proposed data structure is much easier than finding those in a trapezoidal map. In the case of a trapezoidal map, the whole polygon needs to be examined to find out the trapezoids inside it. Whereas, as per the proposed data structure, the odd-numbered trapezoids in a vertical slab are the free trapezoids and the even-numbered ones are inside polygons.

2.5 Comparison with Some Similar Algorithms

The state-of-the-art related to updating planar subdivision due to the addition and deletion of the whole regular polygon is sparse.

A randomized incremental algorithm using a trapezoidal map, proposed by Berg et.al [7], has an expected complexity of $O(n \log n)$ for n line segment insertions. If the same trapezoidal map is used for deleting a line segment, it can be shown that the update to the data structure is $O(N_t + N_{adj} + N_a^2)$, where N_t is the total number of trapezoids in the trapezoidal map, N_a is the number of associated trapezoids of the deleted line segments and N_{adj} is the number of adjacent trapezoids of the associated trapezoids.

Chiang et. al. [15] have proposed a method to dynamically maintain a trapezoidal decomposition of a connected planar map where insertion and deletion of a vertex or an edge are completed with time complexity $O(\log^3 n)$. The total space complexity of this technique is $O(n \log n)$. But this technique does not apply to a map with multiple polygons, considered in this work.

A dynamic data structure for a monotone planar subdivision to find out the point

location has been proposed by Lars et. al. [4] primarily focusing on minimizing the I/O operations by effectively using external memory to store the subdivision. It supports update operations for vertices and edges with time complexity $O(\log_B n)^2$, where n is the number of vertices and B is the number of disk blocks. However, this technique is also not suitable for maps with multiple polygonal obstacles as it is meant for working in a monotone planar subdivision of a connected graph.

2.6 Conclusion

In this chapter, a data structure and its corresponding algorithm are proposed for maintaining a vertical cell decomposition of a map after the insertion and deletion of a polygon. The problem of maintaining a map is a common problem in robot motion planning. The free space of the Planar Subdivision Map is the odd-numbered trapezoids, while the even-numbered trapezoid lies inside the polygons. If the concave polygons are arranged in a specific order as shown in the figure 2.7, the expected time complexity for updating the relevant trees is $O(NH)$ where N is the number of existing polygons and H is the number of concave angles of the polygon which is inserted or deleted.

This work can be extended to plan the placement of surveillance cameras in a given map of obstacles. As the free trapezoids obtained from the planar subdivision designate the free space of the map and are organized using a simple data structure, it would be an interesting problem to determine the location of the vision circles of the cameras so that surveillance coverage can be maximized. This work can be further improved by proposing a data structure for the planar subdivision ensuring a logarithmic time complexity for updating the trees due to the insertion or deletion of a polygon in the map.

In the next chapter, a discrete method has been proposed to find the strategic placement of vision circles in free spaces of an environment with convex and concave polygons to cover free spaces, while avoiding overlapping among circles and polygons. This method uses the data structure outlined in the current chapter. A heuristic method using a genetic algorithm is also proposed in the next chapter to find the placement of circles in an environment with convex polygons such that coverage of free spaces is maximized as much as possible and overlapping among circles and polygons is kept within a limit.

Chapter 3

Placement Of Vision Circles

This chapter starts with deriving the complexity class of the problem related to the coverage of free space with circles inside a rectangular bounded box in the presence of convex and concave polygons. A discrete and a heuristic method is also presented in this chapter to cover the free space of an environment with circles.

The covering problem and the art gallery problem are both foundational challenges in computational geometry, sharing similar goals but with unique emphases. These problems have received extensive attention and development over several decades, resulting in the creation of numerous solutions and techniques.

The "Art Gallery" problem states about placing a minimum number of guards at strategic locations such that the entire gallery is visible to the guards. In most practical problems, these guards can be replaced with cameras with a 120^0 span. If these cameras have a 360^0/panoramic view with a bounded vision, then these can be represented as circles, referred to in this work as "vision circles". In this chapter, "circle" and "vision circle" are used interchangeably in the context of coverage of free space.

Frequently, the resources allocated for surveillance are significantly less compared to the area requiring coverage. This results in a high number of blind spots in surveillance, which are crucial vulnerabilities for any system or organization. So the target would be to cover free space with the "vision circles" as much as possible while ensuring minimum overlapped areas among "vision circles" themselves or with polygons.

3.1 Literature Survey

This section presents a literature survey on the placement of circles in environments containing both convex and concave polygons.

Chrysostomou et.al.[16] employ a swarm-based meta-heuristic algorithm known as the bee colony algorithm to achieve an optimal solution for space coverage using multiple cameras. Additionally, they aim to minimize the cost associated with camera placement necessary for achieving optimal area coverage.

Dickerson and Scharstein [20] provide a method for the optimal placement of a polygon such that it contains the maximum number of points out of a set of n points.

Chazelle and Lee [13] provide a solution to a weighted circle placement problem of fixed radius. Their problem statement is described to be equivalent to the maximum weighted clique problem. They provide a $\mathcal{O}(n^2)$ algorithm solution for the problem.

Pirani et.al. [53] study the optimal sensor placement in networked control systems for improving the detectability of cyber–physical attacks. The problem is formulated as a game between an attacker and a detector. The attacker chooses a subset of nodes within the network to target for attack, while the detector strategically selects nodes to deploy sensors. The objective of the detector is to maximize the detectability of signals indicating an attack.

Caplan et. al. [12] present real-life findings regarding the effectiveness of crime deterrence following the deployment of CCTV cameras in an urban environment. They demonstrate a decrease in crime rates across the city following the installation of cameras that are visible to the general public. This work goes to show the importance of work in this field and the impact it can have.

Mukhopadhyay and Rao[48] came up with a strategy to find the largest rectangle bounded by four points taken from a given set of points.

Daniels *et al.* [18] have proposed a solution for the applications where an internal approximation to a general polygon is needed. This approximation is given in the form of the largest axis-parallel rectangle found inside the polygon.

Nandy *et al.* [49] outlined a method to locate an empty rectilinear rectangle of the maximum area from a set of non-intersecting line segments inside a rectangular boundary.

There are also many studies related to the circle packing problem.

In the study [56] a packing (layout) problem for several clusters (groups) made up of convex objects (such as circles, ellipses, or convex polygons) is taken into consideration. Subject to no overlap between objects within a cluster, the clusters must be crammed within a specific rectangular container. The convex hull of the objects that make up each cluster is used to represent it. If the convex hulls of two clusters don't cross, the clusters are considered non-overlapping. A cluster is considered to be contained in the container if its convex hull is present. The objects in the cluster can all be constantly translated and rotated, and they are all the same shape (although varying sizes are acceptable). Creating a maximum sparse layout for clusters while maintaining non-overlapping and confinement requirements for clusters and objects is the goal of optimized packing. In this context, the term "sparse" denotes clusters that are suitably separated from one another. To analytically explain non-overlapping, containment, and distance restrictions for clusters, new quasi-phi-functions and phi-functions are presented. After that, the layout problem is formulated as a continuous nonlinear non-convex problem. The development of a revolutionary method for finding regionally ideal solutions. The effectiveness of our strategy is demonstrated through computational findings. Although a container-loading difficulty served as the impetus for this research, comparable concerns naturally emerge in many other packing, cutting, and clustering problems.

In this paper [51] they proposed a nonlinear mathematical model for the problem of packing circles, convex and non-convex irregular polygons, within a rectangular envelope to be produced, obeying containment constraints and non-overlapping constraints; the objective of the problem is to minimize the area of the rectangular envelope. Using examples from the literature that simultaneously deal with circles and polygons, computational tests were conducted. In most cases, different solutions with small or equal rectangular envelope areas were discovered, and the execution time was extremely short. This suggests that the model is effective in terms of computation.

In the study [3], investigate the problem of finding a minimum-area container for the translation-based disjoint packing of a set of convex polygons. We specifically take into account axis-parallel rectangles or any convex set as containers. We create effective constant factor approximation techniques for both NP-hard optimization issues.

In this study [63] discussed convex packing of two free-rotating and translating

convex polygons in the plane, P and Q. A convex packing of P and Q with the minimum area is one whose area is minimized. They create an effective $O((n + m)nm)$ time deterministic algorithm for finding a real minimal area convex packing of P and Q using this discretization, together with a delicate algorithmic design and thorough complexity analysis. n and m are the numbers of vertices in P and Q, respectively.

In this study [62] specify the densest packing of equal circles in a square problem. The problem of finding the optimal placement of N identical, non-overlapping, circles with maximum radius in the unit square is a well-known challenge both in classical geometry and in optimization. A database of putative optima is currently maintained at `www.packomania.com`.

Recently, in this study [1] can uncover better configurations for several cases by the creative application of a very straightforward global optimization strategy.

The central rectangle must be located in the middle of the final arrangement, and the aspect ratio of the container must also fall within a certain range.

This study [14] proposes a heuristic technique for solving a specific NP-hard 2D rectangular packing issue. The most important aspect of the suggested algorithm is a greedy constructive procedure, which involves packing the rectangles into the container one at a time and packing each rectangle using the angle that occupies the most space while maintaining the highest degree of fit. 35 well-known benchmark instances are divided into two groups to assess the suggested algorithm. According to computational results, the suggested packing problem algorithm performs better than the existing technique. For the first group of test instances, solutions with an average filling rate of 99.31 can be obtained; for the real-world layout problem in the second group, the filling rate of the solution is 94.75

Bin-packing Problem (BPP) involves attempting to fit a specific number of objects into a fixed-size container while adhering to a variety of restrictions. In this study [67] the mathematical model of a two-dimensional bin-packing problem was developed by taking into account the non-superposition of items and the constraint of the box in the form of coordinates to reproduce common plate-cutting difficulties based on rectangles and circles. A Genetic Algorithm(GA) based approach was then suggested to resolve this two-dimensional BPP with rectangular and circular areas. The single change of mutation probability and the unified change of mutation probability are used to optimize the GA parameters. The efficiency of the suggested

technique for addressing two-dimensional bin-packing issues with rectangular and circular regions is supported by simulation results.

In existing literature, there are some works on finding a large rectilinear rectangle that can cover maximum free space inside a bounding box. However, there is no work where strategic locations are determined in the free space to place multiple circles of varied radii to cover the free space effectively. Moreover, once a circle is placed, it should be treated as "occupied space" and combined with existing polygons to avoid redundant coverage of free spaces by remaining circles. No work could be found in existing literature where this issue of redundant coverage is taken care of.

In this chapter, a discrete and a heuristic method is proposed for placing circles within the free space of a map that includes convex and concave obstacles.

The discrete method effectively covers the free space within a rectangular area, addressing redundant coverage by merging placed circles with existing polygons to form larger ones. Initially, it is assumed that no polygons touch each other or the rectangle's boundary and that there are no saw-tooth polygons within the rectangular bounding box. This configuration is referred to as a map.

The discrete algorithm utilizes a data structure based on slab subdivision, as detailed in chapter 2, where each slab contains a set of trapezoids. The algorithm aims to strategically place vision circles of various radii within the bounding box, taking mobile agents into account.

To identify important regions, the method uses centrality scores to analyze free trapezoids. It calculates large rectangles to determine optimal spots for placing vision circles, employing different objective functions to explore various placements. Once placed, circles are merged with intersecting polygons, and the map is updated accordingly.

Given the absence of a deterministic solution, this work also introduces a heuristic approach employing genetic algorithms. The goal is to identify an optimal arrangement of circle placements maximizing coverage of available space among all explored arrangements. Experimental findings demonstrate that the area of free space coverage expands with the number of generations the genetic algorithm operates through. Alongside circle placement, this method also minimizes overlap between circles, and between circles and existing convex polygons.

The remaining sections of this chapter are arranged as follows: classification of the

problem dealing with circle placement in map with polygons [Section 3.2], the proposed method to find free rectangular spaces [Section 3.3.1], degenerate cases for rectangle finding method [Section 3.3.1], analysis of rectangular finding method [Section 3.5.1], proposed algorithm for strategic placement of vision circles[Section 3.3], analysis of the complete algorithm [Section 3.5] and review and analysis of the proposed algorithm with some similar algorithms [Section 3.5.3].

3.2 Problem Classification For Circle Placements In Map With Polygons

The placement of ”vision circles” as mentioned above can be expressed as a problem (O), which deals with the placement of circles in the free space of a map with polygons while ensuring minimum intersection among circles and between circles and polygons.

A much simpler version of the problem O can be easily shown as NP-complete. In this simplified problem (S), instead of coverage using vision circles in the presence of polygons, coverage using free rectangles in the presence of occupied rectangles is considered. In the actual problem, the vision circle can be placed in an uncountable number of point locations inside free space, but in the case of problem S, any free rectangle is placed only in a designated fixed place. To prove the fact that S is an NP-complete problem, any given input of $0/1$ Knapsack Problem (F), which is already an NP-complete problem, can be reduced to an input of problem S. Problem F can be defined as given weights and profits of n items, these items need to be put into a knapsack with a given capacity of maximum weight, so that total profit can be maximized.

The inputs of problems F and S are as follows.

Input of Problem F: Pairs of $(profit, weight)$ of k items, $(p_1, w_1), \ldots, (p_k, w_k)$ and maximum weight $W \leq \sum_{i=1}^{k} w_i$

Input of Problem S: k free rectangles and k occupied rectangles of unit height placed inside a bounded rectangular environment such that

1. Each free rectangle is overlapped with only one occupied rectangle.
2. Upper and lower boundaries of the free rectangle and corresponding occupied rectangle follow the same horizontal line.

3. Area of overlapping between the free rectangle and occupied rectangle of a pair (p_i, w_i) is $c_i = d \times \frac{w_i}{\sum_{i=1}^{k} w_i}$, where $d = \max(p_1, \ldots, p_k)$.
4. Maximum value of the total overlapped area is any chosen value s sq. units, such that $s \leq d$.

The following steps to map the input of problem F to the input of problem S are as follows.

a) An enclosed rectangular environment(E) of height k units and length of $2d$ is drawn.

b) For any pair of (p_i, w_i), a free rectangle R_i and occupied rectangle O_i are drawn one after another horizontally such that they are placed i units below the top boundary of E, free rectangle touches the left boundary of E and corresponding occupied rectangle touches the right boundary of E. The length of a free rectangle is $(p_i + c_i)$ unit whereas the length of an occupied rectangle is $(2 \times d – p_i) + c_i$ unit. Hence the area of the overlapped portion of this free rectangle and the occupied rectangle is c_i sq. units.

The problem S involves selecting a subset of free rectangles from the previously described setup. These selected rectangles must satisfy the condition that, when placed in the designated positions, the maximum total overlapped area is limited to s square units. It can be noted that all the occupied rectangles will always be present in the environment in their designated places, only a subset of rectangles in free space will be chosen to satisfy the maximum value of the total overlapped area constraint. It is also evident that the time complexity of mapping an input of problem F to input of problem S is $O(k)$, where k is the number of pairs of profit and weight values in the original Knapsack problem F.

Figure 3.1 shows the mapped rectangles and occupied rectangles in the rectangular environment corresponding to the pairs of profit/weight combinations mentioned above.

In figure 3.1, $R_1, \ldots, R_k$ are the rectangles placed in free space, and $O_1, \ldots, O_k$ are the corresponding occupied rectangles. $I_1, \ldots, I_k$ are the overlapped areas. For the topmost pair, $ABCD$ is the free rectangle, $EFGB$ is the occupied rectangle and $BFGC$ is the overlapped area.

A small example is used below to explain the steps to map an input instance of F to

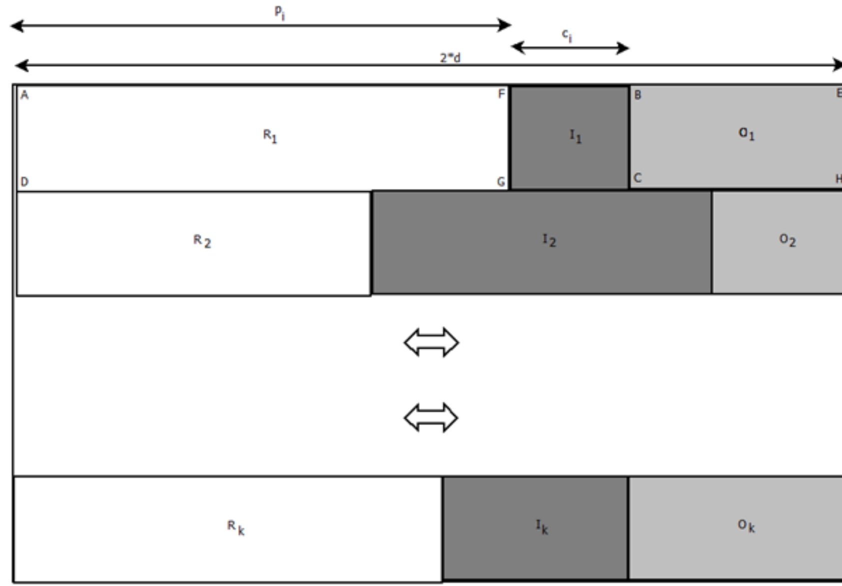


Figure 3.1: Input for the simplified problem S

an instance of S.

Input of F: No of items $k = 3$, Maximum allowed weight $W = 80$, pairs of $(profit, weight) = (4, 10), (6, 30), (10, 60)$.

Input of S:

Length of rectangular environment(E) to be taken as $2 \times d$ of all profits i.e. $2 \times 10 = 20$ units. As there are 3 pairs of $(profit, unit)$, there will be 3 rectangles on the left representing free spaces and 3 rectangles on the right representing corresponding occupied rectangles inside E. Each free rectangle and corresponding occupied rectangle have an equal height of one unit and share common horizontal lines for the top and bottom boundaries. So, the overlapped areas between a free rectangle and the corresponding occupied rectangle are also rectangles with one unit of height.

Lengths of overlapped rectangles are taken from "weight" values of the corresponding "pairs" in proportion to the total weight of all pairs. As, the total weight of all pairs is 100, the lengths of overlapped rectangles are $10 \times \frac{10}{100} = 1, 10 \times \frac{30}{100} = 3$ and $10 \times \frac{60}{100} = 6$ respectively. The length of 3 free rectangles is calculated by adding the length of an overlapped rectangle with 3 profit values i.e. $1 + 4 = 5, 3 + 6 = 9$, and $6 + 10 = 16$ units respectively. The length of 3 occupied rectangles is calculated by subtracting the length of the corresponding free rectangle from E and adding the length of the overlapped rectangle, i.e. $20 - 5 + 1 = 16, 20 - 9 + 3 = 14$ and $20 - 16 + 6 = 10$. The maximum value of the total overlapped area between a free

rectangle and the occupied rectangle must be chosen as any value less than $d = 10$ sq units.

Figure 3.2 shows all of these 3 pairs of free and occupied rectangles placed inside the rectangular environment.

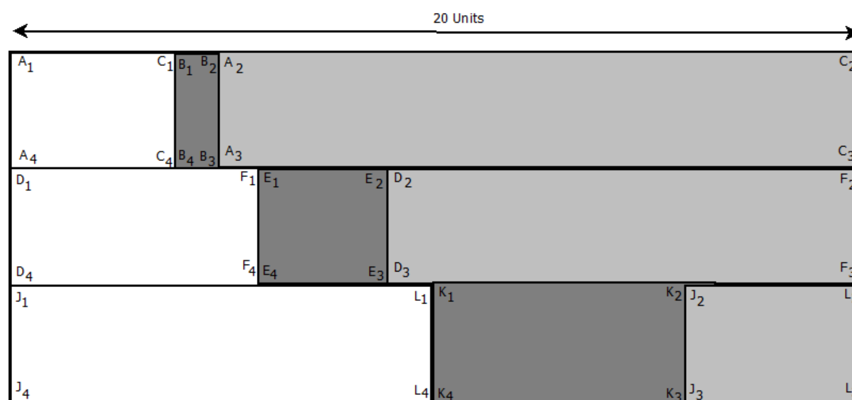


Figure 3.2: Mapped input of problem S from example input of problem F

In figure 3.2, $A_1A_2A_3A_4$, $D_1D_2D_3D_4$ and $J_1J_2J_3J_4$ are 3 rectangles of lengths $5, 9$ and 16 units respectively. $C_1C_2C_3C_4$, $F_1F_2F_3F_4$ and $L_1L_2L_3L_4$ are 3 corresponding occupied rectangles of lengths $16, 14$ and 10 units respectively. If all of these 3 pairs of free rectangles and occupied rectangles are placed inside a given environment E, then the corresponding overlapped rectangular areas would be $B_1B_2B_3B_4$, $E_1E_2E_3E_4$ and $K_1K_2K_3K_4$ with lengths $1, 3$ and 6 units.

