5-20-2019

Geometric Problems in Robot Exploration

Wyatt Preston Clements
Louisiana State University and Agricultural and Mechanical College

GEOMETRIC PROBLEMS IN ROBOT EXPLORATION

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Wyatt Preston Clements
B.S. in Web Dev., Thiel College, 2014
B.A. in Comp. Sci. and Math., Thiel College, 2014
August 2019

Acknowledgments

This thesis would not have been possible without a strong foundation of people who believed in me and supported me along this journey. I would like to extend my gratitude to my advisor Dr. Costas Busch. His faith in me set the stage for a career in theoretical computer science. He allowed me to pursue research of personal interest and included me in research groups to allow me the opportunity to expand me knowledge. This gave me the opportunity to work with Dr. Hsiao-Chun Wu who collaborated in the research group that lead towards my thesis topic.

During this time Dr. Busch has given me the opportunity to also gain the experience as a subreviewer for papers. He also offered me the ability to work as the Website Chair for the International Symposium on Distributed Computing (DISC 2018). This position helped me in learning time management as this began when I was working towards my defense. I am thankful for these opportunities that Dr. Busch has offered to me enriching my experience.

Thanks to Dr. Costas Busch, Dr. Hsiao-Chun Wu and Dr. Feng Chen for their willingness to be in my thesis committee. A special thanks goes to Dr. Xiaoliang Wan for being my Dean's Representative.

I would like to take this opportunity thank my family. My wife, Erica Clements, who has gone above and beyond to be the best partner during my college career. She has supported me and agreed to move with me to Louisiana in order to make this possible. To my siblings, Ethan, Aileen, and Garrott, for their constant love and support. My father, Wade, for always believing in me to accomplish my educational goals. To my mother, Charity, for assisting me in moving to Louisiana to continue my education.

This dissertation is dedicated to Erica A. Clements for her constant love, support, and encouragement.

Table of Contents

List of Figures

Abstract

Robots are increasingly utilized to perform tasks in today's world. This has varied from vacuuming to building advanced structures. With robots being used for tasks such as these, new challenges are introduced. Problems that have been previously researched to be performed, either theoretically or implemented, need to be redesigned to be able to better handle these challenges.

In this thesis, I will discuss multiple problems that have previously been researched and I have redesigned to be possible to be implemented by robots or that I have developed a new way for the robots to solve the problem. I focus on geometric areas and robots tasked with performing exploration in the area. Exploration is a task in which an unknown area is completely traversed. In this work, I have develop multiple algorithms to perform online tasks that can be implemented by robots. With robots performing exploration, a limited viewing range and communication range increase difficulty. These algorithms are focused on utilizing robots to perform Exploration and Concave Decomposition.

The results from this thesis are such that the Exploration algorithm that given a fleet of robots k, the total area n can be explored in $O(n/k)$ time with all agents having work $O(n/k)$. The Concave decomposition task has multiple algorithms focusing on a different aspect. In the first, with an online algorithm, with r reflex points, I perform the decomposition generating $r + 1$ convex areas. The other two online algorithms focus on interior cut length, which previously has not been researched. In response, have developed an algorithm which maintains interior length relative to the perimeter.

Introduction

Tasks previously developed to be implemented by humans are increasingly being redefined in order to be solved by robots. Utilizing robots introduces new challenges, battery life, random failure, communication range, as well as the traversal time.

Robots have an amount of time they can maneuver before they run out of battery. Therefore research with this focus either determines the minimum amount of movements or time bound, for instance, quad copter have a limited amount of time that they have the battery life to fly. Similarly, during the robot's maneuvering they can randomly fail or with sensor networks, a sensor can fail. It is therefore assumed that, if a group of robots are present, at least one will not fail at the end of the algorithm. This shortcoming must also be assumed for all sensors not failing that are placed by robots.

Communication range can vary and introduces different complexities. This can range from either through sensors or touching, up to long range communication. Through sensors, the robots place sensors in communication range to allow communication between robots through a chain of sensors. When robots are in close proximity to communicate, almost touching, techniques must be introduced for multiple robots to work together without repeating work. With long range communication, obstacles are introduced that interrupt communication such as in search and rescue.

Algorithms with robots now give a focus into the time it takes them to move. This requires describing the distance in which a robot can move in one "time step". For instance, if we are given traversing a polygon, traditional solution declares it takes the amount of vertices that make up the polygon but, with robots, it takes

the amount of time steps times the length of the perimeter. In this paper, I will introduce four problems I have worked on with a focus on robot exploration.

In Chapter 1, I research balanced exploration of orthogonal regions, where a fleet of robots, agents, are tasked with exploring the entire area when there exists holes in the area that cannot be traversed. The goal of this research is to develop an algorithm to explore an area in parallel while maintaining equal work. The difficulty is such that the holes are unknown and if there is no organization, the robots will duplicate the exploration of another agent.

Chapter 2 involves concave decomposition, with the goal of reducing the concave polygon into a minimal amount of convex areas. Convex areas can be utilized for guard duty in such a way that each guard can see all points in a convex area. To reduce the amount of guards, it is necessary to minimize the amount of convex areas generated.

The next two problems similarly solve the concave polygon problem but aim to reduce the length of the interior lines used to create the convex areas. This has not been previously researched, but with robot utilization, I aim to reduce the work performed by the robot. Chapter 3, works on rectilinear concave shapes. In this, the algorithm performs online convex decomposition. Continuing with Chapter 4, I generalize the algorithm in Chapter 3 to work on arbitrary concave polygons. This generalization allows me to compare the number of convex areas and cut-length to the results of Chapter 2.

Concluding each of these algorithms, in Chapter 4.2, I observe the results of each chapter and compare the results of the concave decomposition to compare the convex areas created to the interior cut-length. Finally, I will discuss future research goals that are related to this topic. These future goals involve expansions from these Chapters to solve further complex problems.

Chapter 1

Balanced Parallel Exploration of Orthogonal Regions

1.1 Introduction

In mobile agent research, there are two types of problems: coverage and exploration. Coverage, is concerned with patrolling an area, i.e. how to organize the agents so they can maximize the area they can view while still being able to be in contact range of the other agents in a chain [3, 13, 24]. This assumes that the agents already have full understanding of the area to be able to determine the best course of action to achieve these goals. On the other hand, exploration works with a previously unknown area, and attempts to map all of the area. Applications of area exploration is to find survivors quickly, setup a network, or to rendezvous distributed agents. Hence, exploration is a more fundamental problem which may also be used to solve coverage problems.

In order to explore an unknown area, previous papers have used sensor networks [6, 12, 18, 20, 26]. This style of exploration has the agents placing static sensors at intervals of the area to create a network and cover the explored area. The sensors then are able to be used by the agent to find the area boundaries and store

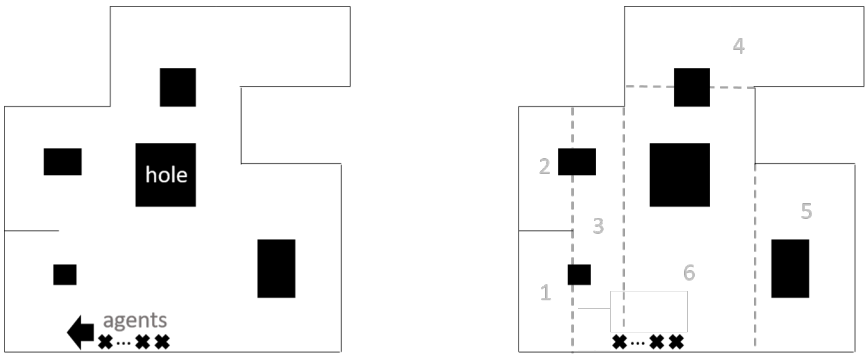

FIGURE 1.1. Example of Area and Generation of Rooms

information from previous scans of the robots that can help to explore new parts of the area. The agent itself has little memory and relies heavily on the sensors. The sensors have to be carried by the agents adding weight that inhibits the mobility.

In this paper, we present a set of algorithms in order to solve area exploration without using auxiliary static sensor nodes, but rather, with just self-reliant mobile agents. The area is an orthogonal closed region that is made up of horizontal and vertical walls. This area is then discretized into N traversable points that are connected to their neighboring points in the cardinal directions. An agent moves north, east, south, or west, on the neighbor points from its current position, whenever these points are available. The number N is not known to the agents. The agents discover the area and the number of points during exploration. The area contains holes, non-traversable locations, which are randomly distributed in the area; for example, Figure 1.1 depicts black holes in the area. These holes are rectangular and have a length and width within an aspect ratio a. The agents begin in a chain along the perimeter of the area, Figure 1.1 black x's.

All agents are assumed to be the same with equal traversal ability, communication range, and total memory. An agent occupies a single point at a time and is able to move one point in the cardinal directions in which the point has connection. We assume that it takes one time step to move point to point. The agents have enough memory to map the entire area (i.e. find coordinates of explored points) which can be shared between agents. To share area information, the agents must be within adjacent points, or in a connected chain in which there is a sequence of adjacent pairs of agents. Agents share information on their total area traversed as well as the orientation of the area. Through this process, agents are able to make decisions. These constraints on the agents enables them to not rely on sensors to have knowledge of the unexplored area. The work of an agent is the total number

of points it traverses. Each agent keeps track of the work it has performed, and the goal is to maintain balanced work among the agents.

The main goal is to explore an unknown orthogonal region of N points with k mobile agents in parallel in order to reduce the work, which is the total number of points visited, of each agent to explore the area to $O(N/k)$, namely, balancing the work and total exploration time $O(N/k)$. The difficulty with this is the introduction of holes in the area. If the holes were not there, we could simply traverse the entire perimeter, divide the area into k equal partitions, and send the agents to their respective area. This is not possible with holes, as one or more of these partitions could consist mainly of holes, so the agents associated with those partitions would not perform $O(N/k)$ leading to an imbalance of work between all agents. Thus, we developed an algorithm that handle the unknown holes in the area in order to maintain balanced work.

The difficulty with these holes is that they are arbitrarily placed in the area. As a sub area was closed, if we simply divided the area into k partitions, then we cannot guarantee that all the agents perform near equal work. For example, let's assume that the hole in a room of n points takes up all space except a single path around the area, then if partitioned, each agent encounters the same hole resulting to repeated work, which involves traversing around the hole and performing nk total work in the area instead of n total work. Moreover, as another example one agent may have to explore smaller number of holes (or total points) than other agents again resulting to imbalanced work.

In order to get the near equal work for each agent, we have introduced the concept of Leader, Active, and Inactive agent roles when an area is being closed (room created) and explored. These roles are determined based on the amount of work that was previously performed in other closed areas, the most, Leader,

the least, Active, and all the rest, Inactive. Leader organizes the other agents to maintain their overall work when Active return to the Leader. Active performs traversal in the closed area, returning to the Leader to maintain equal work. Inactive remain with the Leader until an Active returns sharing its work making them Active. When the closed area is completed the agents return to being against the perimeter to traverse to the next closed area.

In our algorithm, the agents start along all adjacent to each other next to the perimeter. Then they traverse the perimeter maintaining a convoy. During the traversal, the agents create rooms to explore and map. The whole area is divided on the fly into a sequence of rooms, which are explored one after the other. Each room is a rectangular closed area which is identified at corners of the perimeter. When a room is identified the agents move immediately into the room together and explore all the points of the room by dividing the work among each other. A room may contain obstacles which are not immediately detected, and hence, the area of the room may not be able to be divided equally among the robots, which may cause imbalance of the work. To alleviate this problem, the agents start the room exploration slowly, and use exponential delay and meeting times when exploring the room to handle the unknown holes in the room. Agents that exceed their allocated work quota give future work to other agents to maintain parallelism and load balancing.

1.2 Related Work

The most related previous work is solving a similar problem, utilizing "oriented disjoint rectangular obstacles" in an $n \times n$ grid [22]. In this, the agents also use only local information, but with the oriented obstacles, all obstacles follow an oriented pattern that allow the area to be divided into similarly shaped partitions. Our problem has arbitrarily located rectangular obstacles of ratio a. With this, our

problem or theirs cannot be reduced to the same problem. For this reason, a new algorithm had to be developed not only to traverse the entire area, as it does not follow their $n \times n$ grid rules with disjointed obstacles, and another algorithm for the room, as the obstacles do not follow the same constraints. Finally our focus is on balanced work as well as parallel while theirs is only on parallel work.

The most closely related topics in the literature are "map merging", concerned with multiple-robot simultaneous localization and mapping (SLAM) [25], and "map synchronization" [32], where the agents share the map information of the area explored. Agents are not assumed to be together and use probabilistic generalized Voronoi diagram in order to, "achieve fast and accurate map fusion for large maps"[25]. In map synchronization, agents can form into subgroups that are close enough to have the same map, while other agents that are not will synchronize maps when they approach [32]. However, these do not consider load balancing the work of the agents as we do here.

One topic of focus in this area is agents that are able to fail. This can be through "black hole" or "black link" in an edge or link in a sensor network [9], or the agent itself can fail [23]. In order to be able to explore the entire area, the number of sensors must be more than the number of edges plus two times the number of links [9]. With using "pheromones", Levy Walk was determined to outperform random walk when exploring, especially when failure prone agents are introduced [23].

Exploration is also a topic of interest in communication networks. Ad-Hoc networks are constantly changing, mobile agent routing leaves information at nodes it visits and also reads the information that is left at a node [29, 37]. In this way, the agents are able to cooperate much like ants leaving information to share with others when looking for food. When speed of network changes increases the delay decreases but roughly only 68% of packets are received and this amount drops

greatly as speed increases [29]. MAR [37] is able to perform at greater speeds with the same percentage of packets received but less delay.

Machine learning algorithms have also been used for mobile agent exploration. Q-Learning has been used as a "Human-like" learning algorithm to manage energy, avoid obstacles, working towards goals, and following a line [20]. Another implementation used online-learning Bayesian network to predict terrain range and magnetic field intensity [27]. A second online-learning method utilized Gaussian Processes [34]. In this learning, decentralized agents, flying quadcopters, are used to explore an area by mapping the distance of an unknown environment [34].

Agents have also been given roles or working with a stationary agent in order to explore an area. For search and rescue, agents are given a larger environment than what they can explore while staying in range so they must go out of communication range with agents being declared explorers or relays [10]. Area is shared from the explorer to the relay and then back to the command center [10]. Another case is when the agent must communicate with a computer when a obstacle is located so that the computer can determine the path to take [11]. The agents can also make a decision given multiple criteria to locate and rescue victims within a limited amount of time [5]. This MCDM model, used in search and rescue does not attempt to explore the whole map, but to use the knowledge of the agents exploration to move quickly to locations that the victims are believed to be, in this manner it finds the majority in the lest amount of time [5].

Similar to search and rescue, research has also been conducted for agents to rendezvous before exploration of the area [28]. In this research, agents look for "landmarks" to attempt to see if any other agents are within range of their sight or if they can find any other "landmarks" to search from there [28]. This research

does not focus on the exploration itself, but attempts to minimize the time for agents to rendezvous.

Constraints have been set on the power of agents, they are assumed to have limited power and must work together in order to find "targets" distributed in an unexplored area. The first strategy is to use a similar technique as ants, drop pheromones, to tell other agents this area has been explored, or there is a target in this direction [15]. This strategy does not require communication so it is able to save more energy for exploration. Next, the technique is to use communication between agents similar to how fireflies communicate [15]. The downside to this strategy is that it consumes energy to communicate. The more complex the task the better firefly approach performs compared to the ant approach.