Input instance of problem S can also be mapped to input of problem O quite easily. The rectangular environment and occupied rectangles of problem S are kept as it is and each free rectangle is replaced with one circle. These circles will be placed inside the free space of the environment as part of the solution of the problem O. There can be many possible mappings between free rectangles and corresponding circles. The mapping technique considered here tries to maintain the essence of the size difference between free rectangles while choosing the size of the circles. The ratio of diameters of any two circles is taken as the same as the ratio of lengths of corresponding free rectangles. The smallest circle will have diameter (D_{min}) equal to the diameter of the in-circle of the free rectangle with minimum area. The diameter of the other circle is calculated as $D_{min} \times \frac{L}{M}$, where L is the length of the corresponding free rectangle and M is the length of the free rectangle with minimum area. Continuing the example given above, mapped input for the problem O consists of a rectangular environment of length 20 units with 3 occupied rectangles of one unit of height placed horizontally one after another. Lengths of these occu-

pied rectangles (placed from top to bottom) are $16, 14$, and 10 units. Value of D_{min} is 1 unit. The diameter of 3 circles to be placed inside the free space of the rectangular environment are $1, 1 \times \frac{9}{5}$ and $1 \times \frac{16}{5}$. Figure 3.3 shows the rectangular environment along with these 3 occupied rectangles inside it and 3 circles to be placed inside the free space of the environment. In figure 3.3, $C_1C_2C_3C_4$, $F_1F_2F_3F_4$ and $L_1L_2L_3L_4$ are occupied rectangles of lengths $16, 14$ and 10 units respectively, placed inside environment E. V_1, V_2 and V_3 are the 3 circles with diameters $1, \frac{9}{5}$ and $\frac{16}{5}$ units to be placed inside the free space of environment E. For illustration purposes, the heights of free rectangles, occupied rectangles, and diameters of circles in figures 3.2 and 3.3 are taken on a higher scale.

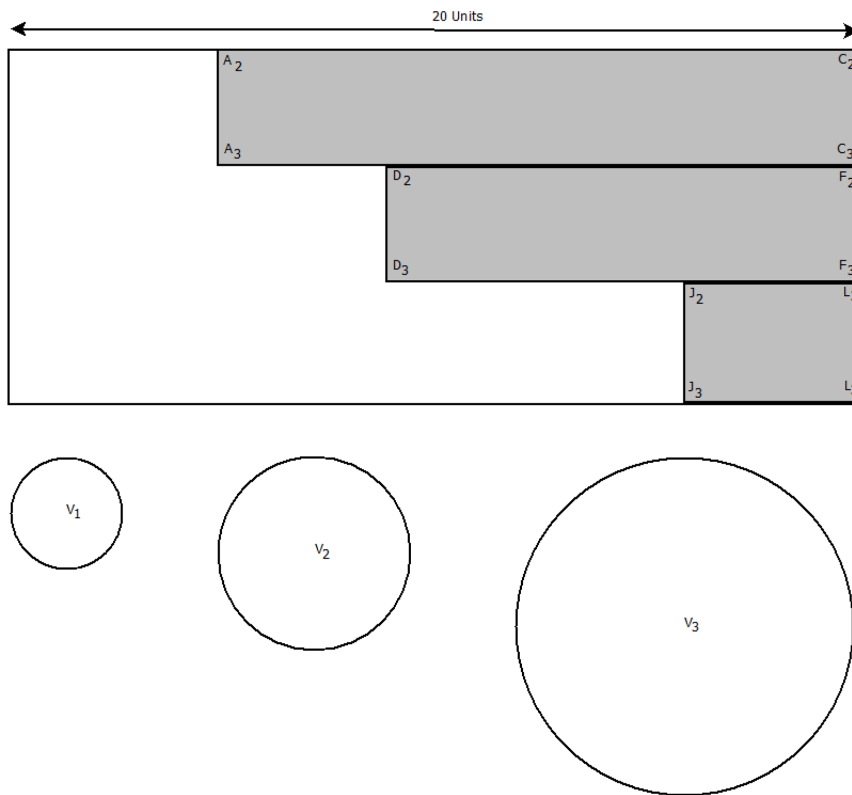


Figure 3.3: Mapped input of problem O from example input of problem F

In the following sections, proposed methodologies for finding the free space are explained.

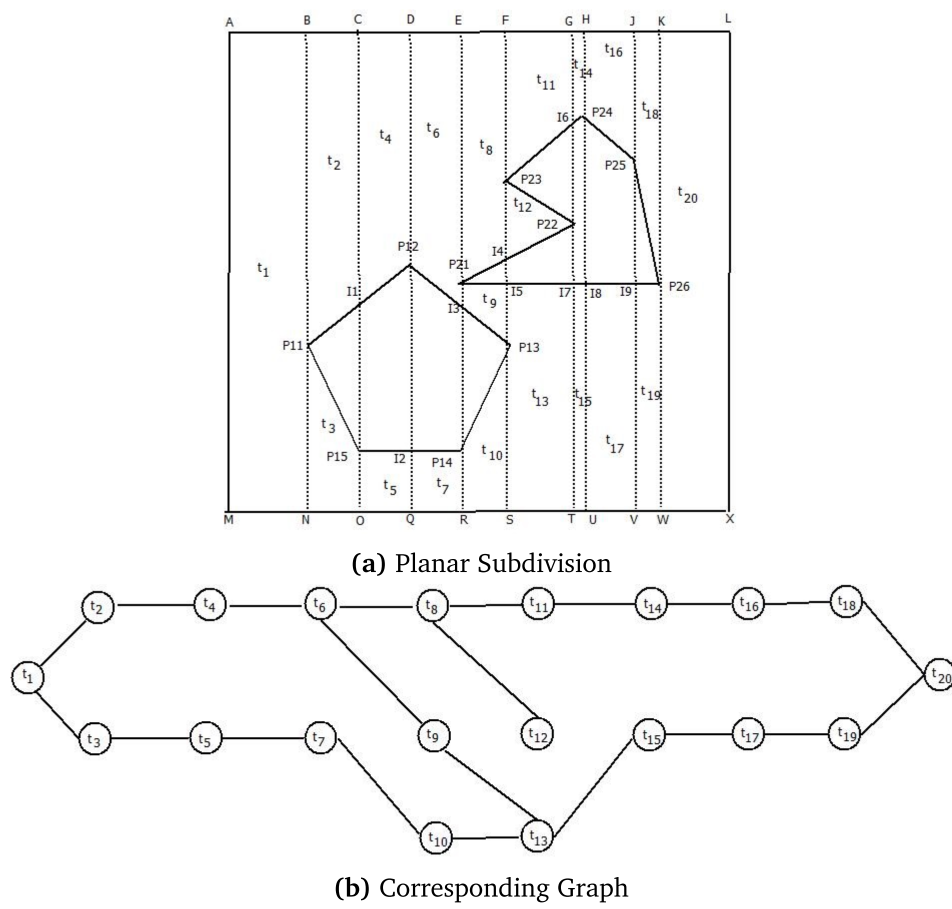
3.3 A Discrete Method for Strategic Placement of Vision Circles In Presence of Convex and Concave

In this section, a method is proposed to strategically place vision circles inside free spaces of an axis-parallel rectangular map having convex and concave polygons (obstacles), L_P. Vertices of L_P are ordered in a clockwise manner and can be traversed in both front and back. The map is subdivided into slabs, $S = \{S_{LM}, S_1, \ldots, S_M, S_{RM}\}$ using vertical lines along the vertices of the polygons [8]: S_{LM} is the left-most slab; S_{RM} is the right-most slab; and $S_i, i > 0$ are the in-between slabs. The slabs are ordered based on the X-coordinate of the slab's left vertical line. Each slab is further subdivided into trapezoids ordered from top to bottom. The slabs are further divided into trapezoids i.e. $S_i = \{t_j | j > 0\}$; and the slabs S_{LM}, S_{RM} are itself the trapezoid. A trapezoid is considered free if no part of any polygon lies inside it. A graph $G_u(V, E)$ is constructed with the free trapezoids of the planar subdivision and an edge exists between two vertices if they share a common side in-between them.

A hash table $L_P Trap$ keeps a list of trapezoids covering a polygon, where the key for the hash table is the polygon itself. Similarly, a hash table Map_R keeps track of the rectangle for every trapezoid of S, ensuring that each rectangle includes the corresponding trapezoid partially or completely. Section 3.3.1 describes the proposed method to find these rectangular free spaces and populate Map_R.

In Figure 3.4a, the slabs are enclosed by vertical dotted lines on the left and right drawn through every vertex of the polygons. It is observed that an even-numbered trapezoid is occupied by a polygonal and the odd-numbered trapezoids are free. The graph shown in Figure 3.4b is the corresponding graph of free trapezoids of the planar subdivision of Figure 3.4a.

The vision circles are placed inside the rectangles of Map_R in which it completely fits, while not completely inside a trapezoid but touch at least two sides of it, and based on a score obtained from an "Objective Function" [Section 3.3.2] and "Centrality Measure" [Section 3.3.2][44]. After a vision circle is placed, it is approximated [Section 3.3.3] as a polygon, and the given map is updated with this. This is repeated until no existing vision circles can be placed [Algorithm 6].



(a) Planar Subdivision

(b) Corresponding Graph

Figure 3.4: Planar Subdivision and its corresponding graph

3.3.1 Finding Free Rectangular Spaces

In this section, a method is proposed for finding a list of large axis-parallel rectangular free spaces in a $2D$-Euclidean map with concave and convex polygons.

The map is subdivided into slabs and trapezoids [8] by drawing vertical lines along every vertex of all the polygons, which are stored in an ordered list of vertical slabs [21]. In Figure 3.5a, the slabs are the vertical blocks enclosed by the blue lines drawn through every vertex of the polygons. The trapezoids in a slab are ordered from top to bottom. The trapezoids are identified by an index, which is the slab number and the trapezoid number (starting from 1) in the slab. It is observed that every even-numbered trapezoid in a slab is occupied, i.e. it is part of a hole and odd-numbered trapezoids are free.

A directed graph, $G(V, E)$ is constructed from all free trapezoids while traversing slabs from left to right and free trapezoids from top to bottom in a slab. Free trapezoids become vertices ($V = \{v_1, \ldots, v_N\}$) of graph G and $\overrightarrow{e}_{ij}$ becomes an edge if

trapezoids corresponding to vertices v_i and v_j share a common vertical side. Figure 3.5b is the graph for the map Figure 3.5a.

A path $T_L = \{t_1^{v_i}, t_2^{v_j}, \ldots, t_n^{v_k}\}$ in $G(V, E)$ is a contiguous chain of trapezoids starting from vertices with $degree^{-}(v_i \in G) = 0)$ and ending at vertices with $degree^{+}(v_k \in G) = 0$. Here $t_i^{v_a}$ means that the trapezoid represents vertex $v_a \in V$ and its index is i in the path T_L. The boundary, $bndry$ of a path T_L is the ordered list of all constituent trapezoid sides which are adjacent to the open space of the map. For every free trapezoid, the Algorithm 5 finds a free contiguous rectangular space that includes at least the current trapezoid partially or completely.

A list of paths P_L for all combinations of vertices of $degree^{+}(v_i \in G) = 0$ and $degree^{-}(v_i \in G) = 0$ is computed from $G(V, E)$. For every path $T_L \in P_L$, Algorithm 5, finds the largest rectangle among all free rectangles computed corresponding to every trapezoid in T_L. It gives a list, Map_R, of pairs of $(t_i^{v_{index}}, R)$ for the path; $t_i^{v_{index}} \in T_L$ and R its corresponding rectangle.

Finding Rectangles For A Path

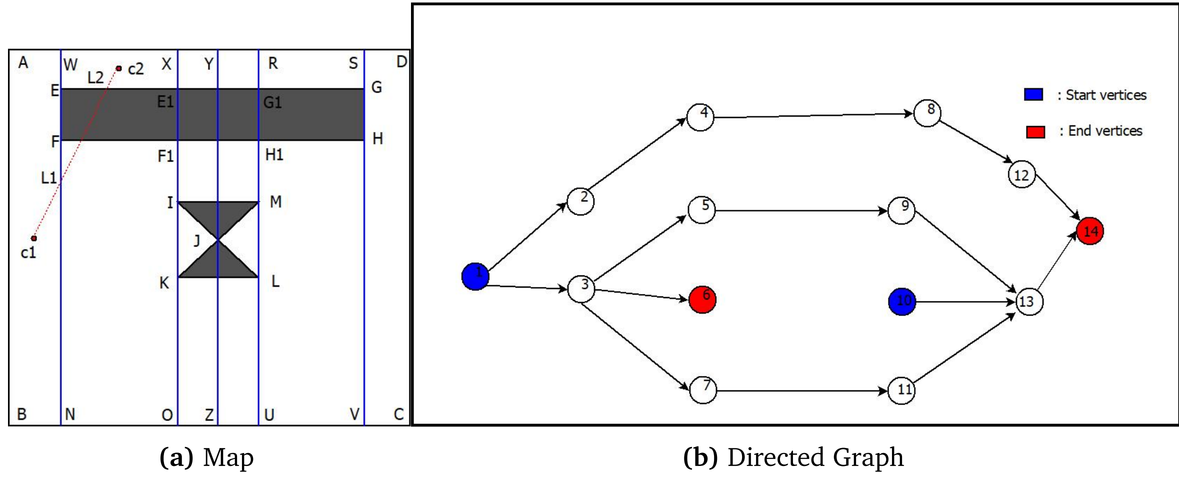
In this section, a method for finding the rectangles in a path is described.

Algorithm 5 finds all the free "axis-parallel rectangles"[18] corresponding to every trapezoid in a path. The inputs to the function are the list of trapezoids of a path T_L of $G(V, E)$, the number of trapezoids in the path (n), a step length Δ and the list of line segments in the path boundary $(bndry)$. The boundary is found by traversing through the trapezoids in the path and adding the line segments that are not hidden in the path to a list. In Figure 3.5b, the boundary for the path $\{1, 2, 4, 8, 12, 14\}$ is the polygon $ABNEGVCD$. A list of pairs $Map_R = \{(t_i^{v_{index}}, NR) | t_i^{v_{index}} \in T_L, NR =$ largest free rectangle $\}$ is given by the method.

The proposed method, Algorithm 5 can be divided into three parts: the first part, where the point P shifts/traverses along T_L; the second part, where the intermediate largest feasible rectangles are computed using axis-parallel orthogonal rays drawn from P; and the third part, where the rectangle is fitted into the path boundary.

Traversal of path with a point P

A list of connected line segments $L = \{l_1, l_2, \ldots, l_{n-1}\}$ is maintained, where $l_i = (c_i, c_{i+1})$ and c_i is the centre of a trapezoid belonging to $C = \{c_1, c_2, \ldots, c_n\}$ along the path, T_L. The large rectangles in this path can be computed from the orthogonal



(a) Map

(b) Directed Graph

Figure 3.5: An example map subdivided into slabs and its corresponding directed graph

axis-parallel bi-directional rays, L_x and L_y drawn from a point $P(P^x, P^y)$, lying on a line segment l_i. It is to be noted that L_x and L_y are extended up to the boundary, $bndry$ of the path. P is shifted along $l_i \in L$ by Δ from c_1 to c_n while checking for the largest axis-parallel rectangle from each shifted point. The updated coordinates of the point P after every shift can be given by:

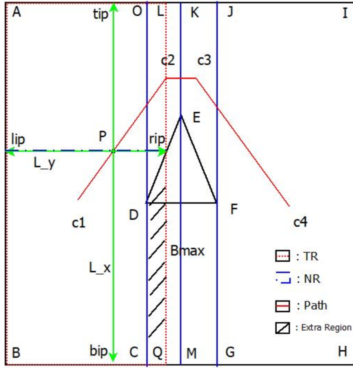
$$P_{new}(P^x_{new}, P^y_{new}) = \\ (P^x + \Delta cos(\Theta), P^y + \Delta sin(\Theta))$$

where $\Theta = \tan^{-1}(l_i)$.

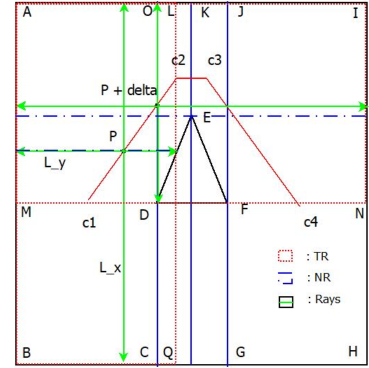
The shifting ends for a line segment l_i when P goes beyond the line segment l_i. At each shift, L_x and L_y from the point P compute the four meeting points of this pair of rays with the $bndry$ of the path. The largest rectangle is found by measuring the largest axis-parallel distances from P. In Figure 3.6, $c1$ is the leftmost point of the list of line segments $L = \{(c1, c2), (c2, c3), (c3, c4)\}$. The figure shows that the point P shifts along the line (in red) joining the centres $C = \{c1, c2, c3, c4\}$.

Computing the Intermediate Largest Feasible Rectangle

The two rays L_x and L_y from P are used to compute the largest feasible rectangle for that point. The four intersection points of these rays with the boundary of the path are calculated as seen in Figure 3.6. L_x and L_y intersects the $bndry$ at (tip, bip, lip, rip). The height of the rectangle is $|L_y|$ and the length of the rectangle is $|L_x|$.



(a) Rays from point P



(b) Rays from point $P + \Delta$

Figure 3.6: Computation of a rectangle using rays from point P

The function $IntSctn()$ computes the top and bottom (tip, bip) points. It is obvious that L_x always intersects with the top and bottom line of the trapezoid in which P belongs [Function 5, Line 9].

The function $LRIntSctn()$ finds the pair $[lip, rip]$ using the line L_y and the line segments of boundary ($bndry$). It checks if the Y coordinate of the point P falls within the range of Y coordinates of the endpoints of the line segment. This check is done for all line segments in the boundary, $bndry$. The line L_y intersects those line segments for which the check holds true. lip and rip are the closest left and right intersection points of L_y from P.

The rectangle, TR is generated from the co-ordinates of the four intersection points. The top line of the rectangle becomes $\{(lip^x, tip^y), (rip^x, tip^y)\}$. Similarly the bottom line becomes : $\{((lip^x, bip^y), (rip^x, bip^y))\}$.

Fitting the Rectangle into the Boundary

The rectangle found by the rays (TR) may contain outside regions in it as seen in [Figures 3.6] as the shaded region. Thus, the rectangle is cropped by $Crop(R(t_i^{index}))$ (Algorithm 5 described in section 3.3.1).

The function returns the list (Map_R) as the output. This list contains a pair: the index of a trapezoid ($t_i^{v_{index}}$) and the largest rectangle (NR) found when the point P is inside the same trapezoid. The new rectangle NR, after cropping replaces the existing rectangle in Map_R (if present and larger than the existing rectangle) or is inserted as a pair: $(t_i^{v_{index}}, NR)$. The v_{index} of the trapezoid depends on which trapezoid the point belongs to when the rectangle is found, since the point P shifts.

Algorithm 5 $RectsForaPath()$

Require: n; $T_L : \{t_1, t_2, \ldots, t_n\}$; Δ; $bndry$
Ensure: $Map_R : \{(t_i^{v_{index}}, NR_1), \ldots, , (t_i^{v_{index}}, NR_n)\}$;

```
 1: C = {c_1, c_2, ..., c_n}, Map_R = φ
 2: for all t_i, t_{i+1} ∈ T_L, where 1 ≤ i < n do
 3:     PtP(P^x, P^y) = c_i(c_i^x, c_i^y)                                  ▷ P at t_i
 4:     while P ∈ l_i, l_i ∈ L do
 5:         if P ∈ T_L then
                                               ▷ Create two lines : x = P^x and y = P^y
 6:             L_x = Line(P^x, 0), L_y = Line(0, P^y)
 7:             tip(tip^x, tip^y) = (0, 0), bip(bip^x, bip^y) = (0, 0),
 8:             lip(lip^x, lip^y) = (0, 0), rip(rip^x, rip^y) = (0, 0)
 9:             if P ∈ t_i then
10:                 tip = IntSctn(L_x, t_i.tLine)
11:                 bip = IntSctn(L_x, t_i.bLine)
12:             else
13:                 tip = IntSctn(L_x, t_{i+1}.tLine)
14:                 bip = IntSctn(L_x, t_{i+1}.bLine)
15:             end if
16:             [lip, rip] = LRIntSctn(L_y, T_L, bndry, P)
17:             TR = ((lip^x, tip^y)(rip^x, tip^y)(rip^x, bip^y)(lip^x, bip^y))    ▷ TR : Temporary
    Rectangle
18:             NR = Crop(TR, T_L)                                  ▷ NR : New Rectangle
19:             if P ∈ t_i then   ▷ Rect(Map_R, t_i^{v_index}) gives the rectangle w.r.t t_i^{v_index} in Map_R
20:                 if Area(NR) > Area(Rect(Map_R, t_i^{v_index})) then
21:                     Replace R of Map_R(t_i^{v_index}, R) with NR
22:                 end if
23:             else
24:                 if Area(NR) > Area(Rect(Map_R, t_{i+1}^{v_index})) then
25:                     Replace R of Map_R(t_{i+1}^{v_index}, R) with NR
26:                 end if
27:             end if
28:             P(P^x, P^y) = (P^x + Δ cos(θ), P^y + Δ sin(θ))    ▷ Shift P by Δ along the line
    segment
29:         end if
30:     end while
31: end for
```

Crop function

The rays drawn from P do not check for any trapezoids not belonging to the path or a polygon above or below them. This sometimes results in rectangles, which partially or fully cover trapezoids not belonging to T_L. Since the objective is to find large empty rectangles within the boundary of the path, these offending rectangles are cropped to fit within the boundary region. In [Figure 3.6], the initial rectangle created (TR) (red dotted line) has some "extra region" (shaded); and the new bottom line of the cropped rectangle (in blue dotted line) is shown for the cropped rectangle (NR).

The area of a rectangle can change after cropping due to the possible presence of large outside regions. Therefore, cropping is done immediately after a rectangle TR is computed. The $Crop()$ function uses the original rectangle and the trapezoid list for the path. In [Figure 3.6], the trapezoid with center $c1$ has an "extra region" in the context of TR below it which can span over multiple slabs. The function iterates through all the trapezoids in the path from left to right and computes the extra region (contributing to the rectangle) for each slab that divides the rectangle respectively. The "Top-Bottom Property" is used in $Crop()$. Two lists, $Tvect$ and $Bvect$ keep the cropped heights from the top and bottom line of the rectangle respectively.

The "Top-Bottom" Property

While traversing the graph from left to right, a path is divided above and below a polygon. Since only the trapezoids belonging to the path are considered valid spaces for a rectangle to fit, there cannot exist two trapezoids belonging to the same path in the same slab separated by a polygon. Thus, only one of the two trapezoids formed belongs in the path, and the remaining region in the slab is "extra". The remaining region in the slab should always be adjacent to either the top or bottom line of the rectangle but not both and they may or may not be adjacent to the left or the right side of the rectangle.

For all the contributing trapezoids, it is checked if the highest point from the top line of the trapezoid has a Y co-ordinate higher than that of the rectangle's Y co-ordinate of the top line and stores the difference in the list $Tvect$. Similarly, the function checks the same for the bottom line and stores in the list $Bvect$. The reason for this computation is, that if the lowest point of the bottom line of the contributing trapezoid is lower than that of the rectangle, there is some outside region present in the slab of the trapezoid inside the rectangle [Figure 3.6a]. This also applies to the top line of the rectangle.

The function then adds $Tmax = \max(Tvect)$ to the Y co-ordinates of the top line and subtracts $Bmax = \max(Bvect)$ from the Y co-ordinates of the bottom line and the cropped rectangle is NR. Therefore, P (from which the rectangle is created) never falls outside NR, ensuring every trapezoid has at least one rectangle that contains the trapezoid even if partially, even though the rectangle may not leave the "maximum area".

Degenerate Cases For Rectangular Finding Method

In this section, some degenerate cases arising in this work are discussed.

1. An extra rectangle should always be computed from the center of the last trapezoid in the path as P can overstep the center $c_{i+1} \in l_i$ when $|l_i| \mod \Delta \neq 0$. If $\Delta > |l_{n-1}|$ or $P + \Delta$ oversteps the centre c_n in such a way that no rectangle for t_n is computed, $n - 1$ rectangles is computed.

2. In some cases, $l_i \in L$ can partially lie outside the boundary of the path. If P lies in the former region, P shifts, but there is no computation of the rectangle, until the P lies inside the path boundary. In Figure 3.5a, for the path $ABNEGVCD$, the line segment joining the centres $c1$ and $c2$ partially lies on or outside outside the region of the path. Therefore, for this line segment, if the point P lies on the line segment $(L1, L2)$, no rectangle is computed.

3.3.2 Placement of the Vision Circles

From $G_u(V, E)$, an axis-parallel rectangle is determined for each free trapezoid ensuring that the rectangles do not include any existing obstacles; and that the corresponding trapezoid is covered partially or completely. This method corresponds to the step 7 inside the algorithm 6 and is described in detail in the previous section 3.3.1. This method has three parts: (a) all the paths of $G_u(V, E)$ are traversed with a predefined step length, (b) the largest possible rectangles are calculated along a path using the end-points of these steps, and (c) these rectangles are cropped such that they can be fitted inside the combined boundary of all the free trapezoids in a path ensuring each rectangle containing one or more free trapezoids. A particular free trapezoid may belong to one or more paths and there can be numerous end-points inside this free trapezoid. The largest rectangle which is computed from all the end-points is kept in Map_R corresponding to the key represented by this free trapezoid. These rectangles are used for placing vision circles. An "Objective Func-

tion" based on a "Centrality Measure" and the area of a rectangle determines the suitable rectangle for inserting a vision circle.

Centrality Measure: A measure for a free trapezoid

A free trapezoid can obstruct information flow fully or partially in $G_u(V, E)$. The centrality measure, C_H measures this obstruction merit of the free trapezoid.

These are kept in a hash map with the trapezoid, t_{index} as the key and the measure as its value.

Calculation of the centrality measure for a free trapezoid is based on the edge weights(E_w) of the graph (G_u), which can be considered as the ease of information flow between the common sides (a, b in Figure 3.7) of the adjacent trapezoids of the graph $G_u(V, E)$. This information flow between the trapezoids can be treated as the "information gain" between them.

However, if the adjacent trapezoids share a partial common vertical line, it can be inferred that there is a partial flow of information between them. Due to this partial flow, the "information lost" is the difference between the union of the sides and the overlap of the sides.

The edge weights of $E(G_u)$ are the ratio of "information lost" and "information gain"

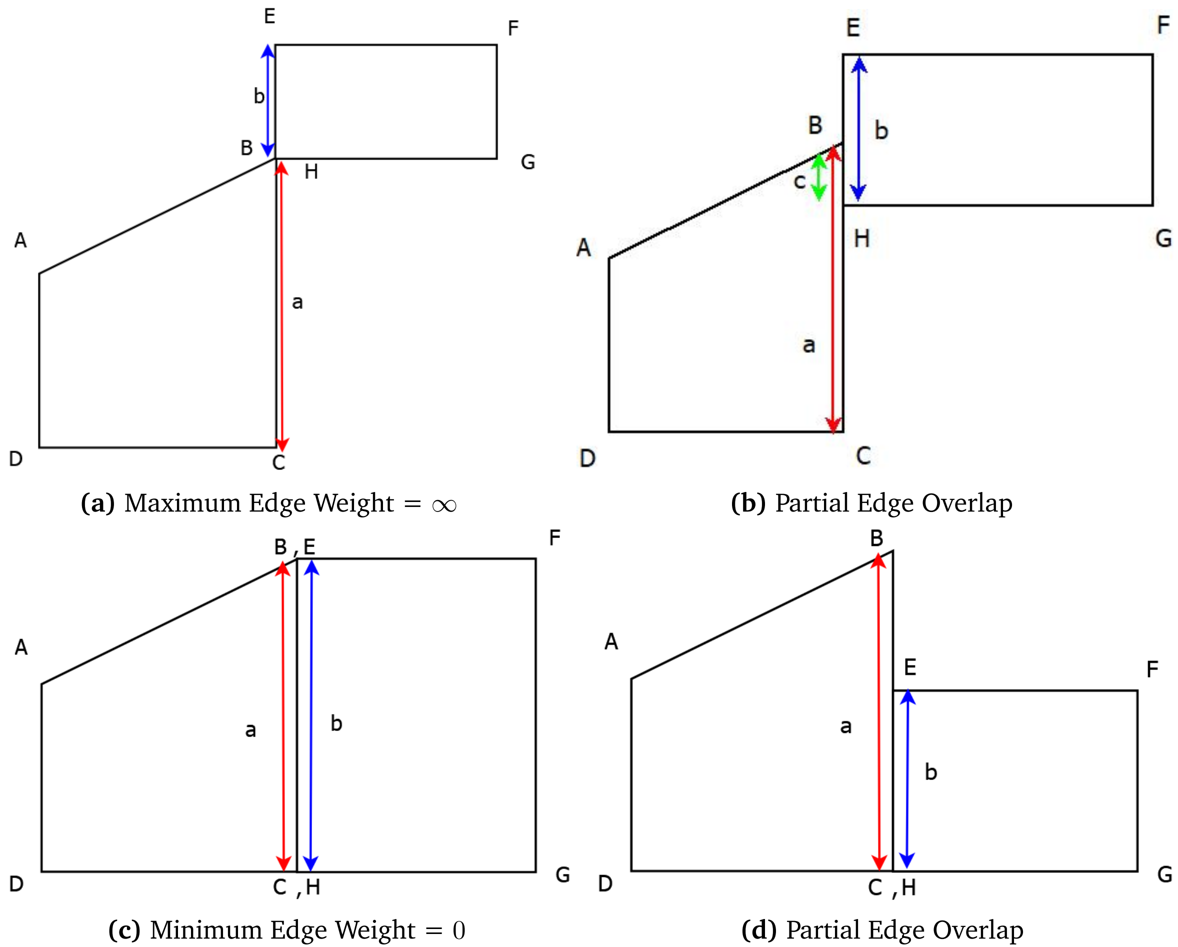
$$Edge\ Weight = \frac{Information\ lost}{Information\ gained} = \frac{Length\ of\ (a \cup b - a \cap b)}{Length\ of\ (a \cap b)} \tag{3.1}$$

The following cases arise :

Case (A): In Figure 3.7a, when the edges meet at a corner, $a \cap b = \phi$, thus, "information gain" is minimum. Again, $a \cup b = a + b$, hence "information lost" is maximum. Thus, the edge weight is maximum (∞).

Similarly, when the trapezoids are disconnected, the edge weight is maximum (∞)

Case (B): In Figure 3.7c, when edges completely overlap, $a \cap b = a \cup b = a = b$, thus, "information lost" is ϕ and "information gain" is $a (= b)$, the maximum possible value. Thus, the edge weight is minimum (ϕ).



(a) Maximum Edge Weight $= \infty$

(b) Partial Edge Overlap

(c) Minimum Edge Weight $= 0$

(d) Partial Edge Overlap

Figure 3.7: Edge Weight Cases for two trapezoids sharing an edge

Case (C): In Figure 3.7b and Figure 3.7d, the "information lost" and "information gain" are not ϕ; thus edge weights have finite values. When the overlap of the common edges reduces, "information lost" increases, and "information gain" decreases. Thus, the edge weight increases as the overlap between the common edges reduces.

Objective Function: Measure for a rectangle

The vision circle is placed in a free rectangle in which it completely fits, based on the highest score obtained from an "Objective Function". The "Objective Function" is based on the area of the rectangle (where the hash key/trapezoid corresponding to the rectangle lies) and "Centrality Measure"[44] of all the trapezoids (which are located fully or partially inside the rectangle).

The "objective function" ($CalcObjectiveFunction()$: Algorithm6: Line 16) helps to decide the rectangle in which the vision circle has to be fitted using the "Centrality

Measure" of all the trapezoids inside the rectangle and the "Area Of the Rectangle. It gives a score for every rectangle in Map_R, which is kept in the sorted list, $RectScore$. The list is searched for the rectangle that completely fits the circle i.e. the diameter of the circle is less than the minimum side of the rectangle. If no such rectangle is found, the circle is placed at the center of the rectangle with the highest score.

A simple objective function can be defined as

$$Score_R = \left(\sum_{t \in R} C_t \right) * Area(R) \tag{3.2}$$

$$where\ R\ is\ the\ rectangle,\ t\ is\ a\ trapezoid\ inside\ R\ and\ C_t\ is\ centrality\ measure\ of\ trapezoid\ t$$

Examples of some more objective functions are as follows:

1. Every rectangle in the list R covers multiple trapezoids with full or partial. Every trapezoid has its own centrality measure calculated from the undirected weighted graph. Since the trapezoids may be partially included in the rectangle, the ratio of the trapezoid's area contributed to the total area is used as a fraction to represent the fraction of centrality measure included in the rectangle. The objective function used in this case is:

$$Score(R) = \left(\sum_{t \in R} (C_t * \frac{Area(t')}{Area(t)}) \right) * \frac{Area(R)}{Area(Map)} \tag{3.3}$$

   where t' is part of the trapezoid included in the rectangle and t is the trapezoid that is included in the rectangle at least partially.

2. This objective function doesn't consider the area of the rectangle. Therefore, this function is good for maps where the rectangles are of similar area and so, the centrality differences play a bigger role in the selection of the rectangle with the maximum score. The objective function is:

$$Score(R) = \sum_{t \in R} \left(C_t * \frac{Area(t')}{Area(t)} \right) \tag{3.4}$$

3. When the number of slabs in the map gets very high and the rectangles are similar in area, the contribution of a trapezoid into the rectangle becomes sim-

ilar to other trapezoids. Therefore, for such maps, an objective function that can be used is:

$$Score(R) = \sum_{t \in R} C_t \tag{3.5}$$

4. The second objective function gives higher values for rectangles spanning over thinner and more number of slabs. This creates a bias in a map with uneven slab distribution. Therefore, a normalized form of the second objective function can be used:

$$Score(R) = \frac{\sum_{t \in R} \left(C_t * \frac{Area(t')}{Area(t)} \right)}{N(t)} \tag{3.6}$$

5. In maps where the number of slabs is very high but the rectangles are of varied areas, the ratio $\frac{Area(t')}{Area(t)}$ has little to no effect on the scoring of the rectangles but the ratio $\frac{Area(R)}{Area(Map)}$ can help give the larger rectangles a better score. Therefore, for such maps, an objective function that can be used is:

$$Score(R) = (\sum_{t \in R} C_t) * \frac{Area(R)}{Area(Map)} \tag{3.7}$$

These objective functions can again vary based on different types of Centrality measures e.g. Betweenness Centrality, Closeness Centrality, Harmonic Centrality, etc.

3.3.3 Revising the Map with the Vision Circles

After the placement of one vision circle, the succeeding vision circles are required to be placed in the remaining free space of the map. To ensure this, the vision circle, original polygon obstacles, and the free trapezoids with which the circle intersect are all combined to form a new polygon. This new polygon is now considered the new obstacle and the original polygon obstacles and free trapezoids with which the circle intersected, are replaced by this new polygon obstacle in the map. To construct the merged polygon, the following steps are performed.

Step 1: *Classify the internal angles w.r.t. the vertices of L_P as concave/convex.*

Algorithm 6 Strategic Placement of vision circles

Require: L_P : List of Polygons; Δ : Step length; $Circle$: Radius of the Circle
Ensure: Updated L_P

1: Create Planar Subdivision S from L_P
2: Mark all trapezoids by either occupied or free
3: $L_P Trap$: Hash Table where the key is a polygon in L_P and the value is a list of trapezoids that cover that polygonal obstacle.
4: $G_u(V, E)$: graph of free trapezoids for the map
5: C_H: Hash Table containing (t_{index}, C_{index}) as key-value pair where t_{index} is the trapezoid and C_{index} is its corresponding centrality measure
6: $RectScore$: Array of ordered pairs containing $(t_{index}, Score_{index})$, where $Score_{index}$ is the score of the rectangle corresponding to the trapezoid t_{index}
7: $Map_R = Algo_All_Free_Rectangles(\Delta, S)$
8: $Centre$: Array of circle centres
9: **for** $\overrightarrow{e}_{ij} \in G_u$ **do**
10: $\quad E_w[\overrightarrow{e}_{i,j}] = CalcEdgeWeight(t_i, t_j)$
11: **end for**
12: **for** $t_{index} \in G_u$ **do**
13: $\quad C_H = CalcCentrality(t_{index}, G_u)$
14: **end for**
15: **for** $t_{index} \in Map_R$ **do**
16: $\quad RectScore[t_{index}] = CalcObjectiveFunction(Map_R[t_{index}], C_H, S)$
17: **end for**
18: Sort $RectScore$ in descending order of score
19: $counter = 1$
20: **for** $Map_R[t_{index}] \in RectScore$ **do**
21: $\quad$ **if** $Centre = \phi$ **then**
22: $\quad\quad$ **if** $min(Length(Map_R[t_{index}]), Breadth(Map_R[t_{index}])) >= 2 * Circle)$ **then**
23: $\quad\quad\quad Centre = CenterofRect(Mar_R[t_{index}])$
24: $\quad\quad$ **else**
25: $\quad\quad\quad counter = counter + 1$
26: $\quad\quad$ **end if**
27: $\quad$ **end if**
28: **end for**
29: **if** $counter = N$ **then**
30: $\quad Centre = Centre_of_Rect(RectScore[1])$
31: **end if**
32: S_{LM} : The leftmost slab that the circle reaches
33: S_{RM} : The rightmost slab that the circle covers
34: $L_P \longleftarrow ReviseMapWithVisionCircle(L_P, S, Centre, Circle, S_{LM}, S_{RM}, L_P Trap)$ ▷ Updating L_P

Step 2: *Two axis-parallel lines are drawn which intersect at the center of the circle, where the top-left quadrant is the first quadrant, the top-right quadrant is the second quadrant, the bottom-right quadrant is the third quadrant and bottom-left quadrant is the fourth quadrant.*

The vertical blocks of the planar subdivision S are divided by these two axis-parallel lines such that the smaller vertical blocks can be grouped under the first, second, third, and fourth quadrants. The traversal of vertical blocks follows this order for the quadrants.

For the new vertical blocks, a new trapezoid is created due to the intersection between the existing trapezoid and the line parallel to X-axis. The new trapezoid is then the top trapezoid in the case of the first and third quadrants and becomes the bottom trapezoid in the case of the two remaining quadrants. The bottom trapezoid of the undivided vertical block (or its smaller version created due to the intersection between an existing trapezoid and Y-Axis) becomes the top and bottom trapezoid of the new vertical blocks in the case of the fourth and third quadrant respectively. Accordingly, the top trapezoid of the undivided vertical block (or new version created due to the intersection between an existing trapezoid and Y-Axis) becomes the bottom and top trapezoid of the new vertical blocks in the case of the first and second quadrant respectively.

Step 3: *Left and right intersection points between the circle and the line parallel to X-Axis are marked as P_L and P_R.*

Step 4: *Point P_L is added to the final merged polygon MP only if it belongs to a free trapezoid.*

Step 5: *The new vertical slabs are first traversed from left to right for the first and second quadrants and then from right to left for the third and fourth quadrants to find out intersection points between the circle and trapezoids under each vertical block. For each vertical block, the traversal starts from the top trapezoid to the bottom trapezoid of the vertical block. If the trapezoid belongs to the first or fourth quadrant, the intersection of the circle with the left vertical line and the top edge is determined. Similarly, for the second and third quadrants, the intersection with the right vertical line and the bottom edge is determined. This top edge or bottom edge (as applicable based on quadrant) of a trapezoid is designated as the "primary edge".*

Step 6: There are cases where the polygon surrounds the circle in such a way that an area in the free space is created for which the boundary is created by a subset of vertices of the polygon and an arc of the circle. This comes from an observation where a line segment intersects a concave polygon in such a way that it enters inside the polygon then comes out of the polygon to travel outside the polygon and then again enters into the polygon Ref diagram. If there are one or more concave vertices of the polygon on any side of this line, then this line segment along with those concave vertices and other vertices on the same side will give rise to a new bounded area [25] which is outside of the actual polygon. This bounded area will be called a 'hole' in the context of the merged polygon. If the top edge or bottom edge (as applicable based on quadrant) of a trapezoid intersects the circle and one of the vertex of that edge is concave and outside the circle, then that vertex will create a 'hole'. While constructing the merged polygon, this 'hole' will be identified but will be assumed to be part of the merged polygon. To determine the vertices of the 'hole' and to update the merged polygon while traversing through trapezoids of the vertical block, the following steps are followed.

- *If the associated polygon edge of the intersected primary edge of the trapezoid enters into the circle from the concave vertex, then the previous polygon edge that exited from the circle is found out. The target vertex of this edge is marked.*
- *Two intersected points on both ends and the polygon vertices starting from the marked vertex and ending at the concave vertex, placed in between these two points constitute the hole.*
- *The polygon vertices that are part of the hole are deleted from the polygon so that they are never included in the merged polygon.*
- *If the associated polygon edge of the intersected primary edge of the trapezoid exits from the circle, then the next polygon edge that enters into the circle is found. The source vertex of this edge is marked.*
- *Two intersected points on both ends and the polygon vertices starting from the concave vertex and ending at the marked vertex, placed in between these two points constitute the 'hole'.*

- *The polygon vertices which are part of the 'hole' is deleted from the polygon.*

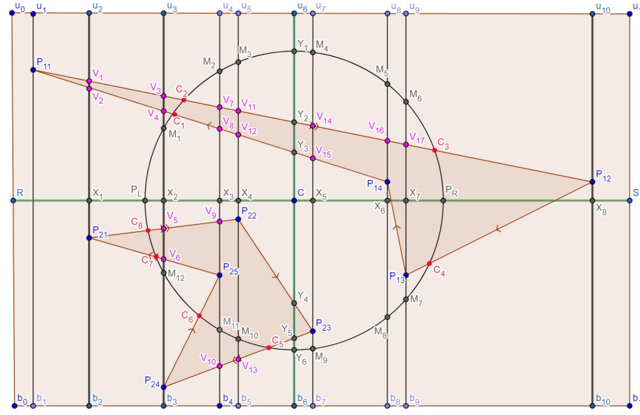
Step 7: *Both circle and vertical line intersection points for the first and second quadrants are added to MP in the order they are found. While adding a circle intersection point, if a polygon edge exists from the circle, i.e. part of the polygon protrudes from the circle, all the subsequent vertices of the polygon are included in the final list of points in the same order as they are present in the original polygon till an edge enters the circle again. These polygon vertices are added just after the intersection point with the circle.*

Step 8: *All the vertical blocks, which have been already crossed while traversing and adding the vertices of protruding polygon in MP, are skipped in the subsequent traversal of vertical blocks. Traversal of the next vertical block might start from some vertical block in the same quadrant or subsequent quadrant.*

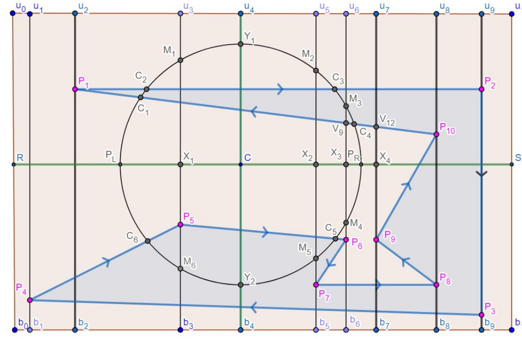
Step 9: *Point P_R is added to the final merged polygon MP only if it belongs to a free trapezoid.*

Step 10: *Circle and vertical line intersection points for the fourth and third quadrants and vertices of protruding polygon (if any) are added to MP as mentioned above.*

As an example, for the figure 3.8a, there are two polygons and their vertices are listed in a clockwise manner. P_L and P_R are left and right intersection points between axis-parallel X-axis and circle. Vertical blocks $X_1X_2u_3u_2$ and $X_7X_8u_{10}u_9$ contain P_L and P_R respectively. Top-left quadrant starts from $X_1X_2u_3u_2$ and ends at $X_4Cu_6u_5$ whereas Top-right quadrant starts from $CX_5u_7u_6$ and ends at $X_7X_8u_{10}u_9$. Similarly, the bottom-right quadrant starts from $X_7X_8b_{10}b_9$ and ends at $CX_5b_7b_6$ whereas bottom-left quadrant starts from $CX_4b_5b_6$ and ends at $X_1X_2b_3b_2$. Traversal begins from the vertical block $X_1X_2u_3u_2$ starting from trapezoid $X_1X_2V_4V_2$. Point P_L is added to the list as it lies within the free trapezoid. There is no intersection between other trapezoids in this vertical block and circle. For the next vertical block $X_2X_3u_4u_3$, traversal starts from the top trapezoid $X_2X_3V_8V_4$. Left vertical line and edge of this trapezoid intersects with the circle at M_1 and C_1 respectively and they are added to the final list. While adding C_1, as there is an outgoing edge $P_{14}P_{11}$ which created the intersection point C_1, the polygon vertices starting from vertex P_{11} in clock-wise order are added after C_1. In this case, only P_{11} is added after C_1 as the edge $P_{11}P_{12}$ enters into the circle through intersection point C_2. Next trapezoid



(a) Circle Polygon Intersection - No Hole



(b) Circle Polygon Intersection - With Hole

Figure 3.8: Merging Vision Circle With Polygon Obstacle

$V_4V_8V_7V_3$ intersects with the circle at C_2, which is added to the final polygon. Next trapezoid $V_3V_7u_4u_3$ does not contribute any point to the list, as its left vertical line and top edge do not intersect with the circle. Following the same approach other vertical blocks of the top-left quadrant add points M_2 and M_3 to the final list. For vertical block $CX_5u_7u_6$, traversal starts from the top trapezoid $u_6u_7V_{14}Y_2$ and ends at bottom trapezoid $Y_3V_{15}X_5C$. The right vertical line of trapezoid $u_6u_7V_{14}Y_2$ intersects with circle at Y_1. The rest of the trapezoid in this vertical block does not contribute any points as they do not intersect with the circle. Other vertical blocks of top-right quadrant adds M_4, M_5, M_6, C_3 and P_{12} points in the list. Following similar approach, vertical blocks of bottom-right and bottom-left quadrants add C_4, M_7, M_8, M_9, Y_6, C_5, V_{13}, V_{10}, P_{24}, C_6, M_{12}, C_7, P_{21}, C_8 points to the final list.