1.3 Model

The problem we are exploring consider an orthogonal area on the plane. Through this, the perimeter edges all meet at right angles. There can also be interior lines within the interior which are extension of the edges from the perimeter. It is assumed that the whole orthogonal area can be discretized into grid points that make up the entire orthogonal area. We assume that grid points are spaced one unit distance apart. Each point is adjacent to those (up to four) neighbor grid points that are one unit distance in the cardinal directions (North, East, South, West).

An agent is dimensionless and resides in exactly one grid point at a time. Two agents directly communicate (they are in communication range with each other) if they reside in adjacent grid points (1 unit distance away). However, they can also indirectly communicate with each other through a chain of agents such that each pair along the chain reside in adjacent grid points. The agent visibility range extends only up to the four adjacent points. Agents can manoeuvre on the grid points moving from the current grid point to an adjacent grid point.

In the area there are rectangular holes such that no grid point exists in the hole making them non-traversable. These holes are unknown until discovered by an agent. Each hole in the area has an aspect ratio a such that a hole can be of any size as long as the width and length are bounded by the aspect ratio; that is, the width over length ratio is bounded by a, and length over width ratio is also bounded by a. Each hole must have at least two unit distance separating it from another hole or from any edge of the perimeter (or internal edge). In this way, at least one grid point fits between the holes or between a hole and an edge. To completely discover a hole, an agent must traverse around the hole, which is the only way to detect whether it is a hole or part of the perimeter.

A unit of work corresponds to visiting a grid point. To completely explore the area, all the grid points that make up the area must be visited and the information is shared by each agent. The agents have enough memory to map the entire area as well as recording the amount of work performed by all agents, both perimeter and room work. Agents share the information of what they have encountered, including holes and grid points, when they return to communication range. All agents are assumed to be identical; able to traverse a single grid point at each time step, communication range of one grid point, vision of one grid point to map the area surrounding the agent. Each grid point that an agent move counts as one unit of work. Multiple agents can all communicate with one another through a chain of agents.

During the perimeter work, agents move together maintaining a chain along the perimeter. In an attempt to close a room, the front agent will attempt to leave the perimeter to reach a previously explored grid point. If able to close, then the front agent returns to the other agents and perform the room work, otherwise, the front agent returns and they continue traversing the perimeter. After completing

room work, the agents determine, through their chain of communication, which agent has the least total perimeter work and that agent is moved to the front of the agents to lead the perimeter traversal.

When a room is encountered, then the agents must determine their role, Leader, Active, or Inactive. The one that is Leader is the one that has the most work, Active is the one that has the least work, and Inactive are all those in between. Leader communicates with the Active as they return to share/give their work to Inactive to maintain equal room work. Active traverse the area, after a set amount of work if there remains work in the room, then the Active returns to Leader to determine what is to be done next. Finally, Inactive wait with the Leader to be given work that remains from the Active. Active agents know the location of the Leader and Inactive agents to return to communicate with the Leader. When the Active gains more work than the Leader, the Active and Leader will change roles. Once the work is completed all agents will return to the Leader to organize into a line along the perimeter.

This process of performing perimeter work until a room is discovered and then performing room work is repeated until all points are traversed.

1.4 Algorithms

As mentioned in the introduction, the exploration algorithms can be separated into two parts, perimeter exploration and room exploration. In perimeter exploration the area is divided into rooms. Agents maintain a track of the work they perform during perimeter and the work they perform during room exploration separately. During perimeter work, agents will rotate the agent with the lowest perimeter work after a room is attempted to be closed to maintain balanced perimeter work. With room work, the agent with the most work will be in charge of coordinating the

room work between other agents and transferring the responsibility to a new agent with more work whenever necessary.

Perimeter Exploration

Algorithm 1: PerimeterScan

```
   /* Algorithm for agents to scan around the perimeter                    */
 1 Each agent tracks its work done during perimeter scan;
 2 All agents begin together and parallel to exterior wall;
 3 Explored perimeter points P ← ∅;
 4 while A perimeter point δ is left to be scanned (i.e. p ∉ P) do
 5     Agents move together as a convoy keeping a line adjacent to the exterior
         wall they're against;
 6     The front of the convoy will be on top of δ; P ← δ ∪ P;
 7     if convoy front agent performs a turn at δ = (X_t, Y_t) or there is an unused
         corner perpendicular to traversal direction then
           // Check possible directions for closing room
 8         RoomPoints ← the set of closest perimeter points (X, Y) ∈ P such
             that either X = X_t or Y = Y_t;
 9         Remove from RoomPoints all the points (X, Y) such that the straight
             line from (X_t, Y_t) to (X, Y) goes through a previously visited hole or
             wall that is not in P;
           // Attempt to close a room
10         for δ' ∈ RoomPoints do
11             WallHit, Sucess ← false;
12             ℓ ← convoy front agent;
13             while not (WallHit or Success) do
14                 Agent ℓ moves one point closer towards point δ';
15                 if agent ℓ has reached δ' then
16                     Success ← true;
17                 if obstacle encountered then
18                     WallHit ← ObstacleCheck(δ, δ');
19             Agent ℓ returns to δ;
20             if Success then
21                 RoomScan(δ');
22                 mark corner as used;
23             Change front agent of convoy to that one with smallest perimeter
                 scan traversal;
```

The first algorithm performs the exploration of perimeter P (Algorithm 1) . When the exploration begins, the agents all agree on a direction to begin traversal,

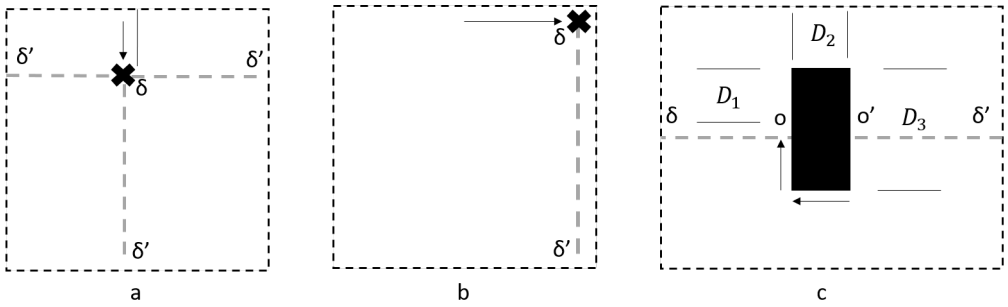

FIGURE 1.2. Possible Room Closure Directions

depicted in Figure 1.1 by the black arrow. The agents follow the front agent in that direction maintaining a convoy along the perimeter of area, adding explored points to P. If the front agent is no longer against the perimeter or they have reached a perimeter wall along the path they notify the other agents that a corner has been reached at point $\delta = (X_t, Y_t)$ or that there is an unused corner perpendicular to the traversal direction. At this point, before moving along the path, if a corner is encountered then the front agent looks at it's mapping of the area to determine if they have already visited a perimeter point across, namely, δ' such that $\delta' = (X, Y)$ where $X = X_t$ or $Y = Y_t$ and (X, Y) is in the visited P. The agent can find possibly up to three candidate δ' depending on exploration history of P, as shown for example in Figures 1.2.a or 1.2.b. The front agent then chooses one of the directions and begins moving towards the picked δ' to determine if a room closure is possible. The front agent continues along a straight path until reaching the previously explored position δ', as for example depicted in room 1 generated in Figure 1.1 where no obstacle is encountered. Otherwise, as depicted in Figure 1.1 in the generation of room 3, if the agent detects an obstacle along its path then it must determine if the obstacle is a hole or a wall, which is determined by calling the obstacle subroutine Algorithm 2.

The front agent returns to δ if an unexplored wall was hit or the agent successfully reached δ'. After returning, if δ' was reached a room has been found, and then room scan is called and the corner is marked as used. After performing room scan, or hitting a wall and returning, the front agent of the convoy switching with the agent that has the least amount of perimeter work completed, in order to balance the work in the perimeter. After all the δ' have been explored for that δ, the agents perform the turn and continue scanning the remainder of the perimeter.