In the example figure 3.8b, while traversing trapezoid $X_3X_4V_{12}V_9$, it can be noticed that the polygon edge $P_{10}P_1$ associated with its primary edge $V_{12}V_9$ enters into the circle from the concave vertex P_{10} and intersects the circle at C_4. Hence vertex P_{10} creates a hole inside the merged polygon consisting of the outer boundary of the

given polygon and some points on the circle. The edge P_5P_6 is just the previous edge which exited from the circle at C_5. So starting from the target vertex P_6 of this edge, and then P_7, P_8, P_9 and P_{10} are added in between C_5 and C_4 to identify the hole. The vertices P_6, P_7, P_8, P_9 and P_{10} are deleted from the original polygon so that they never appear into the final vertices of the merged polygon.

3.4 A Heuristic Method Using Genetic Algorithm for Strategic Placement of Vision Circles In Presence Of Convex Polygons

In this section, a heuristic method using a Genetic Algorithm is proposed for the strategic placement of vision circles in the presence of convex polygons.

The problem of circle packing with overlaps in the presence of convex polygons is formally defined. The objective is to find an arrangement of circles within a given region, respecting the boundaries of convex polygons, while allowing controlled overlaps. The problem formulation includes the definition of the search space, constraints, and the objective to be optimized. The problem can be formally defined as follows:

- **Search Space:** The search space consists of all possible configurations of circles within the given region. Each configuration represents a potential arrangement of circles, where each circle is characterized by its position (coordinates), size (radius), and overlap with other circles.
- **Constraints:** The following constraints must be satisfied:
  - **Convex Polygon Boundaries:** The circles should be positioned within the boundaries of the convex polygons present in the region. The circles should not extend beyond the perimeter of the polygons.
  - **Controlled Overlaps:** Overlaps between circles are allowed but convex polygons should not overlap with each other.
- **Objective:** The objective is to optimize the packing arrangement based on a defined objective function. The objective function takes into account various factors, including:

- **Packing Density:** Maximizing the packing density, which is the proportion of the region occupied by the circles.

- **Overlap Minimization:** Minimizing the overlaps between circles while staying within the allowed limits.

- **Convex Polygon Adherence:** Ensuring that the circles adhere to the boundaries of the convex polygons and do not violate them.

- **Other Performance Metrics:** Additional metrics, such as the uniformity of circle sizes or the avoidance of small gaps, can be considered based on specific requirements

By formulating the problem in this manner, the goal is to find an optimal arrangement of circles that achieves a high packing density, minimizes overlaps, adheres to the boundaries of convex polygons, and satisfies any additional performance metrics. The problem formulation serves as the foundation for developing an effective approach using genetic algorithms to solve the circle-packing problem in the presence of convex polygons with controlled overlaps.

A suitable representation scheme is devised to encode the circle packing configurations. This representation should effectively capture the necessary information about circle positions, sizes, and overlaps while respecting the constraints imposed by convex polygons. One possible representation could be a list of circles, where each circle is defined by its position coordinates, radius, and information about its overlaps with other circles. An illustrative example is depicted in the [Figure 3.9].

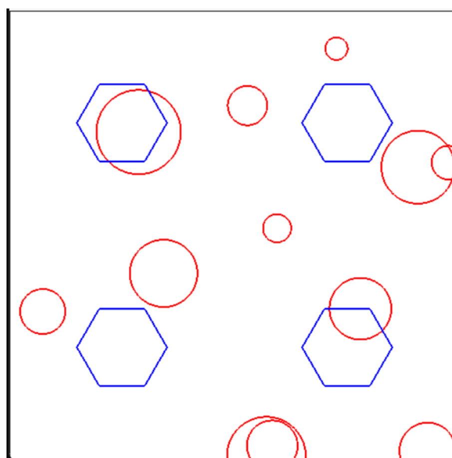


Figure 3.9: Circle Packing with overlap

3.4.1 Genetic Operators To Be Applied for Circle Packing

The main genetic operators are crossover and mutation, which when applied to a generation of population, the generations evolve over time with fitter individuals(chromosomes). Thus, these fit chromosomes have a higher probability of being selected as parents and pass on their genetic material through crossover and mutation. This iterative process drives the optimization towards finding high-quality solutions for the circle packing problem with overlaps and convex polygons.

There are different **selection strategies**, such as tournament selection, roulette wheel selection, or rank-based selection, which are used to select individuals from the population for reproduction using the "crossover" operation based on their fitness values. The selection process ensures that individuals with higher fitness, i.e., better packing configurations, have a higher chance of being selected as parents for the next generation.

In the context of circle packing with overlaps in the presence of convex polygons, different **crossover operators** can be developed to generate new offspring from selected parent individuals. These crossover operators facilitate the exchange of genetic information, such as circle positions, sizes, and overlaps, between the parents to create diverse and potentially improved solutions. Some possible crossover [59] [19] operations can be as follows:

1. **Single-Point Crossover:** This operator selects a random point along the genetic representation of the circles and exchanges the genetic material between the parents at that point. For example, the positions and sizes of circles can be swapped between the parents at the chosen crossover point.

2. **Two-Point Crossover:** Similar to single-point crossover, but instead of a single crossover point, two crossover points are randomly selected along the genetic representation. The genetic material between these two points is exchanged between the parents.

3. **Uniform Crossover:** In this operator, each gene (circle) in the offspring is randomly inherited from one of the parents. For each gene, a coin toss is performed, and based on the result, the gene is selected from either parent.

The final genetic operator is **mutation** which randomly chooses a chromosome and mutates it. The mutation is done by randomly changing the values of a cell in the chromosome, i.e. randomly changing the location of a circle or even interchanging

the location of two different sizes of circles. The chromosomes or the positions of the cells in a chromosome can be chosen using some distribution [36].

The various genetic operators generate many illegal chromosomes i.e. with the change of the locations of the circles, there might be circles overlapping fully with the polygons or multiple circles overlapping fully, etc. All such chromosomes are either eliminated from the population or are corrected by handling the overlaps with the Convex Polygons and among themselves.

3.4.2 Handling Overlaps with Convex Polygons

One of the key challenges in circle packing with overlaps in the presence of convex polygons is efficiently handling the overlaps while respecting the boundaries of the polygons. This involves two main steps: collision detection and collision resolution.

- **Collision Detection:**Collision detection techniques are employed to identify overlaps between circles and polygons in the packing configuration. Special attention is given to handling overlaps that violate the boundaries of convex polygons. Several approaches can be considered for collision detection:
  - **Bounding Box Checks:**A simple and efficient method is to use bounding boxes to approximate the shape of circles and polygons. Bounding boxes are rectangular regions that enclose the circles and polygons. By comparing the coordinates of bounding boxes, potential collisions can be quickly detected. This method provides a fast preliminary check but may result in some false positives.
  - **Geometric Intersection Tests:**More accurate collision detection can be achieved by performing geometric intersection tests between circles and polygons. Techniques such as point-in-polygon tests, line-segment intersection tests, or polygon-polygon intersection tests can be employed to determine if a circle and a polygon intersect. These tests take into account the actual shapes of circles and polygons and can accurately identify overlaps.
- **Collision Resolution:**Once overlaps are detected, collision resolution techniques are applied to adjust the positions and sizes of circles to eliminate or reduce the overlaps while ensuring that the circles adhere to the boundaries of

the convex polygons. Various methods can be considered for collision resolution:

- **Repulsion-Based Techniques:**Repulsion-based methods involve applying repulsive forces between colliding circles to push them apart and reduce overlaps. The magnitude and direction of the repulsive forces are determined based on the degree of overlap and other factors such as circle sizes and proximity. Iterative processes can be used to gradually reduce overlaps and achieve a more optimal packing configuration.

- **Local Search Approaches:**Local search methods involve iteratively exploring neighboring solutions to improve the packing arrangement. In the context of collision resolution, local search algorithms can iteratively adjust the positions and sizes of circles to minimize overlaps while satisfying the convex polygon constraints. Techniques such as gradient descent or hill climbing can be applied to find local optima.

- **Hybrid Approaches:**Hybrid approaches combine different techniques, such as repulsion-based methods and local search algorithms, to achieve better overlap resolution. These approaches leverage the strengths of multiple methods to effectively handle overlaps and optimize the packing configuration.

The choice of collision resolution technique depends on the problem requirements, computational efficiency, and the desired trade-off between packing density and adherence to convex polygon boundaries. The collision resolution process is often integrated into the genetic algorithm framework, where it is applied after crossover and mutation operations to improve the quality of the offspring and overall population.

By effectively handling overlaps with convex polygons, the proposed approach ensures that the resulting circle packing configurations maintain the required adherence to the polygon boundaries while achieving high packing density and satisfying the constraints of controlled overlaps.

3.4.3 Objective Function Formulation

The objective function plays a crucial role in guiding the genetic algorithm towards finding optimal solutions for the circle packing problem with overlaps and convex polygons. The objective function evaluates the quality of a given packing configura-

tion based on various criteria. In the context of this problem, the objective function should consider factors such as packing density, overlap minimization, and adherence to convex polygon boundaries. The formulation of the objective function involves defining appropriate weightings for each criterion and balancing their contributions.

- **Packing Density:**The packing density metric measures the proportion of the given region that is occupied by circles. A higher packing density indicates a more efficient utilization of space. The packing density can be calculated by dividing the total area covered by circles by the total area of the region.

- **Overlap Minimization:**The objective function should also consider the minimization of overlaps between circles. Overlaps should be controlled within predefined limits to ensure the feasibility of the packing configuration. The objective function can penalize excessive overlaps by assigning a higher cost or reducing the fitness value for configurations with large overlaps.

- **Convex Polygon Adherence:**Maintaining adherence to the boundaries of convex polygons is another critical aspect. The objective function can include a term that rewards configurations that closely follow the polygon boundaries. This can be achieved by measuring the distances between circle centers and polygon edges and encouraging configurations where these distances are minimal.

- **Additional Performance Metrics:**Depending on the specific requirements of the problem, additional performance metrics can be included in the objective function. For example, the uniformity of circle sizes or the avoidance of small gaps between circles can be considered. These metrics can be weighted accordingly based on their importance.

The objective function formulation involves assigning appropriate weights to each criterion to reflect their relative importance. The weights can be determined through experimentation or by considering the problem's specific requirements. Balancing the weights ensures that the genetic algorithm searches for solutions that optimize all aspects of the problem simultaneously. The objective function guides the genetic algorithm in evaluating the fitness of candidate solutions and selecting individuals for reproduction, crossover, and mutation. By iteratively optimizing the objective function through the genetic algorithm, the approach aims to discover circle packing configurations that achieve high packing density, minimize overlaps, adhere to

convex polygon boundaries, and potentially satisfy additional performance metrics, leading to efficient and visually pleasing solutions for the circle packing problem with overlaps in the presence of convex polygons.

3.4.4 Implementation of Genetic Algorithm

The genetic algorithm components, including population initialization, fitness evaluation, selection, crossover, and mutation operators, are implemented based on the chosen programming language and genetic algorithm framework. The implementation involves the following steps:

- **Population Initialization:**The initial population of circle packing configurations is generated randomly or using specific initialization strategies. This step ensures a diverse set of initial solutions.
- **Fitness Evaluation:**Each individual in the population is evaluated using the objective function. The objective function assigns a fitness value to measure the quality of the packing configuration. The fitness value indicates how well the configuration satisfies the criteria of packing density, overlap minimization, and convex polygon adherence.
- **Selection:** Selection operators, such as tournament selection, roulette wheel selection, or rank-based selection, are implemented to select individuals for reproduction based on their fitness values. Individuals with higher fitness have a higher chance of being selected as parents for the next generation.
- **Crossover:**The chosen crossover operators, such as single-point crossover, two-point crossover, uniform crossover, or heuristic crossover, are applied to selected parent individuals to generate offspring. The genetic material, including circle positions, sizes, and overlaps, is exchanged between parents to create new configurations.
- **Mutation:**Mutation operators are applied to introduce small random changes to the genetic material of individuals. This introduces diversity into the population and helps explore new regions of the search space. Mutation operators can adjust circle positions, sizes, and overlaps, ensuring that the resulting configurations remain feasible.
- **Termination Criteria:**Termination criteria, such as a maximum number of generations or a convergence threshold, are defined to determine when the genetic

algorithm should stop iterating and return the best-found solution.

Below is the steps for the proposed genetic algorithm for circle packing with overlaps in the presence of convex polygons:

Algorithm 7 Genetic Algorithm for Circle Packing

1: Initialize the population with candidate solutions (chromosomes) by generating positions and radius of circles such that no circle completely falls inside a polygon or existing circle(s)
2: Evaluate the fitness of each chromosome
3: **while** termination condition is not met **do**
4: Select **Parents** for **reproduction**
5: Perform **Crossover** to generate offspring taking care that no circle completely falls inside a polygon or existing circle(s). If such an offspring exists, it is discarded.
6: Perform random **Mutation** on the offspring
7: Evaluate the fitness of the offspring
8: Select individuals for the next generation
9: Update the population with the selected individuals
10: **end while**
return The best chromosome as the optimized solution

By implementing the genetic algorithm components, collision detection, and resolution techniques, and defining an appropriate experimental setup, the proposed approach can be evaluated and compared to existing methods in terms of its performance, effectiveness, and efficiency in solving the circle packing problem with overlaps in the presence of convex polygons.

3.4.5 Experimental Results

Two experiments were conducted to position circles within two distinct maps featuring square boundaries measuring a thousand units, each containing convex polygons. In the first map, three rows of convex polygons were arranged, while in the second map, six rows of convex polygons were situated within the boundary. The first map has three columns and the second map has six columns.

In the first map, every row consisted of one triangle, one rectangle, and one pentagon. Similarly, in the second map, each row comprised one triangle, one rectangle, one pentagon, followed by one triangle, one rectangle, and one pentagon in sequential order.

In both maps, starting from the initial row, the polygons within each row were shifted to the right before being placed in the subsequent row. For instance, if the first row featured one triangle, one square, and one pentagon in that order, the next row would consist of one square, one pentagon, and one triangle in the given sequence, and so on until the final row. All polygons within a row shared the same height, and the gap between any two consecutive rows remained consistent. The radius of the circumscribed circle for each polygon is set to one-twelfth of the side length of the square boundary. The horizontal distance between the centers of two consecutive polygons in the same row is equal to the boundary side length divided by the number of columns. The horizontal distance from the vertical boundary to the center of the nearest polygon is two-thirds of the distance between consecutive polygons in the same row. Similarly, the vertical distance between the centers of two polygons in the same column is the boundary side length divided by the number of columns, and the vertical distance from the horizontal boundary to the center of the nearest polygon is two-thirds of the distance between consecutive polygons in the same column.

For these experiments, the genetic algorithm chooses the circles from a basket of five circles where the radius of the circles is in the Fibonacci series starting with three units. For both experiments, the number of circles (i.e. the chromosome size) to be placed was predetermined by dividing half of the area of the map boundary by the area of the largest circle in the basket. For all experiments, the algorithm was executed using three distinct population sizes: 100, 150, and 200, representing the number of potential arrangements of circles within the map. In each experiment, the algorithm was run for ten different numbers of generations, ranging from 20 to 200, increasing the value by 20 for each iteration.

An experimental result displaying the optimal placement of circles obtained from an experiment is illustrated in Figure 3.10. In the map corresponding to this experiment, there are three rows of polygons (green-colored), with each row consisting of one triangle, one rectangle, and one pentagon. Circles (blue-colored) were positioned in the vacant spaces within the map, ensuring they do not overlap with the existing polygons.

Three graphs were plotted for each experiment, corresponding to population sizes of 100, 150, and 200, illustrating the relationship between free space area and number of generations.

- Figures 3.11a, 3.11b, and 3.11c illustrate graphs depicting the relationship between free space area and the number of generations for three distinct pop-

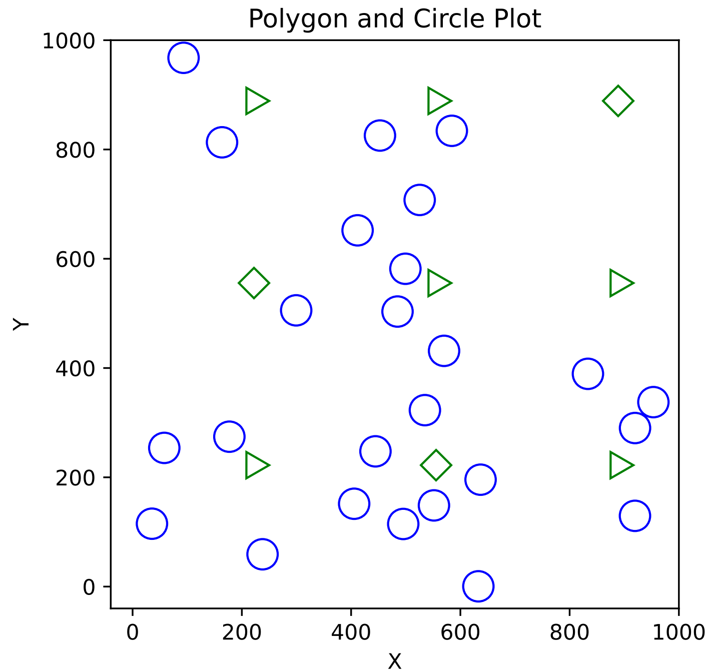


Figure 3.10: Circles placed in a map

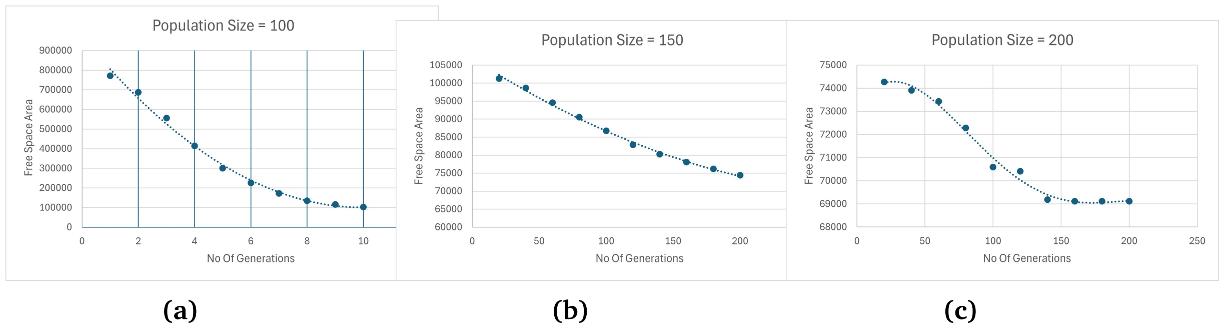
ulation sizes when the algorithm was executed on the first map during the first experiment.

- Figures 3.12a, 3.12b, and 3.12c illustrate graphs when the algorithm was executed on the second map during the second experiment.

It is evident that as the population size increases, the coverage of free space also increases, resulting in a decrease in the area of free space. Likewise, with a fixed population size, an increase in the number of generations leads to a reduction in the area of free space before reaching a stabilized state.

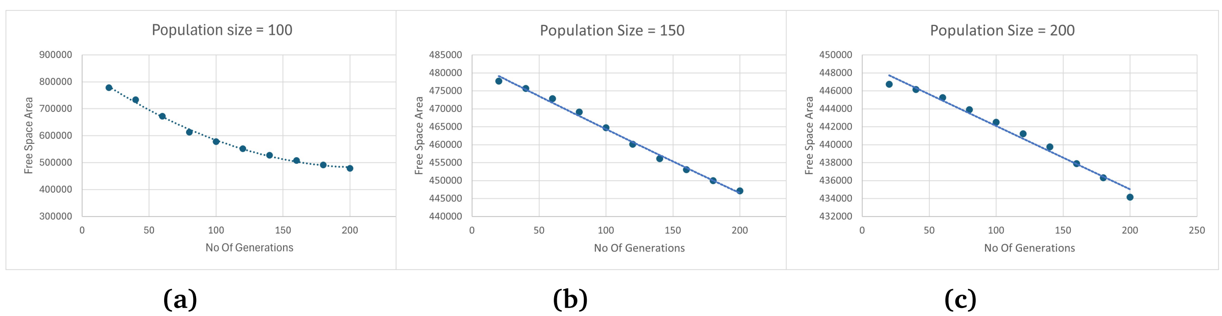
3.5 Complexity Analysis Of Proposed Discrete Method For Placement Of Circles

To analyze the complexity of this proposed method, the complexity of the rectangular finding algorithm to construct a hash table Map_R is examined as the first step in the below sub-section 3.5.1.



(a) **(b)** **(c)**

Figure 3.11: (a) Map 1: Population Size 100, (b) Map 1: Population Size 150, (c) Map 1: Population Size 200



(a) **(b)** **(c)**

Figure 3.12: (a) Map 2: Population Size 100, (b) Map 2: Population Size 150, (c) Map 2: Population Size 200

3.5.1 Complexity Of algorithm to construct Map_R

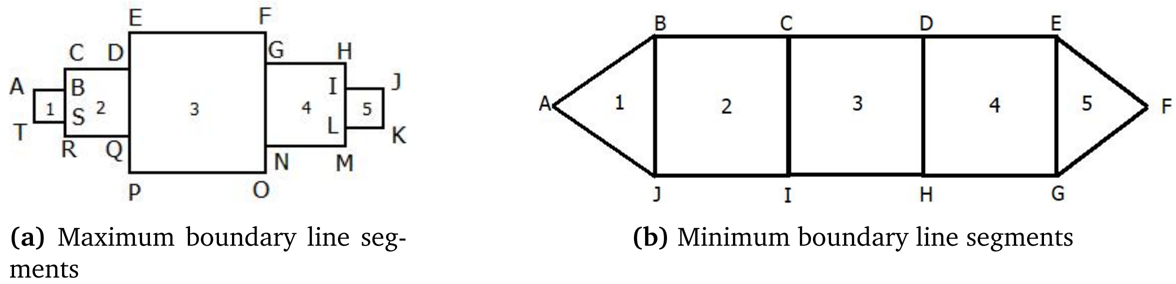
Algorithm 5 computes the largest rectangle corresponding to every trapezoid in a path of the map. The computed rectangles contain its corresponding trapezoids at least partially. Algorithm 5 is invoked for every path between the set of starting vertices and ending vertices.

In the remaining part of this sub-section, an analysis of the proposed algorithm based on the complexities of different steps is given.

Maximum and Minimum number of boundary line segments for n continuous adjacent trapezoids

For every trapezoid, its top and bottom sides always belong to the boundary of the chain of adjacent trapezoids having connected vertical sides. If there are n adjacent trapezoids in a path, then there are $2n$ top and bottom line segments covering the boundary of the contiguous adjacent trapezoids for that path.

In Figure 3.13a, for the adjacent trapezoids $EFOP$ and $GHMN$, line segment FG connects the top lines EF and GH; and NO connects the bottom lines PO and NM.



(a) Maximum boundary line segments

(b) Minimum boundary line segments

Figure 3.13: The maximum and minimum cases of the number of boundary line segments

Thus, at most two line segments are required to join the top points and the bottom points of the two connected trapezoids.

Since there are $n - 1$ connections, there can be at most $2(n - 1)$ line segments required to connect the top and bottom points. Conversely, the minimum number of line segments required can be 0 when the top points and bottom points overlap for every connection.

For the left and right Y-parallel boundary of the adjacent trapezoids, at most two line segments is required for the boundary. However, if the extreme left and right trapezoids of the path are triangles, no line segments are required. Thus, the maximum number of line segments required for the Y-parallel boundary is $2n+2(n-1)+2 = 4n$. The minimum number of line segments required is $2n+0+0 = 2n$ line segments. In Figure 3.13a, there are 5 trapezoids and it can be seen that 20 line segments are required for the boundary. In Figure 3.13b, there are 5 trapezoids and 10 line segments are required for the boundary.

Analysis of $IntSctn()$ and $LRIntSCtn()$ Intersection Functions

In Algorithm 5, the function $IntSctn()$ computes the tip and bip using point P belonging to line segment l_i, which lies in t_i or t_{i+1}. Since the trapezoid is known, the two points of intersection can be easily computed in $\mathcal{O}(1)$ time.

The maximum number of boundary line segments possible for n contiguous trapezoids is $4n$ where $LRIntSctn()$ can intersect every line. Therefore, $LRIntSctn()$ takes $\mathcal{O}(n)$ time to compute lip and rip.

Analysis of $Crop()$ Function

The $Crop$ function checks all trapezoids in the path T_L i.e. n trapezoids. Therefore, the $Crop()$ function also takes $\mathcal{O}(n)$ time to fit the rectangle inside the boundary region.

The cropping method ensures that the point P (from which the rectangle TR is created) never falls outside the new rectangle NR after cropping. Since P lies inside every trapezoid at least once, every trapezoid has at least one rectangle that contains the trapezoid even if partially.

Complexity for construction of Map_R

Since $IntSctn()$ takes $\mathcal{O}(1)$ time, and both $LRIntSctn()$ and $Crop()$ take $\mathcal{O}(n)$ time, the total complexity for computing the largest rectangle w.r.t. a point P is $\mathcal{O}(n)$.

As the number of adjacent trapezoids in the path T_L is n, there are $n-1$ line segments. Since the above functions are called for every line segment, the function takes $\mathcal{O}(n^2)$ time.

For a given map and a given value of Δ, the number of rectangles that the method computes while moving between (c_i, c_{i+1}) is given by : $H = 1 + \left\lfloor \frac{|l_i|}{\Delta} \right\rfloor$, where $l_i \in L$ and $|l_i|$ is the length of the line segment. As point P shifts H times, Algorithm 5 takes $\mathcal{O}(H * n^2)$ time to compute the largest rectangle for every trapezoid in the path. The largest value of H possible is: $H_{max} = 1 + \left\lfloor \frac{|l_i|_{max}}{\Delta} \right\rfloor$ where $|l_i|_{max}$ is the length of the longest line segment in L. When $\Delta > |l_i|_{max}$, H_{min} reduces to 1 for all line segments in the path. For a given map, the values of H are known and vary between $H_{max}\ \&\ H_{min}$, it can be said that $RectsForaPath()$ takes $\mathcal{O}(n^2)$ time.

The method finds the optimal list of the largest rectangles in the path seen from a trapezoid when Δ becomes very small. Since the function returns a different list of rectangles for every Δ, it is difficult to say that the list of rectangles for $\Delta - \epsilon$ contains rectangles that are larger than the former, even though the number of rectangles increases for $\Delta - \epsilon$. The optimal set of rectangles may exist for even a large value of Δ but it is very difficult to know the optimal value.

Algorithm 5 is called for every path in the graph. If there are k paths, the total complexity for the construction of Map_R for all of these paths is $\mathcal{O}(kn^2)$.

If the total number of vertices in the map is N_v, then there would be $N_v + 1$ number

of vertical slabs and in the worst case, each edge can intersect a maximum $N_v - 1$ vertical slab giving birth to $N_v - 1$ new trapezoids. So the maximum number of trapezoids can be represented as $O(N_v^2)$. If a trapezoid is part of only one path and the paths cover all of the trapezoids in the map, then n can be considered as $O(N_v)$. So the time complexity to create the hash table Map_R can be taken as $O(N_v^2)$.

3.5.2 Complexity Of algorithm to place circles in the map

While finding the appropriate rectangle to place the vision circle, every entry of Map_R is evaluated, So the time complexity of the whole process for locating the suitable rectangle is $O(N_v^2)$.

To determine the time complexity for updating the map after placing the vision circle, the following observations are considered:

- Across all the four quadrants, in total $O(N_v^2)$ number of trapezoids are covered to decide the intersection of the circle with the left/right vertical line and the top/bottom edge for each trapezoid.
- All the N_v internal angles w.r.t. the vertices, $v_i \in L_P$ are classified as convex/concave.
- A "hole" is constructed between a circle and a polygon w.r.t. the concave vertices by traversing the N_v vertices of the polygon. This results in a time complexity of $O(N_v)$.
- The maximum number of original polygon vertices, $O(N_v)$ outside the circle is added to the final merged polygon, irrespective of whether a "hole" is constructed after a circle is placed.

Thus, adding up the complexities of all the points discussed above, time and space complexity to revise the map is found to be $O(N_v^2)$.

$G_u(V, E)$ graph is constructed from free trapezoids, so no of nodes of this graph is $O(N_v^2)$. As the edge weights are calculated for each node in $G_u(V, E)$, the time complexity for the edge weight calculation process is $O(N_v^2)$. Similarly, the time complexity of the centrality measure calculation process to calculate scores of free rectangular spaces is also $O(N_v^2)$ as this calculation is done for each node in graph $G_u(V, E)$. Time to construct hash tables $L_P Trap$ is shown to be $\mathcal{O}(kN_v^2)$ as shown in [5], where k is the number of paths in graph $G_u(V, E)$. Both Map_R and C_H contain

entries for each free trapezoid, so the process for calculating scores for corresponding rectangles takes $O(N_v^2)$ time.

Combining the running time of Map_R creation, graph creation, edge weight calculation, scoring of the free rectangular spaces and map revision, the time complexity of the complete circle placement algorithm is found to be $\mathcal{O}(kN_v^2)$. Space complexity for the algorithm is $O(N_v^2)$ as space required for each of $G_u(V, E)$, $L_P Trap$, Map_R and $RectScore$ is $O(N_v^2)$.

3.5.3 Review And Analysis Of Some Related Algorithms

Chazelle and Lee [13] deal with solving the problem of the covering set of weighted points instead of the free space by a circular disk. Dickerson and Scharstien [20] indirectly relate to the work of Chazelle and Lee [13], but it deals with the coverage of the maximum number of points by a polygon which is also different from current work.

The work by Daniels *et al.* [18], optimizes the largest rectangle problem in an n-vertex general polygon. The work only considers both axis-parallel and arbitrarily oriented rectangles in general polygons with non-degenerate holes. Mukhopadhyay and Rao[48] have proposed an $\mathcal{O}(n^3)$ algorithm for finding the largest arbitrarily oriented empty rectangle bounded by a given set of points.

The method proposed by Nandy *et al.* [49] to locate largest *maximal empty rectangle* (MER) among a set of arbitrary polygonal obstacles needs $\mathcal{O}(n \log^2 n)$ time.

3.6 Conclusion

In this chapter, it has been shown that the placement of circles in the free space of an environment with convex and concave polygons is an NP-complete problem. Keeping in mind the complexity class of the problem, two methods are proposed, one is a discrete method and the other one is a heuristic method using a genetic algorithm.

Finding a deterministic solution for this problem is not possible, so the proposed discrete method is an approximate solution that uses a graph constructed from the free trapezoids of the planar subdivision of the map. Once a vision circle is placed, it is treated as a polygon and merged with existing polygons so that in the next iteration, another vision circle can be placed in the remaining uncovered area. This

solution takes quadratic time and space to place a vision circle and merge it with existing polygons and already-covered areas. An interesting extension of this approach would be to use these vision circles to save the autonomous robots during their movement in an environment infested with enemy agents. A centralized system would take data from the 360^0 surveillance cameras placed in the centers of these vision circles and notify the autonomous robots regarding the movement of enemy agents inside the vision circles so that these robots can plan their motion while avoiding damage from the enemy agents. Another extension of the proposed discrete method would be experimenting with a greedy algorithm that uses a suitable heuristic function to determine the location of the vision circle placement.

The proposed heuristic method solves the challenging problem of circle packing with overlaps in the presence of convex polygons using genetic algorithms. The main objective is to develop a methodology that allows for controlled overlaps while respecting the boundaries of the convex polygons within a given rectangle.

The experimental results showcased the effectiveness of the genetic algorithm in achieving optimized layouts with controlled overlaps. The methodology was able to generate layouts that maximize the utilization of the available space while adhering to the constraints imposed by the convex polygons.

In this chapter, methods have been proposed to place the vision circles such that the adversary robots can be observed locally and their movements can be predicted. In the next chapter, two methods are proposed for predicting the future behaviors of an adversary robot, which can be used in a place where internet or GPS connectivity is not available. The proposed methods can be deployed in systems with low computing power.

Chapter 4

Prediction of Behaviour of a Non-communicative Robot

One of the challenges in robotics is predicting the expected path in which a robot moves randomly. There is a lot of literature available for motion planning and path prediction [9] [37], where the path prediction is computed using the sensor data and communication among themselves. The computation of the predicted paths of these robots is commonly done using large computing devices or in the cloud. However, the problem becomes more interesting and challenging in locations lacking GPS or internet connectivity like war and combat zones, or in remote surveillance areas. If a robot is of adversary nature and non-communicative, then predicting its future movements by another surveying robot depends solely on its recent movements, as no internal information or state changes of the adversary robot over a long period of time is available.

As the surveying robots have to be unseen or concealed, they have to be primarily small in size, thus devoid of high computational power. The purpose of these deployed robots is to activate their weaponry or set an alarm off when an unknown robot enters its periphery of observation. In this chapter, the problem of predicting the future movements of a non-communicative robot is analyzed and two simple solutions suitable for running with low computational power are proposed.

This problem is analogically similar to the prediction of movements [46] of a car. As all robots are built to perform a task, they do not move entirely in a random way. Even though the movements of the robots appear random, the paths of these robots can be predicted only by observing their motion. These random movements can be

referred to as the "*behavior*" of the robot. Thus, the "*behavior*" can be defined as *how one behaves in response to a particular situation or stimulus*. As an example, suppose the primary objective of an observing camera having a bird's eye view is to look out for potential threats. The "adversary robot" moves by taking random steps every time but it still has a pattern/behavior for its movement (as it has a purpose). The task is then to predict the future steps of the "adversary robot". These future steps when combined into a sequence will aid in giving a "predicted path".

Path prediction is applied to prevent collisions for an autonomous car in the presence of obstacles, track adversaries in a chase, and find a safe path in an area where mobile adversaries are patrolling.

The proposed system observes the movement of an adversary robot within a "vision circle" and does not have access to movement data outside the "vision circle" over long duration of time. An adversary robot can move in close loops within a "vision circle", before going outside of this. The algorithm to predict future movements of adversary robot needs to depend more on patterns of movements within a boundary rather than time-series location data. As described before, these patterns define the "*behavior*" of an adversary robot and can be expressed as "*state*" of the robot. So, state-based algorithms are more suitable for predicting the future movements of an adversary robot within a "vision circle" contrary to time-dependent algorithms.

In the previous chapter, methods have been proposed to place the "vision circles" such that the adversary robots can be observed locally within a "vision circle" by an autonomous non-communicating robot and their movements can be predicted. This chapter proposes a list and a graph-based method to predict the future behaviour of an adversary robot/agent. The proposed algorithms are time-independent and state based. These algorithms are also suitable for robots with low computing power, which do not get any support from GPS or an internet connection.

4.1 Literature Survey

In this section, a literature survey related to the path and behavior prediction of robots/humans is discussed. There are several approaches for path prediction including Markov-based models [24] [33] [52], particle filters models [68] and machine learning models [31] [55] [23]. Galata et.al. [24] described planning based on prediction for low-cost paths with obstructions having information of the environment in which the planning is required. Kim et.al. [34] proposed a trajectory

model for a sparse set of vector sequences using Gaussian Process Regression, which allows for incrementally predicting possible paths and detecting anomalous events from online trajectories. This representation also supports the matching of complex motions with acceleration changes within a trajectory. Namhoon Lee et. al. proposed [40] a Deep Stochastic IOC(inverse optimal control) RNN(Recurrent Neural Network) Encoder-decoder framework for future predictions of multiple interacting agents in dynamic scenes. Adam Houenou et al. [30] proposed a trajectory prediction method based on the Hidden Markov model which combines the yaw rate and acceleration.

Human behavior motivates the use of more sophisticated motion models for people tracking especially since humans frequently undergo lengthy occlusion events. As shown by Belhadi et.al. [6], abnormal behaviors of human beings can be interpreted as finding outliers in a crowd. They proposed algorithms for studying the anomaly behavior of a crowd based on the study of a single anomaly in the crowd. Mathias Luber [42] integrated a model based on social force into a multi-hypothesis target tracker. They demonstrated that the refined motion predictions translated into a more robust tracking behavior. Video games have also played a role in understanding behaviors. Reinforcement learning techniques have been used in playing video games. Rashid et.al. [55] learn the cooperative behavior of StarCraft II by initially using centralized training and then executing it in a decentralized fashion. Their exhibited results excelled the state-of-the-art multi-agent deep reinforcement learning methods. Generative Adversarial Networks combined with sequence prediction [26] models can observe motion histories and predict future behavior. It is possible to predict socially plausible futures by training adversaries against a recurrent discriminator and encouraging diverse predictions with a novel variety loss. Parker et.al. [50] proposed an approximate approach to observe the movements of multiple targets in a bounded area by a cooperative team of autonomous sensor-based robots. [17] presented a system that observes spatio-temporal activity of humans participating in playground games and then learns the relationship between these players, their goals, intentions, and the rules of the games. Past trajectories of obstacles in the form of time series data have been analyzed using the LSTM-based technique by [65] for the navigation of a robot in a dynamic environment. Masaya et.al. [32] proposed a path planning method that combines rapidly exploring random tree (RRT) and long short-term memory (LSTM) network. Mezair et. al. [45] proposed deep learning model-based algorithms to analyze the behavior of interconnected vehicles from information collected from the different sensors of the interconnected vehicles.

In most of the above literature like [50],[17],[26], the authors have used “time” as one of the factors for predicting the path. Long Short-term Memory(LSTM) based deep learning techniques [65],[32] memorize the time series data of past trajectories and effectively predict future movements but these are extremely costly in terms of processing and memory power.

The following sections study the “behavior” of adversary robots and apply it further to predict their path. In nature, one’s behavior is unique under normal conditions, which do not vary much randomly. Thus, one can predict the future course of action by studying its behavior. In this chapter, the states and path of the adversary robot are predicted from observations of a concealed robot, unlike the observation robots in [66], which are quite large and cannot be easily concealed. The proposed method deals with two novel approaches: one based on a list and the other based on a graph(known as a “Behavioural Network”) to analyze the “behavior” of autonomous non-communicating robots. The problems have been solved by analyzing the visual behavior of the adversary robots. The proposed approaches are not dependent on time and do not demand high computational power but are suitable for devices with low computing and memory requirements like Arduino or Raspberry Pi.

This chapter has been organized in the following way: Section 4.2 describes the theoretical background of the proposed work; Section 4.3 and Section 4.4 narrates the List-based and Graph-based behavior analysis for predicting the sequence of steps of 2-DOF wheeled robot along with space and time complexity; Section 4.5 analyses the experimental results of these two approaches in a simulated environment for the movement of a robot in a path with and without self-loops.

4.2 Theoretical Basis

In this section, a preliminary background of the proposed work is given. A robot is perceived to move with a continuous motion. However, this continuous motion can be discretized into a sequence of “states”, where each “state” is represented as a “step”. The *behavior* of the robot is defined as a “sequence of states”, where each state is a step over here.

A robot’s movement is restricted to its “Degree of Freedom” (DOF) which is defined as the possible number of independent variables affecting the states of the system, in particular any of the directions in which independent motion can occur. Thus, a simple wheeled robot has 2-DOF with a translation (front/back) and turning rota-

tional (left/right) motions; while an aerial robot that has rotation, translation, and rolling movements (yee/yaw) has 6-DOF. When a wheeled robot moves, it moves with a translation motion. When it wants to rotate, the robot first "stops" and then rotates. When the robot rotates, it either moves along a hinge or the wheels rotate in opposite directions. Thus, if a "state" is defined with a *rotation of* θ and *translation of* R, then a sequence of the "states" forms the "motion" or the *behavior* of the robot. A "state" thus can be represented as $[C_1, C_2] \equiv (\theta, (R, d))$, where R is multiples of the unit step length (R_{Gap}) in a translation movement; d is the linear direction of the translation movement; and θ represent a "turn angle" of the robot [Figure 4.1a], which is a multiple of one unit of a rotational angle (θ_{Gap}).

For practical purposes, one can consider the R_{Gap} and θ_{Gap} to be a fraction of the wheel circumference, R_{Max} and rotation, θ_{Max} respectively. Thus, the maximum number of states for translation will be $(R_{Gap}, \pm 1), (2 \times R_{Gap}, \pm 1), \ldots, (N \times R_{Gap} = R_{Max}, \pm 1)$. It will be similar for the rotational motion also.