Algorithm 2: ObstacleCheck(δ, δ')

```
/* Algorithm to check if perimeter wall hit when closing a room
   */
```
$Dist \leftarrow |X_t - X| + |Y_t - Y|$, where $\delta = (X_t, Y_t)$ and $\delta' = (X, Y)$;
$o \leftarrow$ point obstacle encountered;
Move to corner of obstacle (wall or hole), in the direction point of previous than δ in list P;
$D_1 \leftarrow$ points traversed;
if $D_1 > a \cdot Dist$ **then**
    return *true*;
Turn along obstacle;
Move to next corner of obstacle;
$D_2 \leftarrow$ points traversed;
if $D_1 > a \cdot D_2$ **then**
    return *true*;
Turn along obstacle;
Move along obstacle until $a \cdot (D_2 + 1)$ points have been traversed or corner is reached;
$D_3 \leftarrow$ points traversed;
$o' \leftarrow$ point along δ and δ' found during traversal of D_3;
if $D_3 < a \cdot D_2$ *or* $D_2 < a \cdot D_3$ **then**
    Attempt to return to o along points not yet traversed;
    if *obstacle is rectangular* **then**
        Move back to o';
        return $false$;
return *true*;

The obstacle subroutine Algorithm 2 determines if the obstacle encountered is a hole or a wall. Let the point that the obstacle is encountered be o. The agent

checks that the first length of the obstacle D_1 (see Figure 1.2.c), perpendicular to the line l between δ and δ', is shorter than the aspect ratio a multiplied by the length of l which is $Dist = |X_t - X| + |Y_t - Y|$. If this value is exceeded, than the obstacle cannot be a hole that fits between δ and δ'. Continuing from this point, the agent checks each dimension to make sure that the obstacle is a hole. It checks if both of the lengths D_2 and D_3, Figure 1.2.c, are within the a aspect ratio, first between D_1 and D_2 and finally between D_2 and D_3. During the exploration to D_3 the agent will pass o' such that o' is on the line from δ to δ'. After this point, it is known that the obstacle adheres the aspect ratio restrictions of a hole, but the agent must make sure that it is rectangular by returning to o around the points that are between D_1 and D_3 that have not been explored. If it is indeed a hole, the agent adds in its memory as a known hole, a hole that is detected in the area, and returns to o'.

Room Exploration

Once a room is found, the room scan is invoked. After front agent in perimeter scan successfully reaches δ' and returning to the other agents, the agents determine their roles beginning room scan. The agents are set to the roles of leader, active, or inactive. The purpose of the leader is to maintain a ratio of work between all agents to balance the work performed by each agent. The active agent is given a task in the form of a stack of areas to explore, and performs the exploration until a work threshold is reached. The inactive agents are in a convoy by the leader waiting for the leader to give them a task and become active. The allocation of the work to the agents is performed in a balanced way. If this is the first room attempted to close then no agent has any room work so it is arbitrary which agent is set to leader, active, or inactive. Otherwise, the agent with the most room work is set to the leader, yellow 'x' in Figure 1.3, the one with the least room work is

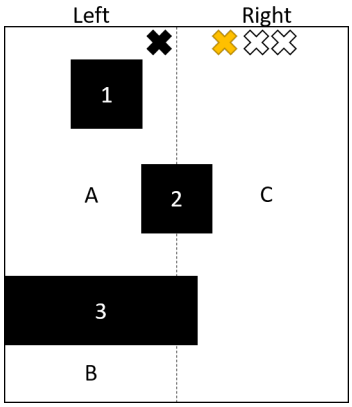

FIGURE 1.3. Splitting a Room with Holes

set to active, black 'x' in Figure 1.3, and the rest begin as inactive, white 'x' in Figure 1.3.

The work threshold of an active agent is initially set to a value d_{max} that is determined in more detail in Section **??** where we analyze the algorithm. As the room exploration progresses, the work threshold exponentially increases to accommodate the expected work in the newly discovered areas. Gradually, inactive agents are becoming active as more area is discovered in the room. The work of the newly discovered areas is evenly distributed among the nodes. The leader subroutine (Algorithm 3) describes the leader's role as active agents return to the leader. The leader's job is to maintain the remaining work between itself and the active and inactive agents. When an agent i has performed work that exceeds its threshold or it ran out of tasks to perform, it returns to the leader. If the agent i has a stack remaining, or the leader has a stack, then the leader takes agent i's stack and merges it with its own. The leader must first make sure that the total work performed by agent i has not exceeded a ratio 3 between the leader and agent i, if agent i did exceed this ratio, then the leader and agent swap roles and the leader begins the Active part of RoomScan. If at least one other inactive agents j is waiting with agent i then the leader must check the ratio of work between agent

Algorithm 3: Leader part of RoomScan

```
/* Algorithm performed by each Leader until room is entirely
   explored                                                    */
Upon (agent i returning to meeting point with Leader)
set agent i to inactive;
if agent i has a stack or leader has a stack then
    if agent i has a stack then
        Leader takes stack of newly discovered areas from agent i and merges
         with current stack;
    if agent i's work exceeds ratio 3 above Leader's work then
        Leader and agent i role swapped;
    else if more than one agent is waiting without a stack then
        if agent i's work is within ratio 4 of all other agents' work which is at
         least 1/4 the initial traversal of i then
            Leader splits stack into two parts of equal size;
            Leader gives one part back to agent i;
            Other part is given to an inactive agent j with lowest work;
            Reset agent i work threshold to minimum;
            Signal agents i and j to get active;
        else
            Give stack to inactive agent with lowest work;
    else if agent i within ratio 4 of all other active agents then
        Give stack to agent i;
        Signal agent i to get active;
```

i and all inactive agents j to decide what to do with the stack. If agent i's work is within a ratio 4 of all other agents j then the leader splits the stack in half, giving one part back to agent i and the other to the agent with lowest total work. After this, the leader resets the threshold back to the minimum for agent i and signals agents i and j to become active. Otherwise, if inactive agent(s) j are waiting and i has exceeded the ratio, then the entire stack is given to the agent with lowest total work. Finally, If no other agent is inactive and agent i is within a ratio of 4 between all the active agents, then the stack is given back to agent i and agent i is signaled to become active. The active subroutine (Algorithm 4) describes the

Algorithm 4: Active agent i in RoomScan

```
/* Algorithm performed by each agent i after being activated by
   Leader in RoomScan                                          */
// Initially work threshold is set to minimum by leader
```

1 Upon(Signal received from Leader)
2 **while** *Area.top && NewWork* $<$ *workthreshold* **do**
3 CurArea = Area.pop;
4 $SmDim = \min(CurrArea_x, CurrArea_y)$;
5 $LgDim = \max(CurrArea_x, CurrArea_y)$;
6 Move to middle of $LgDim$;
7 Determine the path to be taken and the amount of areas, $area_1, \ldots, area_i$ to be created given known holes along the path;
8 Move along path to other appending $NewWork$ end of area;
9 If a hole is encountered agent i moves around to remain in their half of $LgDim$ returning to path;
10 Without loss of generality, assume Leader is top right of room and initial split is vertical;
11 Push areas onto stack from right to left;
12 **for** *each right rooms* **do**
13 Push farthest from leader first
14 **for** *each left rooms* **do**
15 Push closest to leader first
16 Similar stack construction for horizontal split and other locations of Leader;
17 Double work threshold and go to the leader position;

active agents' role as they are given a task by the leader. The agent i tracks their

current work, and their current threshold of work to determine when to return to the leader. While the agent has task(s) to accomplish and the new work they have completed has not exceeded the current work threshold the agent will take a task to accomplish. Agent i then takes the current task area and determines what are the smaller and larger dimensions of this task area. Then, agent i moves to the middle of the larger dimension and determines how many sub areas will be generated by traversing across the line that splits the task area. The sub areas are determined by the amount of holes encountered through the traversal line.