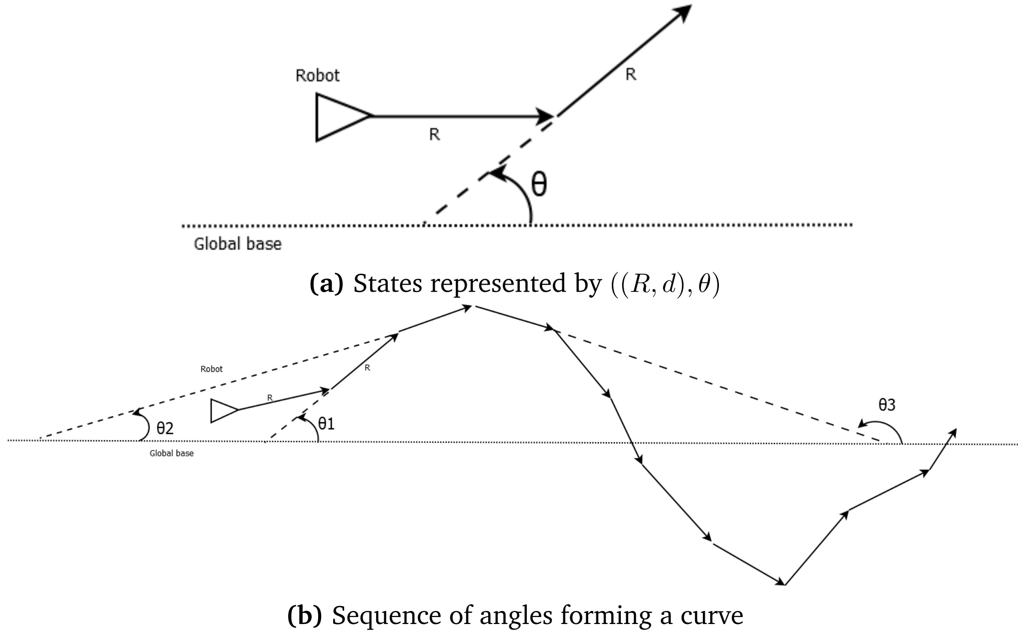
The direction d can take the values of reverse($R/-1$), forward($F/+1$), or pause($S/0$) for a wheeled robot when it moves in the reverse direction, forward direction, or is paused respectively. In the following work, it is assumed that the robot does not halt i.e. the values of $\theta, (R, d)$ will not be $(0, (0, 0))$ at any moment except at the beginning when it starts moving. The robot will only pause when it decides to take a turn with an angle θ. The values of R and d will be 0 when the robot has rotational movement. Similarly, when there is translation movement, the value of θ will be zero, whereas R will have a non-zero value and d will be either $+1$ or -1. Thus, $C_1 = \theta$ and $C_2 = (R, d)$ are mutually exclusive while the robot is in motion.

Robustness of the System

However, for practical systems, there can arise "slips" in the motion of the robot i.e. $s_{Obs}((R_{Obs} \pm \delta_R, d), \theta_{Obs} \pm \delta_\theta) \notin (\theta_{Obs}, (R, d))$. If the minimum difference between any two R_i and R_j values is R_{Gap} and between any two θ_i and θ_j values is θ_{Gap}, it is assumed that $\delta_R < R_{Gap}$ and $\theta_R < \theta_{Gap}$.

In such cases, the following modifications can be considered: $R'_{Obs} = R_{Obs}$ if $R_{Obs} + \delta_R < R_{Obs} + \frac{R_{Gap}}{2}$; $R'_{Obs} = R_{Obs} + R_{Gap}$ if $R_{Obs} + \delta_R \geq R_{Obs} + \frac{R_{Gap}}{2}$; $R'_{Obs} = R_{Obs}$ if $R_{Obs} - \delta_R \geq R_{Obs} - \frac{R_{Gap}}{2}$; and $R'_{Obs} = R_{Obs} - R_{Gap}$ if $R_{Obs} - \delta_R < R_{Obs} - \frac{R_{Gap}}{2}$.

This is similar for $\theta_{Obs} \pm \delta_\theta$, i.e. if there exists slips, δ_θ for the rotation DOF.



(a) States represented by $((R, d), \theta)$

(b) Sequence of angles forming a curve

Figure 4.1: Representation of movement

In this work, it is assumed that the robots move in an environment containing convex polygonal-shaped obstacles. There is an observer who observes the location of the robot moving in the environment. The observer is a static low computing robot, inconspicuous to the environment or to the opponent trying to predict the future steps of the adversary robot based on its observed "states" at fixed intervals. Within this interval, it is assumed that only a single event will take place; the events being a rotation or a translation. This series of events can be represented as a vector, $[C_1, C_2]$. This set of vectors can be easily converted into location coordinates by following the adversary robot's position from the first coordinate it was at.

4.3 List Based Approach for the Behaviour Analysis

In this section, a method is proposed for predicting the states of a robot moving in the 2D-Euclidean domain having 2 DOFs, namely translation and rotation motion. The observations of the two DOF parameters are maintained in two separate lists. The angular DOF is maintained as $C_1(\theta) = (\theta_1, \theta_2, \ldots, \theta_n)$, where θ_i is the turning angle of the robot, which can be both positive or negative representing a left or right turn respectively. The translation DOF is maintained as $C_2(R, d) = ((R_1, d_1), (R_2, d_2), (R_3, d_3), \ldots, (R_n, d_n))$, where R_i is the observed distance traversed

and $d_i \in \{-1, 0, +1\}$. A queue C_{Obs} having length $|C_1| + |C_2|$ maintains the observations of both translation and rotational movements. Every new observation is inserted at the front of the queue and when the queue becomes full, entries are deleted from the back to make space for the new entries. This queue C_{Obs} gives a zoomed-in snapshot of the movements, and thus a subset of the states of the robot.

Adding more parameters for DOFs allows one to gain more insight into the behavior of the robot. It can be noted here that the parameters which are considered for behavior analysis be not *dependent parameters*, i.e. the value of R is not a function of θ or vice versa.

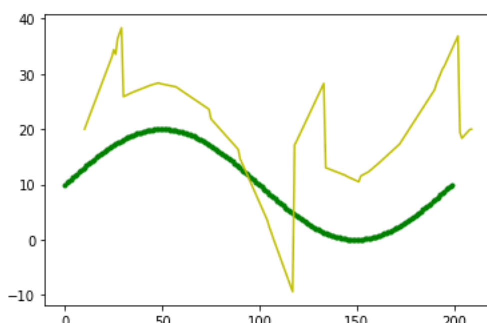


Figure 4.2: Initial curve (green line) and updated curve (yellow line)

Initially, the $C_1(\theta), C_2(R, d)$ lists are initialized with values representing a premeditated "behavior" of the robot. For example, if the premeditated behavior of the robot is that of a motion of a circle/sine curve [Figure 4.2, Green Line] with 1.8^o "turning radius" and 1 unit "translation motion", the lists will each have $n = \frac{360}{1.8} = 200$ elements. The robot moves only in the *forward direction* F.

As the robot starts to move, the first n observations are replaced in the list of the "states". The lists of C_1 and C_2 are updated separately using Algorithm 8 and Algorithm 9] respectively, which, when combined gives us the actual behavior of the robot.

4.3.1 Behaviour Analysis With List $C_1(\theta)$

It is initially assumed that the robot has a pre-defined behavior which is represented as a list $C_1(\theta) = (\theta_1, \theta_2, \ldots, \theta_n)$ where $0 < \theta_i < 360$ is the angle which the robot rotates w.r.t. the observer.

The observed θ is recorded periodically and the list, C_1 is updated using [Algorithm 8]. This iteration is repeated till a threshold $\lambda \geq 0$ is achieved. After updating, the

new list $C_1(\theta)$ is used to predict the future values of θ.

Algorithm 8 Update $C_1(\theta)$

1: Initialize $C_1(\theta)$ with pre-meditated θ
2: $C_1^0 = C_1$
3: **repeat**
4: $C_1^{-1} = C_1^0$
5: At interval t, observe θ_{Obs}
6: Error Vector $C_1^{Err}(i)$:

$$C_1^{Err}(i) \leftarrow \{C^{Err}(\theta_i^{Err}) \mid \theta_i^{Err} = |\theta_{Obs} - \theta_i^{-1}|, \quad 1 \leq i \leq |C_1|, \quad \theta_i^{-1} \in C_1^{-1}\}$$

7: $C_1^0(\theta_j) \leftarrow \theta_{Obs} \mid j = \min_{\forall i} \theta_i^{Err} \in C^{Err}$ [Strategies discussed in Section 4.3.1]
8: **until** $|C_1^{-1} - C_1^0| < \lambda_1$

Strategies for Updating items in $C_1(\theta)$

The strategies for updating $C_1(\theta)$ is emulated

Strategy 1 ***Instant update*** **:** In this strategy, the corresponding θ_i for which the $\theta_{Error} = |\theta_{Obs} - \theta_i|$ is minimum, is replaced.

With the updated $C_1(\theta)$, a **spline curve (B-Spline or NURBS)** [61] is created where items in the list $C_1(\theta)$ are the control points for the spline. This spline curve is discretized as $C_1^i(\theta) = f(SplineCurve, i)$ where $f(SplineCurve, i)$ is a function which gives θ corresponding to i, where $1 \leq i \leq |C_1(\theta)|$ in the $SplineCurve$. The spline curve removes the irregularities of the observed θ values.

Figure 4.2 shows how the $C_1(\theta)$ gets updated over time. The initial behavior of the robot is represented in $C_1(\theta)$ as a list of angles ranging from 0° to 360os in intervals of 1.8^o forming a sine curve [Figure 4.2 (dark green)]. The X-axis represents the indices/intervals of the list $C_1(\theta)$ and the Y-axis represents the angular movement of the robot. The robot moves randomly in the forward direction. With multiple observations and updates, the curve gets modified, and the robot's C_1 list gets updated as in Figure 4.2 (yellow curve).

Strategy 2 ***Delayed update*** **:** This is an updated strategy of the previous strategy where the Non-smooth Spline is created after taking T observations of θ_{Obs}. After that, the new $C_1(\theta)$ is created from the spline using the technique suggested in strategy 1.

Strategy 3 ***Weighted update*** **:** In this approach, a weight w_i is assigned corresponding to each θ_i, $1 \leq i \leq |C_1|$. The w_i signifies the frequency of the observed behavior of θ. In the beginning, $w_i = 0$ for θ_i is initialized. The weight w_i corresponding to the *minimum* θ^i_{Error} is incremented by 1.

After $N < |C_1|$ observations, only values of θ_i are chosen for which the corresponding w_is are greater than a predefined $Threshold$. This strategy helps to filter out behaviors that are less likely to occur and focuses more on frequent behaviors. The chosen θ_is are used to create a spline using the technique suggested in Strategy 1. The spline is then discretized into $|C_1|, \theta$ states, which are replaced into C_1.

When a spline function is re-computed in $C_1(\theta)$ list after insertion of the state(s), s, the randomness in the states gets absorbed in the newly updated lists.

4.3.2 Update of $C_2(R, d)$

In this section, a method is presented for updating the $C_2(R, d)$ list, where R is the distance traversed and d takes the value of $\{R/-1, F/+1, S/0\}$ representing the "Reverse" and "Forward" direction for translation movement in which the robot moves; and "Still" for when the robot has no translation movement respectively.

Algorithm 9 Update $C_2(R, d)$

1: Initialize $C_2(R, d)$ with pre-meditated behaviour (R, d)
2: $C_2^0 = C_2$
3: At interval $[t_0, t_1]$, observe $[s^{-1}_{Obs}, s^0_{Obs}] = [(R^{-1}_{Obs}, d^{-1}_{Obs}), (R^0_{Obs}, d^0_{Obs})$
4: **repeat**
5: $\quad C_2^{-1} = C_2^0$
6: $\quad$ Minimum Absolute Error $C^{mae}(i)$:

$$C^{mae}(i) \leftarrow \{|R^{-1}_{Obs} \pm R^{i-1}_{C_2^0}| + |R^0_{Obs} \pm R^i_{C_2^0}|\} \quad 1 < i < |C_2^0|$$

where minus($-$) is used when d is same for $(R_{Obs}, d_{Obs}$ and $C_2(R_i, d_i)$ else addition(+) is used.
7: $\quad C_2^0((R^{j-1}, d^{j-1}), (R^j, d^j)) \leftarrow \{[(R^{-1}_{Obs}, d^{-1}_{Obs}), (R^0_{Obs}, d^0_{Obs})] \mid j = \min_{1<i<|C^{mae}|} C^{mae}(i)\}$
8: $\quad$ At interval $[t_k, t_{k+1}], k > 1$, observe $[s^{-1}_{Obs}, s^0_{Obs}] = [(R^{-1}_{Obs}, d^{-1}_{Obs}), (R^0_{Obs}, d^0_{Obs})]$
9: **until** $|C_2^{-1} - C_2^0| \leq \lambda_2$

Strategy for updating $C_2(R, d)$

For every iteration, two consecutive observations s^{-1}_{Obs} and s^0_{Obs} are cached. These observations are searched in C_2 and a pair of consecutive items from C_2 is associated

with it for which minimum absolute error (m.a.e) exists. The m.a.e is the summation of the absolute error between two consecutive (R, d) values in C_2 and $s_{Obs}^{-1}, s_{Obs}^{0}$

This associated pair is replaced with the observations $s_{Obs}^{-1}, s_{Obs}^{0}$ in $C_2(R, d)$. While calculating the error between two consecutive (R, d) values, if d denotes the same direction, the R values are subtracted, otherwise, they are added. The iterations to update the C_2 list continue until the absolute difference between the current C_2 and updated C_2 reach a threshold, $\lambda_2 \geq 0$.

4.3.3 Prediction using the Behaviour Analysis

In this section, a method is discussed for predicting the successive N_F states: $PRC\{s_1, s_2, \ldots, s_{N_F}\}$, where $s_i \in C_1/C_2$ using both $C_1(\theta)$ and $C_2(R, d)$ when two consecutive observation states, s_{Obs}^1 and s_{Obs}^2 are given.

Our system consists of two lists C_1 and C_2, which have information about the frequently occurring states when the robot moves. A queue C_{Obs} is also maintained which has a snapshot of the states of the movement of the robots.

The predicted states, given s_{Obs}^1 and s_{Obs}^2 consist of a combination of states from both the lists C_1 and C_2. Each state can be identified with two identities, the value and the type. The value is the observed measurement and the type will be c_1 for θ and c_2 for (R, d). Based on the given observations, lists of probably predicted N_F states are created using the C_{Obs}. If the value and type of state s_{Obs}^1 followed by the type of state s_{Obs}^2 occurs m times in C_{Obs}, then this will form m lists of length N_F, where each list starts with movement type of s_{Obs}^2 and the consecutive cells having the type of state, i.e. c_1/c_2. Similarly, if the value and type of state s_{Obs}^2 occurs n times, then there will be n lists of states of length N_F starting with the succeeding movement type of s_{Obs}^2 from C_{Obs}. The occurrence of state $s_{Obs}^i, i = 1, 2$ might not occur exactly but with an error less than ϵ in C_{Obs}. Each list ends with a $Marker$ which is initialized to 0. An alternate search approach could be implemented in C_{Obs}. In this method, the first list is created using the value and type of state s_{Obs}^1 followed by the type of state s_{Obs}^2. The second list is created using the type of state s_{Obs}^1 followed by the value and type of state s_{Obs}^2.

Thus, in this way, a matrix M_{Obs} of size $(m+n) \times (N_F+1)$ is formed, where each cell $M_{Obs}[i][j] = c_1/c_2, j \neq (N_F + 1)$. The algorithm is explained with a simple example. Let s_{Obs}^1 type is of c_2, s_{Obs}^2 type is c_1 and $N_F = 4$. Thus, the first two rows of M_{Obs} represent lists of movement types where value s_1^{Obs} occurs in C_{Obs} and starts with

movement type $s_2^{Obs} = c_1$. The next four rows of M_{Obs} represent lists of movement types where value s_2^{Obs} occurs in C_{Obs}.

$$
M_{Obs} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & s_4 & Marker \end{array} \\ \begin{pmatrix} c_1 & c_2 & c_1 & c_2 & 0 \\ c_1 & c_2 & c_2 & c_1 & 0 \\ c_2 & c_1 & c_2 & c_2 & 0 \\ c_1 & c_1 & c_2 & c_1 & 0 \\ c_1 & c_2 & c_1 & c_2 & 0 \\ c_2 & c_1 & c_1 & c_2 & 0 \end{pmatrix} \end{array}
$$

The states in the prediction list will be selected from either the lists C_1 or C_2. Thus, if s_{Obs}^1 has movement type c_1, then it occurs in C_1 at index i where

$$
\min_{1 \leq i \leq (|C_1|)} \{(s_{Obs}^1(\theta) - C_1^i(\theta))^2\} \tag{4.1}
$$

If s_{Obs}^1 has movement type c_2, then it occurs in C_2 at at index i where

$$
\min_{1 \leq i \leq |C_2|} \{|s_{Obs}^1(R) \pm C_2^i(R)|\} \tag{4.2}
$$

where minus($-$) is considered when the directions are the same between observed values and values in $C_2(R, d)$ else sum($+$) is considered.

Similarly, index j will be evaluated for s_{Obs}^2. In the given example, as $s_{Obs}^2 = c_1$, it can be safely concluded that index j is associated with state type c_1. If the movement type of s_{Obs}^2 matches that of s_{Obs}^1, then index j is obtained from the list different from the one associated with the movement type of s_{Obs}^1.

Each predicted state, except for the first state, will depend on the total number of c_1 and c_2 types in each column of the matrix M_{Obs} for rows having $Marker = k - 1$: $M_{c_1}^k, M_{c_2}^k; k = 1 \ldots N_F$ and the previous states in the predicted list.

The probability that the first predicted state of PR will be c_1 type is:

$$
P(s_1 = c_1) = \frac{M_{c_1}^1}{M_{c_1}^1 + M_{c_2}^1}
$$

.

If $P(s_1 = c_1) \geq P(s_1 = c_2)$, the first predicted state in PR, s_1 will be from C_1 after incrementing the index(i/j) corresponding to c_1. Following this, the $Marker$ for all rows having c_1 as their first state will be incremented. If $P(s_1 = c_1) = P(s_1 = c_2)$ for any state, then any one of the states can be chosen, which should be consistent for all periods.

The probability that the second state type $s_2 = c_2$ given $s_1 = c_1$ is:

$$P((s_2 = c_2)|(s_1 = c_1)) = \frac{P((s_2 = c_2) \cap (s_1 = c_1))}{P(s_1 = c_1)}$$

. Probability that $s_2 = c_1$ given that $s_1 = c_1$: $1 - P((s_2 = c_2)|(s_1 = c_1))$.

Similarly, if $P((s_2 = c_2)|(s_1 = c_1)) \geq P((s_2 = c_1)|(s_1 = c_1))$, s_2 will be from C_2 after incrementing the index(i/j) corresponding to c_2. Thus, all $Marker$ corresponding to $s_1 = c_1$ and $s_2 = c_2$ will be incremented. This can be easily done by choosing rows with $Marker = 1$.

This strategy continues for the all next states of PRC. Thus, after all the states have been predicted, the final state of the M_{Obs} and the predicted states of PRC are :

$$M_{Obs} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccc} s_1 & s_2 & s_3 & s_4 & Marker \end{array} \\ \begin{pmatrix} c_1 & c_2 & c_1 & c_2 & 4 \\ c_1 & c_2 & c_2 & c_1 & 2 \\ c_2 & c_1 & c_2 & c_2 & 0 \\ c_1 & c_1 & c_2 & c_1 & 1 \\ c_1 & c_2 & c_1 & c_2 & 4 \\ c_2 & c_1 & c_1 & c_2 & 0 \end{pmatrix} \end{array}$$

$$PRC = \{C_1^{j+1}, C_2^{i+1}, C_1^{j+2}, C_2^{i+2}\}$$

where C_1^j, C_2^i represents the jth and i^{th} element of lists C_1, C_2 respectively.

4.3.4 Complexity

As s_{N_F} is approached, the number of entries in the corresponding state column to be searched gets reduced. While searching for a predicted state $s_i, i < N_F$, if there occurs only one row to be chosen indicated by the $Marker$, the remaining entries of that row represent the remaining states of PRC.

If the indices both are from a single list, i.e. s^1_{Obs}, s^2_{Obs} are of the same state types, then all the states of PRC are chosen from the successive indices corresponding to s^2_{Obs}. If, however, after evaluation of M_{Obs}, a state type corresponding from another list is evaluated, then the predicted state will start from the index having the minimum value in the list.

The indices i/j can be scanned from C_1/C_2, given s^1_{Obs} and s^2_{Obs} in $\mathcal{O}(|C_1| + |C_2|)$ time. Again, the number of scans required to create the $(m+n)$ lists of movement types from C_{Obs} is $(|C_1| + C_2|)$. The entries of each of N_F columns are scanned to decide the count of c_1/c_2 movement types, resulting in a maximum $(m+n) \times N_F$ number of scans.

In the queue C_{Obs}, the state s^1_{Obs} followed by state s^2_{Obs} and the value of state s^2_{Obs} both can occur maximum $(|C_1| + |C_2|)$ times. Thus, the maximum and minimum values of $(m+n)$ is $2 \times (|C_1| + |C_2|)$ and 1 respectively. As all states are equally probable in C_{Obs} and consecutively in M_{Obs}, the expected number of scans for c_1/c_2 in each column is $\mathcal{O}(|C_1| + |C_2|)$. Hence, the expected time required to predict N_F successive states is $\mathcal{O}((|C_1| + |C_2|) \times N_F)$ and the expected space requirement for M_{Obs} is $\mathcal{O}((|C_1| + |C_2|) \times N_F)$.