For example Figure 1.3.3 shows the vertical traversal line that goes through the room and splits its across the hole boundaries (holes labeled 1, 2, 3) into three subareas A, B, and C. As shown in Figure 1.3, agent i does not have to change its traversal due to hole 1 as this hole is at least one point away from the traversal line. Agent i will have to change its traversal, due to hole 2 as this is along the traversal path and unknown to agent i. As Figure 1.3 has the leader in the top right and the split is vertical, the agent pushes areas onto the stack from the right to the left so that the area furthest from the leader is the next task to be explored. For each room on the right of the split the agent i pushes the furthest from the leader on the stack, from Figure 1.3 only C is on the right. Then, agent i begins pushing the rooms on the left from the closest to the furthest from the leader, Figure 1.3 first A then B. This process is similar for other locations of the leader and split type, in order to maintain a stack of connected areas with the top being the furthest from the leader on the opposite half of the area.

When the work threshold was reached and agent i still has a stack of task(s), it doubles the work threshold and goes back to the leader. The purpose of doubling the work threshold is such that if no other agent was there to share work, then the other agents j are all active and the agent i should perform more exploration

before going to the leader again. As all other agents j are active, agent i is able to perform double the work and maintain the ratio 4 between all other active agents j. If at any point agent i has no more tasks to do then agent i returns to the leader.

1.5 Analysis

We continue now with the analysis of the algorithms. We have separated the two types of agents' actions, perimeter and room scan, to clearly discuss the work performed during each algorithm. Section 1.5 goes though the proof for perimeter scan and obstacle check. Section 1.5 will deal with room scan and the actions of the leader and other agents (active/inactive). Finally in Section 1.5, we combine the analysis for the works completed during all algorithms to prove the total work performed by each agent is $O(N/k)$ and total time is $\Theta(N/k)$ where N is the total number of grid points in the orthogonal region.

Perimeter Scan

To analyze these algorithms we begin by proving the completeness of perimeter scan. We show that the algorithm performs a post-order traversal of all the rooms through converting the sequence of steps as rooms are generated into a tree structure. From here, we analyze the total amount of work that has been completed by each agent during the perimeter scan and obstacle check algorithms.

Sequence of Steps

Perimeter scan can be broken down into into a sequence of steps $S = \sigma_1, \sigma_2, \ldots, \sigma_z$, such that each σ_a is a grid point $\sigma_a = (x_a, y_a)$, and any two consecutive points in the sequence differ in only one dimension, namely, either $|x_a - x_{a+1}| = 1$ and $y_a = y_{a+1}$, or $|y_a - y_{a+1}| = 1$ and $x_a = x_{a+1}$, for $1 \leq a < z$. From this sequence, three types of subsequences occur: Wall, Corner, and Cross.

- Wall: subsequence of steps $s_a, \ldots, s_{a+b}$, where $j \geq 2$, such that either the x-coordinate is the same ($x_a = x_{a+1} = \cdots = x_{a+b}$), or the y-coordinate is the

same ($y_a = y_{a+1} = \cdots = y_{a+b}$), that is, the subsequence of steps follow along the same x or y coordinate.

- Corner: Is a single point σ_a, $1 \leq a \leq z$, that changes from the dimension of the subsequence path, either $|x_{a-1} - x_a| = 1$, $y_{a-1} = y_a$ and $|y_a - y_{a+1}| = 1$, $x_a = x_{a+1}$ or $|y_{a-1} - y_a| = 1$, $x_{a-1} = x_a$ and $|x_a - x_{a+1}| = 1$, $y_a = y_{a+1}$.

- Cross: a subsequence of closing a room from Corner or a Wall to another Wall or Corner. Namely, it is a subsequence after a Corner or Wall, $\sigma_a, \ldots, \sigma_{a+b}$ such that σ_{a+b+1} is a recently visited grid point and σ_a and σ_{a+b+1} share either x or y value ($x_a = x_{a+b+1}$ and $y_a \neq y_{a+b+1}$ or $y_a = y_{a+1}$ and $x_a \neq x_{a+b+1}$). The other points will differ in one point in the x or y direction such that $x_i - x_{i-1} = 0$ and $|y_i - y_{i-1}| = 1$ or $x_i - x_{i-1} = 1$ and $|y_i - y_{i-1}| = 0$ as the agents could have encountered holes along the Cross. Algorithm 1, when closing a room ends in *Success* (Lines 11-25).

Special Subsequences and Rooms

Define a new sequence $Q_0 = q_1, q_2, \ldots, q_l$, where each element $q_i \in Q_0$ is a subsequence of S, representing either walls, corners, crosses, that is, $q_i.type \in \{Wall, Corner, Cross, Empty\}$. First three type have an attribute: for $Wall$, $q_i.attribute \in \{s_x, s_y, e_x, e_y\}$, where the start point is (s_x, s_y), and the end point is (e_x, e_y); for $Corner$, $q_i.attribute \in \{x, y, Turn\}$, where (x, y) is the corner coordinate, and $Turn$ denotes the direction of the turn which is either left or right; and for $Cross$, $q_i.attribute \in \{s_x s_y, e_x, e_y\}$, where start point is (s_x, y_x) and the end point is (e_x, e_y).

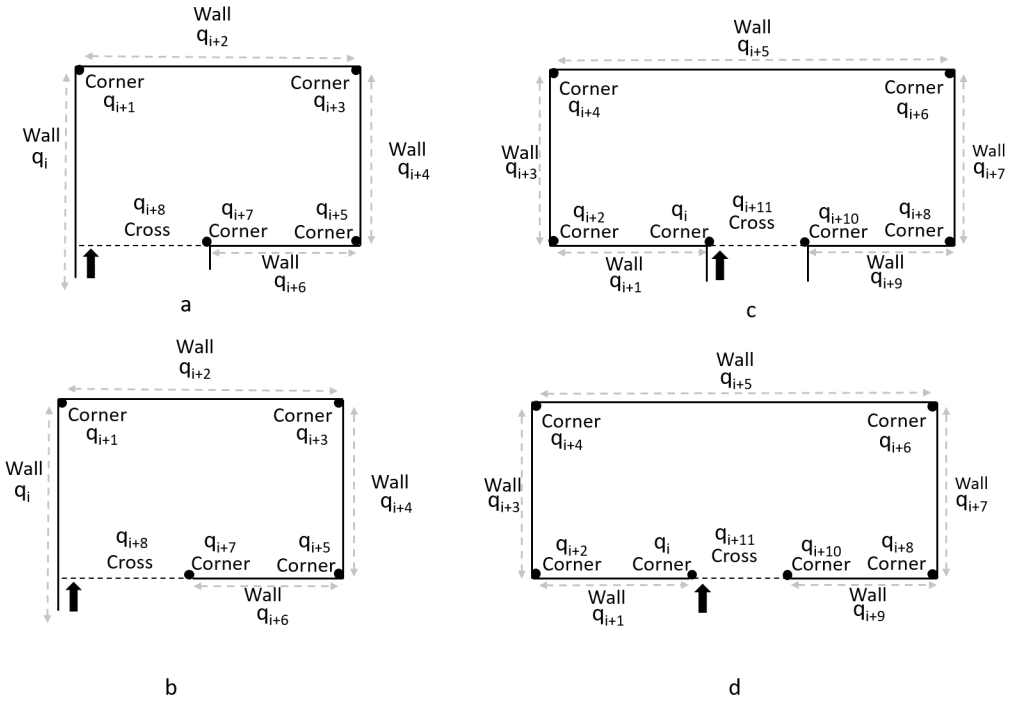

FIGURE 1.4. Two Basic Room Types

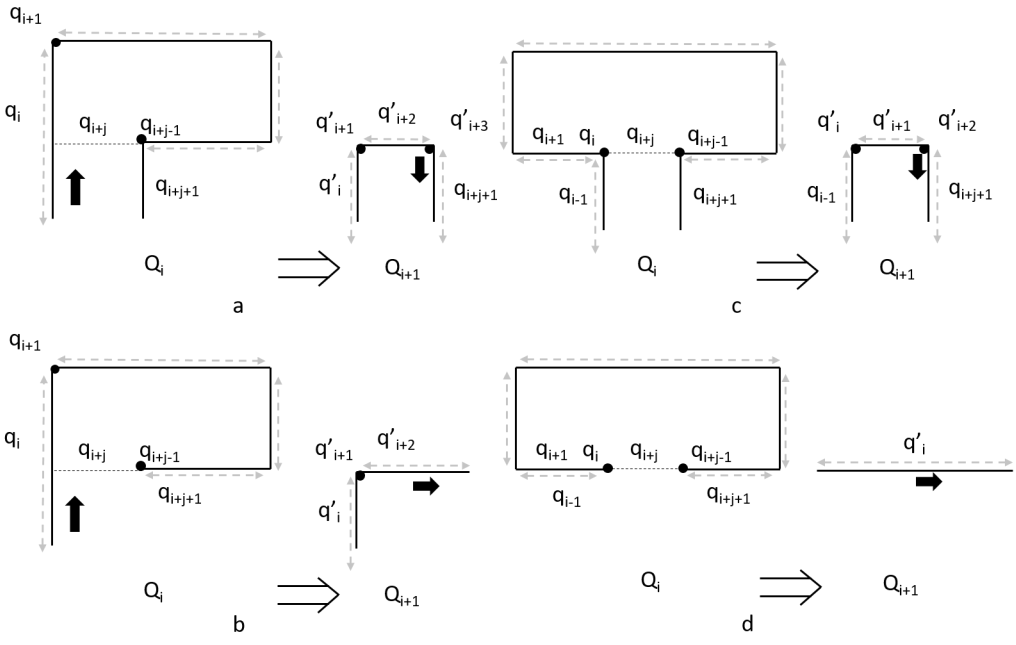

FIGURE 1.5. Two Basic Transformations After Room Scan

TABLE 1.1. Basic Possible Values of q'

	Figure 1.5.a	**Figure 1.5.b**
q'_i	$q'_i.e_y = q_{i+j}.e_y - 1$ $q'_i.s_y = q_i.s_y$ $q'_i.s_x = q_i.s_x$	$q'_i.e_y = q_{i+j}.e_y - 1$ $q'_i.s_y = q_{i-1}.s_y$ $q'_i.s_x = q_{i-1}.s_x$ $q'_i.e_x = q_{i-1}.e_x$
q'_{i+1}	$q'_i.x = q_{i+j}.e_x$ $q'_i.y = q_{i+j}.e_y - 1$	$q'_i.x = q_{i+j}.e_x$ $q'_i.y = q_{i+j}.e_y - 1$
q'_{i+2}	$q'_i.s_y = q_{i+j}.e_y - 1$ $q'_i.e_y = q_{i+j}.s_y - 1$ $q'_i.s_x = q_{i+j}.e_x$ $q'_i.s_x = q_{i+j}.e_x$	$q'_i.s_y = q_{i+j}.e_y - 1$ $q'_i.e_y = q_{i+j+1}.s_y$ $q'_i.s_x = q_{i+j}.e_x$ $q'_i.e_x = q_{i+j+1}.e_x$
q'_{i+3}	$q'_i.x = q_{i+j}.s_x$ $q'_i.y = q_{i+j}.s_y - 1$	

In closing a room, a room starts with a $Wall$ or $Corner$, and ends with a $Cross$. A $Wall$ is followed by a $Corner$ or a $Cross$ if an unused $Corner$ is perpendicular to $Wall$, and a $Corner$ is preceded by a $Wall$, if there is no possible room closure directions or if they all had exterior walls along the path, or a $Cross$ if there was a successful room closure direction. In Figure 1.4, part a begins with a $Wall$, q_i, and repeats until the $Corner$ is reached that reached that has the previously explored point in q_i that was successfully reached by the front agent, where as in part b the room begins with a $Corner$, q_i.

Replacing Rooms and Sequence Transformation

We can define a sequence of subsequences $Q_0, Q_1, \ldots, Q_r$. Each pth subsequence is generated by the $p-1$ item in the sequence, so that subsequence Q_p is generated after room discovered in Q_{p-1}. Subsequence Q_p replaces the subsequence of Q_{p-1}: $q_i, q_{i+1}, \ldots, q_{i+j}$, that made up room R_p with a new subsequence that transforms

the $Cross$ into $Wall$(s) and $Corner$(s) depend on how the room was entered and the holes along the $Cross$.

This new subsequence is $q'_i, q'_{i+1}, \ldots, q'_{i+j}$ such that j is the maximum number of subsequence elements. In Figure 1.5.a the sequence in Q_i, $q_i, q_{i+1}, \ldots, q_{i+j}$ is replaced in Q_{i+1} to $q'_i, q'_{i+1}, q'_{i+2}, q'_{i+3}$, the attributes developed from the attributes of q_i, q_{i+j}, and q_{i+j-1}, see for example Table 1.1. Similarly, Figure 1.5.b transforms from q_i, q_{i+j}, and q_{i+j-1} to q'_i, q'_{i+1}, q'_{i+2}. As these are only two basic room types, without any holes, other closures will have multiple q' created for q_{i+j}, the $Cross$, as with the introduction of holes the $Cross$ will transform into multiple $Wall(s)$ and $Corner(s)$ as it is no longer a traversal along a single dimension.

For Q_r, the last room in the region, as $q_i, q_{i+1}, \ldots, q_{i+j}$ will be all elements of Q_{r-1}, Q_r replaces the subsequence of Q_{r-1}, $q_i, q_{i+1}, \ldots, q_{i+j}$ that made up room R_{r-1} with $q'_i.type = Empty$ signifying that the region is explored in its entirety.

Conversion to Tree

Given Q_i is a sequence of steps q_i that discovers a room $i+1$, traversing the Discovery of rooms in reverse, $Q_r, Q_{r-1}, \ldots, Q_1, Q_0$, a Tree T of rooms can be generated (see Figure 1.6.b). The root of T is room R_r (room R_3 in Figure 1.6) which is the last room to be discovered and corresponds to the room where the agents started the whole discovery process. Suppose that R_r corresponds to sequence $q_1, \ldots, q_l$ in Q_{r-1}. Room R_{r-1} was generated from Q_{r-2} and is included in $q_1, \ldots, q_l$.

The children of R_i are the rooms R_h such that R_h is in the sequence of R_i in Q_{i-1} between the points q_i and q_{i+j} that generated R_i (see Figure 1.6.c). Given multiple children of R_i they are added to the tree from left to right following their ordering in the sequence. If a room R_h does not have any room sequence between it's respective q_i and q_{i+j}, then it is considered to be a leaf of the tree. Following these rules we generate a tree structure, with R_r as the root, as as we work backwards

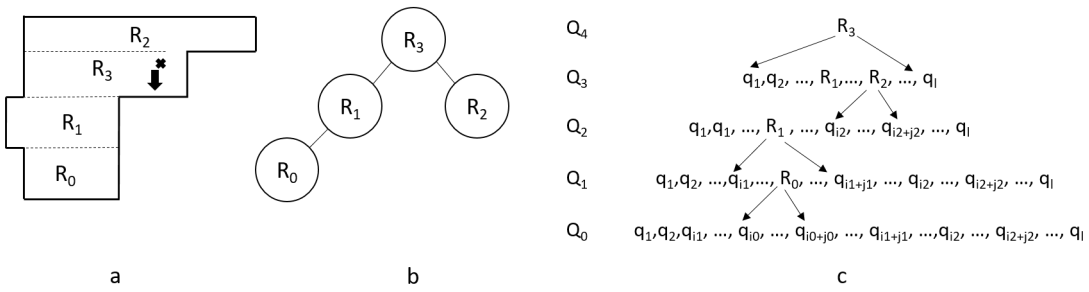

FIGURE 1.6. Conversion from Rooms to Tree Structure

from the room discovery, Figure 1.6.c, adding the respective children of each room as it is expanded from Q_i to Q_{i-1} until we return to Q_0.