4.4 A Graph-Based Approach: Alternative Solution for the Behaviour Analysis for 2-DOF

In this section, another method is proposed for predicting the sequence of "successive steps" analyzing the behaviour of a 2-DOF wheeled robot. The motion of a robot, which can be a "continuous curve" can be discretized into a sequence of valid states, $S = \{s_i | i \geq 1, s_i =$ is the corresponding observed state.$\}$.

4.4.1 Data Representation and Construction of the "Behavioural Network"

A graph $G(V, E)$ represents the behavioural knowledge of the robot's motion, which can be referred to as the "Behavioral Network" for the robot. In this graph, the set of vertices $V(G) = S$, where each vertex represents a state s_i and $E(G)$ is the weighted directed edge between the two states. A state s_i is the "independent value" w.r.t. to each "independent movement" which constitutes the DOF of the robot. The edge weight, $(w(\overrightarrow{e}(s^i_{Obs}, s^j_{Obs}))$ represents the frequency of the adversary robot,

which follows the transition between the two states s^i_{Obs} and s^j_{Obs}. An edge of high frequency/weight signifies that the robot frequently follows these intermittent states.

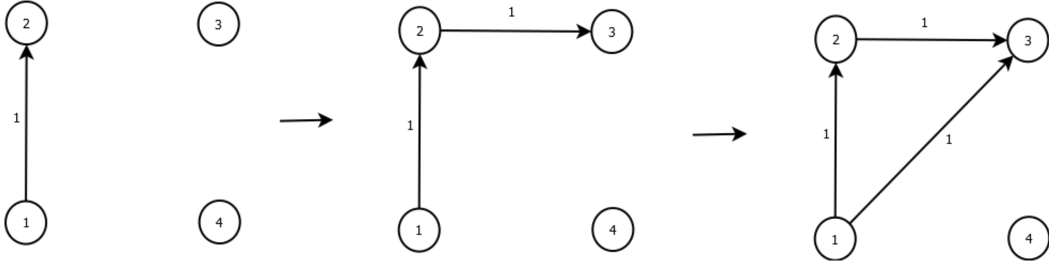


Figure 4.3: Graph creation

While updating the graph, one starts with the first two observed states s^1_{Obs} and s^2_{Obs}. These state nodes are joined by a directed edge $w(\overrightarrow{e}(s^1_{Obs}, s^2_{Obs})) = 1$. If $s^1_{Obs} = s^2_{Obs}$, then a loop is created at s^1_{Obs} with edge weight $w(\overrightarrow{e}(s^1_{Obs}, s^1_{Obs})) = 1$. If the next observation, s^3_{Obs} does not exist among the existing states, a new directed edge will be created with $w(\overrightarrow{e}(s^2_{Obs}, s^3_{Obs})) = 1$. If $s^1_{Obs} = s^2_{Obs} = s^3_{Obs}$, then $w(\overrightarrow{e}(s^1_{Obs}, s^1_{Obs}))$ is incremented by 1. If any two consecutive observed states s^i_{Obs} and s^j_{Obs} is repeated, then $w(\overrightarrow{e}(s^i_{Obs}, s^j_{Obs}))$ is "incremented" by 1.

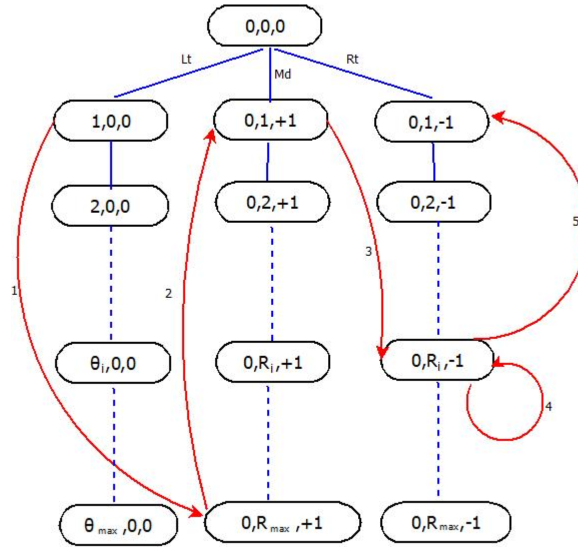


Figure 4.4: Search and Transition Combined Graph

A single graph can represent the state transitions between the identified states and the search for s_{Obs}. The graph has a starting node represented by the state $0, (0, 0)$. From this state, the robot can have any one of the three motions i.e. $(\theta, (0, 0))$ or $(0, (R, \pm 1))$. Thus, the state $(0, (0, 0))$ points to three lists of states, Lt, Md and Rt:

1. Lt is a sorted list of valid non-zero θ values (in ascending order) and $(R, d) = (0, 0)$.

2. Md and Rt are a sorted list of $(R, +1)$ and $(R, -1)$ (in ascending order) respectively and $\theta = 0$ for both.

Each state node in the lists has two outgoing edges [Figure 4.4]: RED representing the state transitions and BLUE for searching the s_{Obs} among the existing states. If the s_{Obs} does not exist in any of the lists (which can be linearly searched in a list) Lt, Md, or Rt, it can be appropriately inserted in the valid list. The observed state s_{Obs} can be searched in the lists Lt, Md or Rt based on $(\theta_{Obs}, (0, 0))$ or $(0, (R_{Obs}, +1))$ or $(0, (R_{Obs}, -1))$ respectively.

As the translation length due to a complete rotation of a wheel is R_{Max}, any translation $R_{Obs} > R_{Max}$ in the forward direction can be represented with $\left\lfloor \frac{R_{Obs}}{R_{Max}} \right\rfloor + R'_{Obs}$. The state transition will result in $\left\lfloor \frac{R_{Obs}}{R_{Max}} \right\rfloor$ number of self loop transition at state $(0, (R_{Max}, +1))$ and another transition from $(0, (R_{Max}, +1))$ to $(0, (R'_{Obs}, +1)$. This will be similar for θ_{Obs} and $(0, (R_{Obs}, -1))$.

Complexity of Construction

The total number of vertices depends on the DOF of the robot. For a 2-DOF robot, the maximum number of valid states/vertices is $2 \times |R| + |\Theta|$, where $R = \{R_i | R_i = \{R_{Gap}, 2 \times R_{Gap}, \ldots, N \times R_{Gap} = R_{Max}\}\}$; and $\Theta = \{\theta_i | \theta_i = \{\theta_{Gap}, 2 \times \theta_{Gap}, \ldots, N' \times \theta_{Gap} = \theta_{Max}\}\}$. Every node representing a state in the graph needs to store references for a maximum $2 \times |R| + |\Theta|$ states where possible transitions can occur. So space complexity of this graph is $\mathcal{O}((|R| + |\Theta|)^2)$. As searching for s_{Obs} is done in an ordered list, the time complexity for searching is $\max\{\mathcal{O}(|R|), \mathcal{O}(|\theta|)\}$.

4.4.2 Prediction using the Behavioural Network $G(V, E)$

This section discusses a method for predicting the "succeeding" n states from a state, s using the "Behavioural Network". The network comprises $|V(G)|$ nodes corresponding to the $|S|$ states. After searching the state s in the graph, The following cases can arise while traversing the graph $G(V, E)$ for predicting n "succeeding states" from a state $s \in S$

Case 1 : If there exists a single outgoing edge from vertex s, the "succeeding" step is the next vertex corresponding to next state.

Case 2 : If there exist multiple outgoing edges with unique weights, the "maximum weighted" outgoing edge from vertex s is the "succeeding" state.

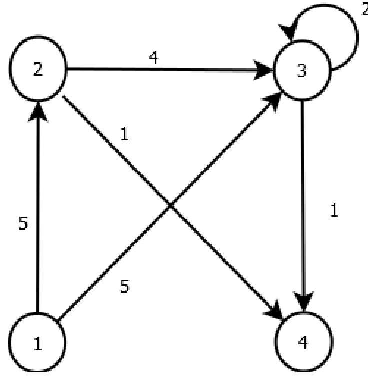


Figure 4.5: Graph with 4 nodes, where each node corresponds to s_i, for $i = 1, \ldots, 4$

Case 3 : If there exist multiple outgoing edges with identical maximum weights, one can adopt different strategies while choosing the "succeeding" state.

In one strategy, one gives equal probability to all the similar weighted edges from s and the maximum-weighted edge is chosen randomly.

In another strategy, a 1-state **look-forward** search among all the head nodes of the similar weighted edge(s) from s is performed. Let $S^0_{succ} = \{s^{0i}_s | i \leq Degree(s)\}$ be the set of all head nodes incident from s having a similar maximum weighted edge(s). If s^{1i}_s be the head of the maximum-weighted edge incident from $s^{0k}_s \in S^0_{succ} | 1 \leq k \leq |S^0_{succ}|$, then the "succeeding nodes" of s are $\{s^{0k}_s, s^{1i}_s\}$.

Similarly, if there are again similar maximum-weighted edge(s) from some/all $s^{1i}_s \in S^1_{succ}$, then the 1-state **look-forward** search continues from these s^{1i}_s nodes. This search continues till a unique edge with maximum weight is found. Thus, the "succeeding nodes" of s will be all nodes along the path from s to the head of the unique edge found.

Case 4 : If the highest outgoing node is a loop, the "succeeding state" is the same node, and the following "succeeding state" will be the head of the second highest weighted edge. This step will be important to break the loop or else all the subsequent states will be similar.

Figure 4.5 shows a "Behavioral Graph" with the traversed edges and their frequency of state transitions. Based on the above rules, starting from node $s = 1$, the next **four(4) steps** are predicted

Step n=1: Node $s_1 = 1$ has two(2) outgoing edges. The edge weights $w(\overrightarrow{e}(s_1, s_2)) =$

$w(\overrightarrow{e}(s_1, s_3)) = 5$ are same. To break the tie, the outgoing edges of nodes s_2 and s_3 are observed. It can be noted that the highest weighted edge is for s_2. Thus, s_2 is selected.

Step n=2: Node s_3 is chosen as this has the highest weighted outgoing edge from s_2.

Step n=3: Node s_3 as it has a self-loop with the highest outgoing edge from s_3.

Step n=4: After the self-loop is traversed, the second highest outgoing edge from s_3 corresponds to node s_4.

Thus, the predicted following 4 states from s_1 are $\{s_2, s_2, s_3, s_4\}$

Let us start with vertex s_2. From Figure 4.5, it can be seen that vertex s_2 has two(2) outgoing edges connecting vertices s_3 and s_4; $w(\overrightarrow{e}(s_2, s_3)) = 4$ and $w(\overrightarrow{e}(s_2, s_4)) = 1$. Thus, the probability that the next step taken by the robot corresponding to vertex s_3 is $\frac{4}{5}$ and s_4 will $\frac{1}{5}$. Thus, $P(s_3|s_2) = \frac{4}{5}$ and $P(s_4|s_2) = \frac{1}{5}$

Generalizing, the probability that state s_j will be chosen provided the current state is s_i is given by

$$P(s_j \mid s_i) = \frac{w(\overrightarrow{e}(s_i, s_j))}{\sum_{e=\text{outgoing edges from } s_i} w(\overrightarrow{e})} \quad (4.3)$$

Thus, the total probability of error for predicting n states is from node s_i

$$1 - \Pi_{s_i, s_j \in S} P(s_j|s_i) \text{ for } n \text{ states} \quad (4.4)$$

Complexity of Prediction

It can be observed that after finding the state in the graph, the prediction of the next states just needs linear traversal following the transition edges. So the time complexity of predicting the future steps is the same as the time complexity of searching a state in the graph i.e. $\mathcal{O}(|R| \log |R|)$ or $\mathcal{O}(|\theta| \log |\theta|)$, based on which chain the searching was carried out.

4.4.3 A “Markov Chain” equivalent of “Behavioral Network”

The conditional distribution of any state in the Behavioural Network depends only on its previous state, thus this can be treated as a Markov chain. The Markov chain

can be represented by the n states $\{S_k | k = 0, 1, 2, \ldots, n-1\}$ of the “Behavioural Network”. If the process is at state s_i at time k, then this is represented as $S_k = i$. If the one-step transition probability matrix for this Markov chain is A, then the probability of transition from state s_i to s_j in k steps is given by $P^k_{ij} = A^k[i,j]$, where A^k is the k-step transition probability matrix. As all the states are equally likely, the probability for any state to start as the initial state is equal and the corresponding probability for the state s_i is given by $m_{s_i} = P\{S_0 = i\} = \frac{1}{n}$. The unconditional probability that the Behavioural Network will be at state s_j after l steps, irrespective of the starting state, is given by $P\{S_l = j\} = \frac{1}{n}\sum_{i=0}^{n-1} P^l_{ij}$.

If the “equivalence relation” of communication is applied on the Markov Chain, then the state space is partitioned into disjoint irreducible subsets, where all states in the subset communicate with each other [57][60]. The weight of a partitioned set W_l $(0 \leq l \leq n-1)$ is the number of states ($|W_l|$) in the partition and $1 \leq |W_l| \leq n$. Considering the above observation, the transitional probability among states can be adjusted before predicting the n states. If there are k outgoing edges from s_i, then the ”adjusted” transitional probability from s_i to s_j can be stated by

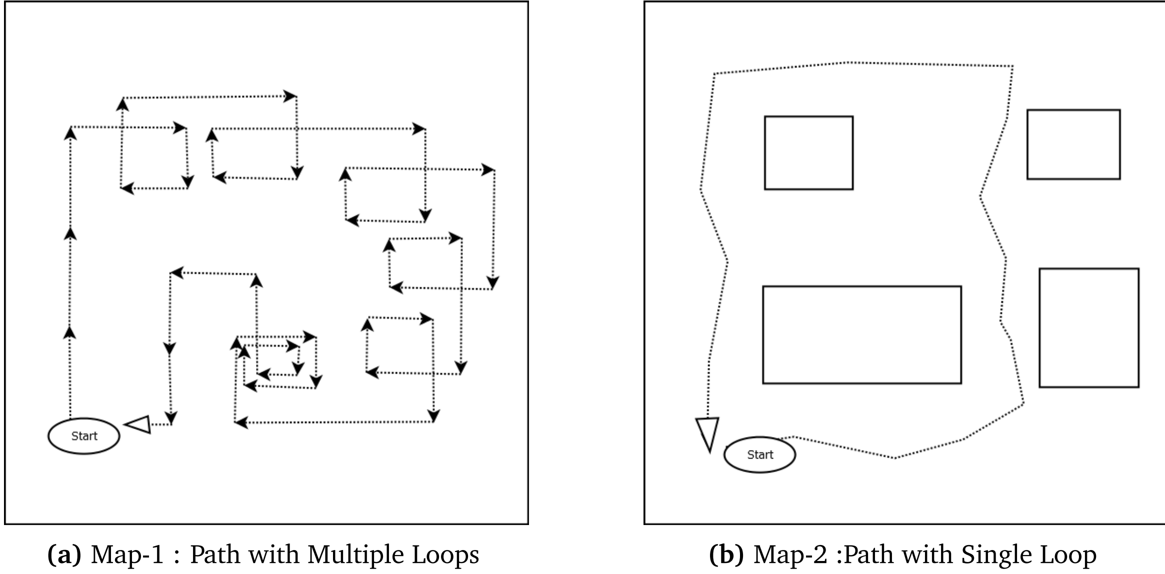
$$P^1_{ij}(adjusted) = \frac{P^1_{ij} W_j}{\sum_{m=1}^{k} P^1_{im} W_m}$$

where W_j and W_m are the weights of the sets to which the states s_j and s_m belong to. This “adjusted” transitional probability can be used instead of the original transitional probability while predicting the n-steps, while the other steps remain the same as in subsection 4.4.2.

4.5 Comparative study between the proposed algorithms

In this section, the results of the proposed methods are discussed and compared based on the two maps: Map-1 [Figure 4.6a] and Map-2 [Figure 4.6b].

Figure 4.6b depicts an adversary robot moving in a path around some polygons without any self-loops while Figure 4.6a depicts the movement of a robot having multiple sub-loops, and both the robots’ source and destination point are same.



(a) Map-1 : Path with Multiple Loops

(b) Map-2 :Path with Single Loop

Figure 4.6: Experimental Maps

4.5.1 Analysis of Update Activity in List-Based Approach

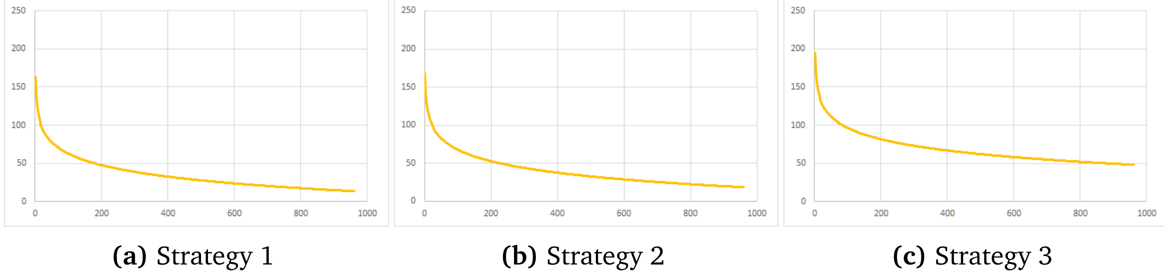
In this section, the behavior based on the list-based approach on maps in Figure 4.6 is analyzed. For both the maps, the lists $C_1(\theta)$ and $C_2(R,d)$ with an observation length of 200, are initialized with a premeditated motion; $C_1(\theta)$ is a list of angles from 0° to 360° in intervals of 1.8^o; $C_2(R,d)$ is a list where it is assumed that $R = 15$, i.e. the robot at each observation has a translation motion of 15 units; and $d = -1$ for observations from $1 - 50$ and $d = +1$ for from $101 - 150$.

As the robot does its actual motion, its angle of rotation and translation is observed, and the corresponding lists of $C_1(\theta)$ and $C_2(R,d)$ are updated. These observations are updated as per Algorithm 8 and Algorithm 9. The experiments were conducted repeatedly for a different number of steps of the robot and, for each experiment, the errors[1] is the summation of all the errors calculated for each steps corresponding to that experiment.

$$Error = \sum_{k=1\to j | j=\text{Total number of steps in the experiment}} L_{Robot}|\theta_k - \theta_{obs}| + |R_k \pm R_{obs}| \qquad (4.5)$$

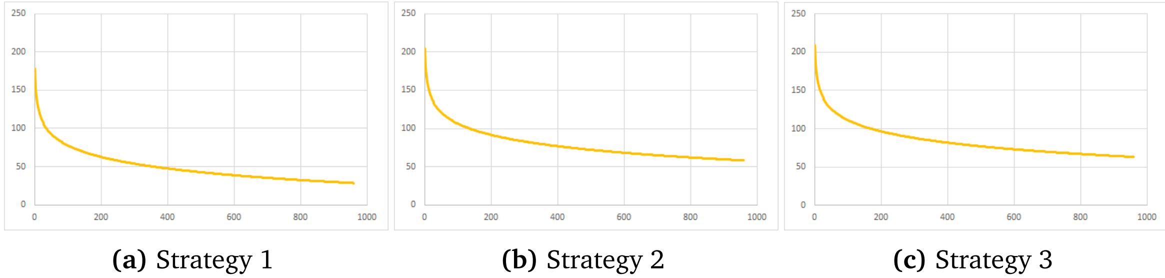
where L_{Robot} is the length of the robot; $L_{Robot}|\theta_k - \theta_{obs}|$ and $|R_k \pm R_{obs}|$ are both

[1] The $Error$-axis, i.e. Y-Axis in the graphs of Figures 4.7, 4.8, 4.9, 4.10 is $Error \times 10^{-3}$ and the X-Axis represents the number of steps taken by the robot for the experiment.



(a) Strategy 1 **(b)** Strategy 2 **(c)** Strategy 3

Figure 4.7: 2-DOF Convergence of error with respect to steps on Map-1



(a) Strategy 1 **(b)** Strategy 2 **(c)** Strategy 3

Figure 4.8: 2-DOF Convergence of error with respect to steps on Map-2

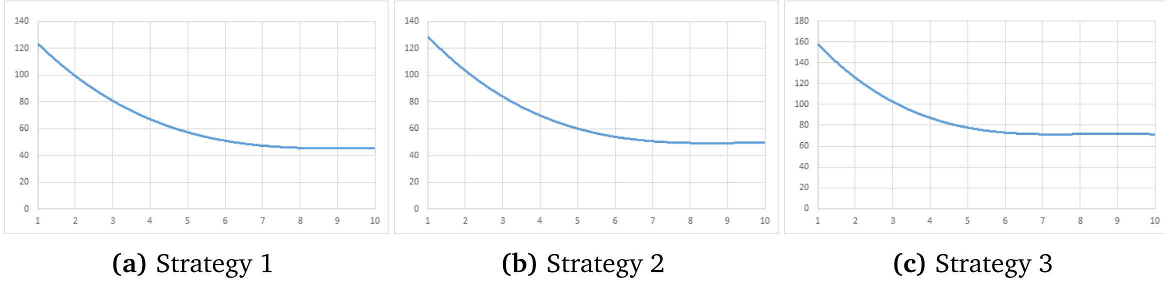
normalised; and the rules for choosing $+/-$ are previously discussed.

It is observed that the error during update decreases and converges logarithmically as the number of steps increases; and Map-1 converges faster than Map-2 as shown in [Figure 4.7] and [Figure 4.8].