Consider two rooms R_p and R_t that in Q_0 have the corresponding subsequences $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$ and $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$ respectively. The sequences of R_p and R_t in Q_0 are either *disjoined*, when $q_{i_{0p}} > q_{i_{0t}+j_{0t}}$ or $q_{i_{0p}+j_{0p}} < q_{i_{0t}}$, or *contained*, when $q_{i_{0p}} > q_{i_{0t}}$ and $q_{i_{0p}+j_{0p}} < q_{i_{0t}+j_{0t}}$ or $q_{i_{0t}} > q_{i_{0p}}$ and $q_{i_{0t}+j_{0t}} < q_{i_{0p}+j_{0p}}$, or *connected*, when $q_{i_{0p}} = q_{i_{0t}+j_{0t}}$ or $q_{i_{0t}} = q_{i_{0p}+j_{0p}}$.

Lemma 1. *Any pair of connected rooms, with neither being the root, have the same parent.*

Proof. Let's assume that R_p and R_u are two connected rooms. Let R_t be the parent R_p. Since R_t is the parent of R_p we have that R_t is the first room that contains R_p, we have $q_{i_{0t}} < q_{i_{0p}}$ and $q_{i_{0p}+j_{0p}} < q_{i_{0t}+j_{0t}}$. Also, from connected we know that either $q_{i_{0p}} = q_{i_{0u}+j_{0u}}$ or $q_{i_{0u}} = q_{i_{0p}+j_{0p}}$. Thus, $q_{i_{0t}} < q_{i_{0u}+j_{0u}}$ or $q_{i_{0u}} < q_{i_{0t}+j_{0t}}$. Therefore, from our projections R_t and R_u are not disjoined or connected, namely, one contains the other.

Suppose that $t < u$. Therefore, R_u contains R_t. Since R_t contains R_p, we have that R_u contains R_p, which is a contradiction, since R_u and R_p are connected. Therefore, $t > u$. Consequently, R_t must also contain R_u.

Suppose that there is another room R_γ that is the parent of R_u. Therefore $q_{i_{0\gamma}} < q_{i_{0u}}$ as rooms u and p are connected then γ contains p, $q_{i_{0\gamma}} < q_{i_{0p}}$ As t is the parent of p, t must be contained in γ. This results in a contradiction because t contains u and is a descendant of γ From this, R_t is the parent of R_p and R_u. □

If the room discovered in Q_{r-1} was not the room the agents began the exploration in, then there are two connected rooms that are the roots of two trees.

Lemma 2. *For disjoint rooms R_p and R_t where $p < t$, the subsequence that generates R_p must appear earlier than R_t in Q_0.*

Proof. Let's assume that it is possible for R_t subsequence, $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$, to appear earlier than R_p subsequence, $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$, in Q_0; $q_0, \ldots, q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$, $\ldots, q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}, \ldots, q_l$. As traversal is from left to right, at some Q_c such that $c < p$ the subsequence, $q_{i_{0t}}, \ldots, q_{i_{0t}+j_{0t}}$ would be traversed and then at Q_p the subsequence $q_{i_{0p}}, \ldots, q_{i_{0p}+j_{0p}}$ would be traversed. Thus, it is not possible for an earlier subsequence that makes up a disjoined room to be explored after a later disjoined room. □

All points will be covered by agents after the entire perimeter is traversed. Given Lemma 1 and Lemma 2 the whole area is divided into room which are explored in a post-order traversal of T. As we see from Figure 1.6.*a*, from the agent's start position and traversal direction, the agents discover room $0, 1, 2$, and 3 which after generating the tree structure, Figure 1.6.*b*, traverses the left subtree, then the right subtree, and finally the root, a post order traversal.

Theorem 1. *The area is divided into rooms through the traversal of the perimeter. Following in reverse order create a tree T of rooms which are explored following post order traversal of the tree.*

Perimeter Traversal

During perimeter scan, all agents will perform the traversal along the perimeter of the outer wall. When agents reach a corner that is along a previously explored point, the front agent will attempt to close the room and return. If an obstacle is detected then that agent will try to determine if it is a hole, given the aspect ratio, if it is then that agent continues to attempt to close the room. The agent returns to other agents if it is a room or runs into a wall. From there, if it is a room, room scan will begin, otherwise the agents will continue perimeter scan. Before continuing perimeter scan the agent that has the least amount of perimeter work completed, such that it hasn't attempted to close a room or has had the smallest attempt of all agents to close a room, switches places with the front agent.

Lemma 3. *A shortest path of adjacent points in the orthogonal region between any two points on a single dimension with Euclidean distance C from each other, has a maximum length of $C + 2aC$.*

Proof. Without loss of generality, let us assume that the start and end points are on the same horizontal line. Then we can form a path which traverses from the start to end consisting of several path segments:

$$\sum_a p_a + \sum_b w_b + \sum_c g_c.$$

Each p_a are the lengths of horizontal path segments on same line as start and end, w_b are the lengths of horizontal path segments along encountered holes parallel to the line between start and end, g_c are the lengths of vertical path segments of encountered holes that are perpendicular to the line between start and end. There are two g_c for every one w_b as there is a g_c connecting p_a to w_b and g_{c+1} connecting w_b to p_{a+1}. Due to holes being rectangular, the sum of $\sum p_a + \sum w_b = C$.

We will continue to show $\sum g_c \leq 2a \sum w_b$. As a is the aspect ratio of the hole dimensions, then for each hole length w_b we can assume the longer length is the associated g_c and g_{c+1}, meaning that at most these can be aw_i larger. Thus, since each w_b has two associated g_c, $\sum g_c \leq \sum 2aw_b$. As stated above, $\sum p_a + \sum w_b = C$ then, since $\sum w_b \leq C$, $\sum g_c \leq 2aC$, thus, $\sum p_a + \sum w_b + \sum g_c \leq C + 2aC$.

□

Suppose that the perimeter, not including *Crosses* attempted, up to discovery of room R_i has P_i traversable points. Each of these P_i points is traversed by all k agents. The total traversable points is N_{it}, then orthogonal region up to room i has a total interior traversable points N_i such that $N_i = N_{it} - P_i$ where we remove the perimeter points from the total points in the area.

Lemma 4. *Total perimeter work from all agents performed is* $O(kP_i + N_i)$.

Proof. As the perimeter is traversed by all agents, k agents perform P_i movements to traverse all points along the perimeter. During a *Cross* only the agent with the least perimeter total work will gather work. Holes and walls discovered by an agent during a *Cross* are known by all other agents when the crossing agent returns. Agents do not attempt to close a room if a previously discovered hole or wall is located along that path. For this reason there are two opportunities to traverse a position in the room, horizontal and vertical crosses, and as such, 4 traversals over any position in both directions of the Cross. Thus the total amount of work performed during is kP_i for perimeter movements and $4N_i$ for the attempted crosses. □

From Lemma 3,there is a *Cross* with maximum Euclidean distance $Cross_{\max}$ corresponds to a maximum path distance $C_{\max} = Cross_{\max} + 2aCross_{\max}$, when

moving around holes. $C_{\max}$ is the largest distance that any agent traverses during the traversal of its assigned Cross.

Lemma 5. *Difference in perimeter work between any two agents is* $O(C_{\max})$.