Comparison of Convergence of Error w.r.t. Recurring "loop"s

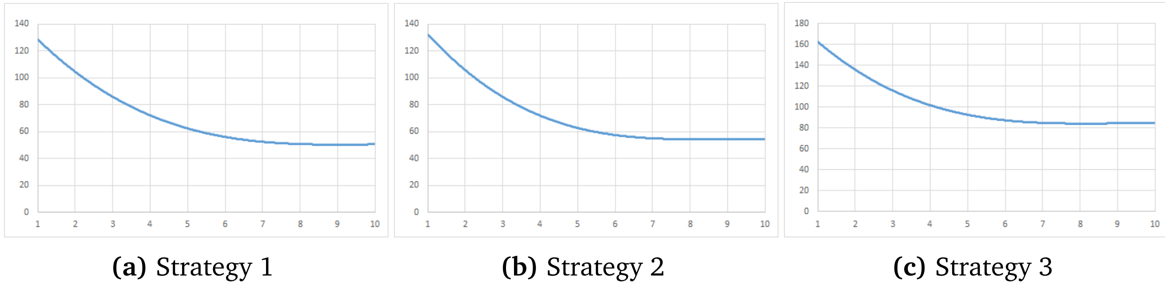
A "loop" is defined as a path/sub-path when the source and destination are the same. In Map-1, there are multiple inner loops inside the path; and for Map-2, the path does not have any inner loops. Thus, when a robot traverses in Map-1, the error for locating the indices in $C_1(\theta)$ and $C_2(R, d)$ decreases faster than Map-2. The algorithms were run for a number of iterations for each experiment, where in each iteration the robot reaches the same location from where it started. The convergence for three different strategies of the total error with respect to a number of iterations for Map-1 and Map-2 are shown in [Figure 4.9] and [Figure 4.10] respectively. The total error for Map-1 converges faster than Map-2 for the same number of iterations because Map-1 contains more smaller loops indicating more repetition of specific behavior and hence easily identified by the algorithm, generating less error.

The *instant update strategy* for $C_1(\theta), C_2(R, d)$ converges faster than *step interval update* and *weighted update strategies* as in the latter two strategies, the learning is



(a) Strategy 1 **(b)** Strategy 2 **(c)** Strategy 3

Figure 4.9: 2-DOF Convergence of error with respect to iteration on Map-1



(a) Strategy 1 **(b)** Strategy 2 **(c)** Strategy 3

Figure 4.10: 2-DOF Convergence of error with respect to iteration on Map-2

delayed and thus locating the correct indices of $C_1(\theta)$ and $C_2(R, d)$ also takes more steps. The weighted update strategy also takes more steps to converge as this strategy ignores small updates and only captures information about the major behaviors of the robot.

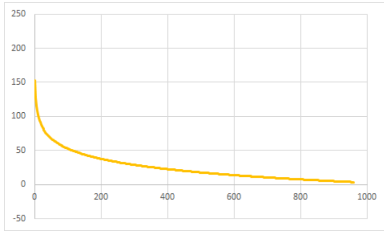
4.5.2 Analysis of Update Activity in "Behavioral Network"

In this section, the error during the update of the "Behavioral Network" based on observations is analyzed.

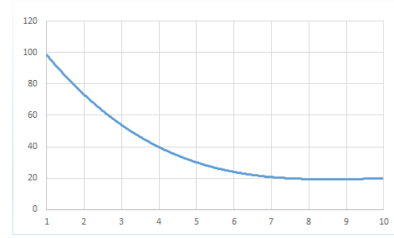
The calculated error for the Behavioural Network converges faster than when using $C_1(\theta)$ and $C_2(R, d)$ lists. This is because the Behavioural Network keeps the information about all the possible connections between a state and all possible next states and the next state is chosen for which the connecting edge has the higher frequency of use. This is not possible for the $C(\theta)$ and $C_1(\theta), C_2(R, d)$ list, where there is no indication of previous patterns while selecting indices in these lists.

It is observed that the total error converges following a negative logarithmic function for both maps. Also the total error w.r.t. steps converges faster for Map-1 [Figure 4.11a] than Map-2 [Figure 4.12a].

As more iterations, the robot makes w.r.t. its source/destination, the lesser the total

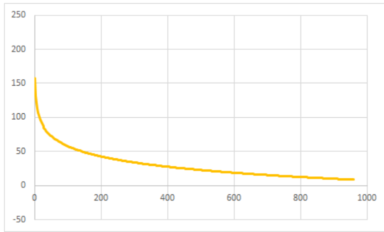


(a) Convergence of Error w.r.t. steps

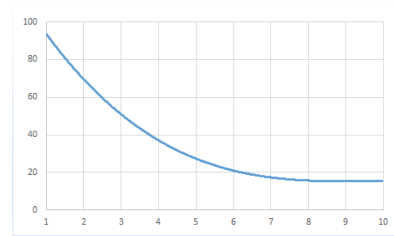


(b) Convergence of Error w.r.t. iterations

Figure 4.11: Behavioral Analysis using the "Behavioral Network" on Map 1



(a) Convergence of Error w.r.t. steps



(b) Convergence of Error w.r.t. iterations

Figure 4.12: Behavioral Analysis using the "Behavioral Network" on Map 2

error. It can be noticed that Map-1 performs better than Map-2 as the robot moves to make repeated loops in an iteration while moving. The error convergence w.r.t. iterations occurs earlier in case of Map-1 [Figure 4.11b] than Map-2 [Figure 4.12b].

4.6 Conclusion

In this chapter, two methods are proposed that use a list and a graph respectively to predict the expected path of autonomous non-communicating robots. The predicted path of the adversary robot is represented in terms of $(\theta, (R, d))$. Without any knowledge about the robot's internal state, the prediction is based only on the observed motion. This observed motion is expressed as the "behavior" of the robot.

Three strategies have been proposed for updating the observed steps using a spline function for $C_1(\theta)$ in the list-based method. The spline function has been used to remove the randomness of the steps. For future research, a study of other spline functions on $C_1(\theta)$ can be done. Moreover, strategies can also be proposed for updating $C_2(R, d)$. These methods can be extended to incorporate other observations like the yea/yaw of an actual robot.

The proposed methods were tested on two different maps: Map-1 having multiple

loops in a path and Map-2 having a single looped path. Results show that all the methods performed better on Map-1. This shows that the methods could effectively analyze the pattern in the movement and predict the path faster with maps having paths with more loops. These methods are also effective for open paths having repeating patterns within them.

Chapter 5

Conclusion and Future Works

The primary motivation of this research was finding simple solutions for the strategic placement of circles covering free space within a bounded environment and predicting future states of adversary agents using time-independent models that require low computing power. In this thesis, two novel methods have been proposed for the strategic placement of vision circles in an environment. One of these methods is a discrete method that works in an environment with convex and concave polygons and tries to cover the free space as much as possible. The other method is a heuristic one that uses genetic algorithm and is applicable for an environment with convex polygons.

Finding a deterministic solution for the circle placement problem is not possible, so the proposed discrete method generates an approximate solution. In this solution, once a vision circle is placed, it is treated as a polygon and merged with existing polygons so that in the next iteration, another vision circle can be placed in the remaining uncovered area. This solution takes quadratic time and space to place a vision circle and merge it with existing polygons and already-covered areas. An interesting extension of this approach would be to use these vision circles to save the autonomous robots during their movement in an environment infested with enemy agents. A centralized system would take data from the 360^0 surveillance cameras placed in the centers of these vision circles and notify the autonomous robots regarding the movement of enemy agents inside the vision circles so that these robots can plan their motion while avoiding damage from the enemy agents. Another extension of the proposed solution would be experimenting with a greedy algorithm that uses a suitable heuristic function to determine the location of the vision circle

placement.

The heuristic method uses genetic algorithm to find an arrangement of circles that maximizes packing density while adhering to the boundaries of convex polygons and already placed circles. Potential avenues for future research in this method include exploring additional optimization techniques, refining the components of the genetic algorithm, and expanding the methodology to handle more complex scenarios involving diverse polygon shapes and sizes. Computational results are provided to demonstrate the efficiency of the heuristic approach.

This thesis also proposes a data structure and its corresponding algorithm for maintaining a vertical cell decomposition of an environment after the insertion and deletion of a polygon. It has been shown that for specific arrangements of concave polygons, the expected time complexity for updating the relevant trees is $O(NH)$ where N is the number of existing polygons and H is the number of concave angles of the polygon which is inserted or deleted.

In this work, two methods, using a list and a graph, are proposed to predict the expected path of autonomous non-communicating robots. The predicted path of the adversary robot is represented in terms of $(\theta, (R, d))$ or the "behaviors" of the robot.

Three strategies have been proposed for updating the observed steps using a spline function for $C_1(\theta)$ in the list-based method. The spline function has been used to remove the randomness of the steps. For future research, a study of other spline functions on $C_1(\theta)$ can be done. Moreover, strategies can also be proposed for updating $C_2(R, d)$. These methods can be extended to incorporate other observations like the yea/yaw of an actual robot.

The proposed methods were tested on two different maps: Map-1 having multiple loops in a path and Map-2 having a single looped path. Results show that all the methods performed better on Map-1. This shows that the methods could effectively analyze the pattern in the movement and predict the path faster with maps having paths with more loops. These methods are also effective for open paths having repeating patterns within them.

The limitation of these two prediction methods stems from their inability to utilize data from strategically positioned surveillance cameras in open areas. An intriguing enhancement to these methods would entail collecting real-time location data of adversary agents within one or more visual range circles and integrating it with the

comprehensive 360-degree view camera data for heightened predictive accuracy.

Bibliography

[1] Bernardetta Addis, Marco Locatelli, and Fabio Schoen. Packing circles in a square: New putative optima obtained via global optimization. *Optimization Online*, 154, 2005. pages 41

[2] Ali Ahmadinia, Christophe Bobda, Sandor P. Fekete, Jurgen Teich, and Jan C. van der Veen. Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices. *IEEE Transactions on Computers*, 56(5):673–680, 2007. pages 17

[3] Helmut Alt, Mark de Berg, and Christian Knauer. Approximating minimum-area rectangular and convex containers for packing convex polygons. In *Algorithms-ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 25–34. Springer, 2015. pages 40

[4] Lars Arge and Jan Vahrenhold. I/o-efficient dynamic planar point location. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 191–200, 2000. pages 37

[5] Aishik Basu, Bappaditya Das, and Chintan Kr Mandal. A method for finding multiple large rectangular free spaces in a map with convex and concave obstacles. In *2021 IEEE 18th India Council International Conference (INDICON)*, pages 1–6, 2021. pages 78

[6] Asma Belhadi, Youcef Djenouri, Gautam Srivastava, Djamel Djenouri, Jerry Chun-Wei Lin, and Giancarlo Fortino. Deep learning for pedestrian collective behavior analysis in smart cities: A model of group trajectory outlier detection. *Information Fusion*, 65:13–20, 2021. pages 12, 83

[7] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. pages 14, 18, 33, 36

[8] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008. pages 48, 49

[9] M. Berg, de, O. Cheong, M.J. Kreveld, van, and M.H. Overmars. *Computational geometry : algorithms and applications*. Springer, Germany, 3rd edition, 2008. pages 81

[10] Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry*. Cambridge university press, 1998. pages 18

[11] J. Bruce, M. Bowling, B. Browning, and M. Veloso. Multi-robot team response to a multi-robot opponent team. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 2281–2286 vol.2, 2003. pages 12

[12] Joel M. Caplan, Leslie W. Kennedy, and Gohar Petrossian. Police-monitored cctv cameras in newark, nj: A quasi-experimental test of crime deterrence. *Journal of Experimental Criminology*, 7(3):255–274, Sep 2011. pages 39

[13] B. M. Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36(1):1–16, Mar 1986. pages 39, 79

[14] Mao Chen, Xiangyang Tang, Zhizhong Zeng, and Sanya Liu. An efficient heuristic algorithm for two-dimensional rectangular packing problem with central rectangle. *Journal of Industrial and Management Optimization*, 16(1):495–510, 2018. pages 41

[15] Yi-Jen Chiang, Franco P. Preparata, and Roberto Tamassia. A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps. *SIAM Journal on Computing*, 25(1):207–233, 1996. pages 36

[16] Dimitrios Chrysostomou and Antonios Gasteratos. Optimum multi-camera arrangement using a bee colony algorithm. In *2012 IEEE International Conference on Imaging Systems and Techniques Proceedings*, pages 387–392, 2012. pages 39

[17] Christopher Crick and Brian Scassellati. Controlling a robot with intention derived from motion. *Topics in Cognitive Science*, 2:114 – 126, 01 2010. pages 83, 84

[18] Karen Daniels, Victor Milenkovic, and Dan Roth. Finding the largest rectangle in several classes of polygons. *Harvard Computer Science Group Technical Report TR-22-95*, 1995. pages 39, 50, 79

[19] Lawrence Davis. *Handbook of genetic algorithms*. VNR computer library. Van Nostrand Reinhold New York, New York, 1991. pages 67

[20] Matthew Dickerson and Daniel Scharstein. Optimal placement of convex polygons to maximize point containment. *Computational Geometry*, 11(1):1–16, 1998. pages 39, 79

[21] David Dobkin and Richard J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, 1976. pages 49

[22] Bahaa Eldin El-Shweky, Karim El-Kholy, Mahmoud Abdelghany, Mahmoud Salah, Mohamed Wael, Omar Alsherbini, Yehea Ismail, Khaled Salah, and Mohamed AbdelSalam. Internet of things: A comparative study. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 622–631, 2018. pages 13

[23] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Soft + hardwired attention: An lstm framework for human trajectory prediction and abnormal event detection. *Neural Networks*, 108:466–478, 2018. pages 82

[24] Aphrodite Galata, Neil Johnson, and David Hogg. Learning variable-length markov models of behavior. *Computer Vision and Image Understanding*, 81(3):398–413, 2001. pages 82

[25] A.S. Glassner. *An Introduction to Ray Tracing*. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Elsevier Science, 1989. pages 62

[26] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2255–2264, 2018. pages 83, 84

[27] Mehmet Serdar Guzel and Robert Bicker. Vision based obstacle avoidance techniques. *Recent advances in mobile robotics*, pages 83–108, 2011. pages 11

[28] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011. pages 13

[29] Van-Dung Hoang and Kang-Hyun Jo. Path planning for autonomous vehicle based on heuristic searching using online images. *Vietnam Journal of Computer Science*, 2(2):109–120, May 2015. pages 17

[30] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui, and Yao Wen. Vehicle Trajectory Prediction based on Motion Model and Maneuver Recognition. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, pages 4363–4369, Tokyo, Japan, November 2013. pages 83

[31] Manh Huynh and Gita Alaghband. Trajectory prediction by coupling scene-lstm with human movement lstm. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Daniela Ushizima, Sek Chai, Shinjiro Sueda, Xin Lin, Aidong Lu, Daniel Thalmann, Chaoli Wang, and Panpan Xu, editors, *Advances in Visual Computing*, pages 244–259, Cham, 2019. Springer International Publishing. pages 82

[32] Masaya Inoue, Takahiro Yamashita, and Takeshi Nishida. Robot path planning by lstm network under changing environment. In Sanjiv K. Bhatia, Shailesh Tiwari, Krishn K. Mishra, and Munesh C. Trivedi, editors, *Advances in Computer Communication and Computational Sciences*, pages 317–329, Singapore, 2019. Springer Singapore. pages 83, 84

[33] Joshua Joseph, Finale Doshi-Velez, Albert S Huang, and Nicholas Roy. A bayesian nonparametric approach to modeling motion patterns. *Autonomous Robots*, 31(4):383, 2011. pages 82

[34] Kihwan Kim, Dongryeol Lee, and Irfan Essa. Gaussian process regression flow for analysis of motion trajectories. In *2011 International Conference on Computer Vision*, pages 1164–1171. IEEE, 2011. pages 82

[35] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. pages 18

[36] Andreas C Koenig. A study of mutation methods for evolutionary algorithms. *University of Missouri-Rolla*, 2002. pages 68

[37] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/. pages 81

[38] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006. pages 14, 18

[39] D.T. Lee and C.C. Yang. Location of multiple points in a planar subdivision. *Information Processing Letters*, 9(4):190–193, 1979. pages 18

[40] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017. pages 83

[41] Zhaoying Li, Zhao Zhang, Hao Liu, and Liang Yang. A new path planning method based on concave polygon convex decomposition and artificial bee colony algorithm. *International Journal of Advanced Robotic Systems*, 17(1):1729881419894787, 2020. pages 17

[42] Matthias Luber, Johannes A Stork, Gian Diego Tipaldi, and Kai O Arras. People tracking with human motion predictions from social forces. In *2010 IEEE International Conference on Robotics and Automation*, pages 464–469. IEEE, 2010. pages 83

[43] Andréa Macario Barros, Maugan Michel, Yoann Moline, Gwenolé Corre, and Frédérick Carrel. A comprehensive survey of visual slam algorithms. *Robotics*, 11(1), 2022. pages 11

[44] Peter V. Marsden. Network analysis. In Kimberly Kempf-Leonard, editor, *Encyclopedia of Social Measurement*, pages 819–825. Elsevier, New York, 2005. pages 48, 57

[45] Tinhinane Mezair, Youcef Djenouri, Asma Belhadi, Gautam Srivastava, and Jerry Chun-Wei Lin. Towards an advanced deep learning for the internet of behaviors: Application to connected vehicles. *ACM Trans. Sen. Netw.*, 19(2), dec 2022. pages 83

[46] D. Mitrovic. Reliable method for driving events recognition. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):198–205, 2005. pages 81

[47] M.G. Mohanan and Ambuja Salgoankar. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100:171–185, 2018. pages 11

[48] Asish Mukhopadhyay and S. V. Rao. On computing a largest empty arbitrarily oriented rectangle. *International Journal of Computational Geometry & Applications*, 13(03):257–271, 2003. pages 39, 79

[49] Subhas C. Nandy, Arani Sinha, and Bhargab B. Bhattacharya. Location of the largest empty rectangle among arbitrary obstacles. In P. S. Thiagarajan, editor, *Foundation of Software Technology and Theoretical Computer Science*, pages 159–170, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. pages 39, 79

[50] L.E. Parker and B.A. Emmons. Cooperative multi-robot observation of multiple moving targets. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2082–2089 vol.3, 1997. pages 83, 84

[51] Jeinny Peralta, Marina Andretta, and José Fernando Oliveira. Packing circles and irregular polygons using separation lines. In *ICORES*, pages 71–77, 2018. pages 40

[52] Francesco Piccialli, Salvatore Cuomo, Fabio Giampaolo, Giampaolo Casolla, and Vincenzo Schiano di Cola. Path prediction in iot systems through markov chain algorithm. *Future Generation Computer Systems*, 109:210–217, 2020. pages 82

[53] Mohammad Pirani, Joshua A. Taylor, and Bruno Sinopoli. Strategic sensor placement on graphs. *Systems & Control Letters*, 148:104855, 2021. pages 39

[54] Sarthak Pradhan, Ravi Kumar Mandava, and Pandu R. Vundavilli. Development of path planning algorithm for biped robot using combined multipoint rrt and visibility graph. *International Journal of Information Technology*, 13(4):1513–1519, Aug 2021. pages 17

[55] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob N Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21, 2020. pages 82, 83

[56] T Romanova, A Pankratov, I Litvinchev, Yu Pankratova, and I Urniaieva. Optimized packing clusters of objects in a rectangular container. *Mathematical Problems in Engineering*, 2019:1–12, 2019. pages 40

[57] Sheldon M. Ross. 4 - markov chains. In Sheldon M. Ross, editor, *Introduction to*

Probability Models (Twelfth Edition), pages 193–291. Academic Press, twelfth edition edition, 2019. pages 98

[58] Hanan Samet and Robert E Webber. Storing a collection of polygons using quadtrees. *ACM Transactions on Graphics (TOG)*, 4(3):182–222, 1985. pages 18

[59] Lothar M. Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1):1–61, 2001. pages 67

[60] Richard Serfozo. *Markov Chains*, pages 1–98. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. pages 98

[61] Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics*. AK Peters/CRC Press, 2009. pages 88

[62] Péter Gábor Szabó, Mihaly Csaba Markót, Tibor Csendes, Eckard Specht, Leocadio G Casado, and Inmaculada García. *New approaches to circle packing in a square: with program codes*, volume 6. Springer Science & Business Media, 2007. pages 41

[63] Kai Tang, Charlie CL Wang, and Danny Z Chen. Minimum area convex packing of two convex polygons. *International Journal of Computational Geometry & Applications*, 16(01):41–74, 2006. pages 40

[64] Tawseef Ahmed Teli and M. Arif Wani. A fuzzy based local minima avoidance path planning in autonomous robots. *International Journal of Information Technology*, 13(1):33–40, Feb 2021. pages 17

[65] Vasudha Todi, Gunjan Sengupta, and Sourangshu Bhattacharya. Probabilistic path planning using obstacle trajectory prediction. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 36–43, 2019. pages 83, 84

[66] Motoyasu Tooyama, Kazuyoshi Wada, and Mutsuki Yageta. Development of behavior observation robot ver.2. In Naoyuki Kubota, Kazuo Kiguchi, Honghai Liu, and Takenori Obo, editors, *Intelligent Robotics and Applications*, pages 266–275, Cham, 2016. Springer International Publishing. pages 84

[67] Nan Wang, Jie-Sheng Wang, Yong-Xin Zhang, and Tian-Zhu Li. Two-dimensional bin-packing problem with rectangular and circular regions solved

by genetic algorithm. *IAENG International Journal of Applied Mathematics*, 51(2):1–11, 2021. pages 41

[68] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa. Planning-based prediction for pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3931–3936, 2009. pages 82