Proof. All agents perform the perimeter traversals, so the difference in work appears when Crosses are performed, as these are done by a single agent. Agents swap after completing a Cross if another agent has less work to allow them to perform the next Cross. Let us assume that an agent could exceed $C_{\max}$ work compared to other agents. For two agents i and j with respective works W_i and W_j, if $W_i > W_j + C_{\max}$ then $W_i > W_j$ before the Cross was taken, as agent i could gain at most $C_{\max}$ new work. This contradicts the algorithm as the agent with the lower work would perform a Cross. Hence, all agents are bounded by a difference in work of $C_{\max}$. □

Lemma 6 (Perimeter Work)**.** *If* $N_i \geq 4C_{\max}k$*, the perimeter work of each individual agent is* $O(P_i + N_i/k)$.

Proof. From Lemma 4, we have that the total work is $O(kP_i + N_i)$ and Lemma 5 proves the max difference between any two agents is $O(C_{\max})$. As agents are within $C_{\max}$ work of one another, each agent must have work at most $(kP_i + 4N_i)/k + C_{\max} = P_i + 4N_i/k + C_{\max}$. With the assumption that $N_i \geq 4C_{\max}k$, this can be simplified to $P_i + 4N_i/k + (1/(4k))N_i$ which is $O(P_i + N_i/k)$. □

Lemma 7 (Perimeter Time)**.** *With total number of Corners* Γ *the total perimeter time is at most* $P + 6\Gamma C_{\max}$.

Proof. Each agent follows along the perimeter moving one position each time step, given P perimeter points, takes P time to visit each perimeter position. At every *Corner* we have at most 3 directions that a room can be generated with a maximum

of $C_{\max}$ distance to traverse. An agent must traverse the closure in both directions, thus $6\Gamma C_{\max}$ to traverse possible room closures for all $Corners$ Γ. This creates an upper bound on the total time to finish the perimeter scan portion $P+6\Gamma C_{\max}$. □

Room Scan

We continue with the work analysis of the agents during room scan. In our algorithm, as the perimeter scan progresses new rooms are being discovered, and the each room is explored separately in room scan by the work performed by the active agents and controlled by the leader. For balancing the work, the robots keep a separate count of the total work performed during room scan. The first room discovered is a special case, as there are no previous rooms or room work. Hence, we start with analyzing the work completed by each agent in the first room. From this point, we can now describe the relationship between all agents work throughout each subsequent room exploration to explain total work of all agents together and work performed by each agent individually after the completion of each separate room. Finally, after the last room we are able to analyze the total work and work completed by each agent at the end of the whole area exploration.

First Room Exploration

Consider the first closed room and suppose that it is of size $L_{long} \times L_{short}$, where the length of L_{long} is at least as large as L_{short}. Suppose also that there is a total of n traversable points in the room. The aspect ratio of the holes in the room is a. Without loss of generality, we will assume that L_{long} is horizontal, and L_{short} is vertical. All agents currently have 0 initial work. Before, we proceed with the work analysis, we need bound the longest shortest path in the room, which will be later used to bound the difference on the works of different agents.

Lemma 8. *The maximum shortest path between any two points in a room has length* $D = O(aL_{long})$.

Proof. A shortest path between any arbitrary pair of points in the room can be formed by connecting two subpaths h and v, the first is horizontal of length H, and the second is vertical of length V, to reach the other point.

There can be a hole along the two subpaths. This creates key positions in the path pos_0, pos_1, pos_2, and pos_3. pos_0 is the starting point, and pos_3 is the ending point. Assuming horizontal subpath taken first, pos_0 and pos_1 are the start and end positions, respectively, of h and pos_2 and pos_3 are the start and end positions of v. If $pos_1 \neq pos_2$ then there is a path around a hole that goes from $pos_1.x$ to $pos_2.x$ and $pos_1.y$ to $pos_2.y$. This hole is only traversed along two dimensions, one horizontal and one vertical, with lengths l_1 and l_2, respectively. Therefore, $l_1 + l_2 \leq 2L_{long}$.

From Lemma 3, $H \leq (2a+1)L_{long}$ and $V \leq (2a+1)L_{short}$. Thus, the total length of the path at most $(2a+1)L_{long} + (2a+1)L_{short} + l_1 + l_2 \leq (4a+4)L_{long}$. □

Lemma 9. *In a room with n points the total work by all agents is $O(n)$.*

Proof. Let's take the case that there is a single agent with no specific work threshold to returning to the meeting point in the room. From Algorithm 4, the agent takes the original area, moves to the middle of the long dimension and performs a split traversing unexplored positions along this traversal. After exploration $Area_0$, the total room, generates a stack of connected areas $Area_1, Area_2, \ldots, Area_i$ such that there is a path of previously explored points on the split that connects each pair of $Area_j$ and $Area_{j+1}$. When $Area_j$ is popped off it can either make sub areas $Area_{j_1}, \ldots, Area_{j_i}$ that are connected through the split of $Area_j$ and $Area_{j_i}$ connect to $Area_{j+1}$ through the previous split or the agent moves to $Area_{j+1}$ along the split of $Area_0$. Through this recursive process, the agent traverses split positions at most 4 times (down and up both halves). Every point with $Area_0$ will be

part a split in one of the recursive steps. Therefore, the whole recursive process is generating $4n$ total traversals of the total $Area_0$.

Now let's add multiple agents that balance their work, but without acting concurrently. That is, when the work threshold is reached an agent swaps its stack of work with another agent to balance its work. With a threshold, agents are now able to leave and return to an $Area_j$ if they have completed enough work to return to the leader and another agent returns in their place. From Lemma 8, the threshold $2D$ newly explored points. In this, $2D$ newly explored points pays for $2D$ already explored points when going to the origin and swap the stacks. Since in the worst case an agent may return at most $n/2D$ times to the origin, the total work with the threshold is at most $4n + (n/2D)2D \leq 4n + n \leq 5n$.

Finally, we view the complete algorithm with multiple agents that can also share the work when the threshold is reached, that is, the agents perform parallel exploration. With agents splitting their work with another agent, according to Algorithm 4 one agent had work $Area_1, Area_2, \ldots, Area_i$ and reached the threshold. Upon returning, it shares half the work, some area $Area_j$, such that the first agent contains $Area_1, Area_2, \ldots, Area_j$ and the second agent contains $Area_{j+1}, \ldots, Area_i$. With this addition, agents are able to perform work in parallel, gaining the ability to share work but requiring another D each time sharing happens to account for the new agent to traverse to its given area. Therefore, total work becomes at most $4n + (n/2D)3D = 4n + 3n/2 \leq 6n$. □

Lemma 10. *At the end of the room scan of a room with n points, given the initial work of all agents is* 0*, one of the two holds:*

i. If $n < 2Dk$, the difference in work between any two agents is $4D$, or otherwise,

ii. the ratio of the work between any two agents is bounded by 8.

Proof. For case i, as agents return to meet after $2D$ work, where $n < 2Dk$, then there is not enough traversable points for all agents to get work. This will lead to some out of k agents completing $4D$ work and other agents not completing any work.

For case ii, $n \geq 2Dk$, such that all agents were given a chance to be active. The leader maintains a ratio 4 of work between all agents and a ratio 3 between all other agents and the leader. As two agents may approach the ratio 4 and the lower one could complete work just after the one with greater work has completed, then the agent with greater work will double their current work before returning being at most ratio 8 more work than this agent. After returning this agent will be set to inactive and swapped with the lower total work agent. As the Leader has global knowledge of the work of all agents, after they share information, the leader can maintain the ratio 8 of work between all agents. □

Theorem 2. *At the end of the room scan with n points, each agent perform* $O(n/k)$ *work when* $n \geq 2Dk$.

Proof. From Lemma 9, let the total work be $W = c_1 n$, for some constant $c_1 > 0$. From case (ii) of Lemma 10, the ratio of the work between any two agents is at most 8. Therefore, since there is an agent that has work at most $c_1 n/k$, each agent's work does not exceed $8c_1 n/k = O(n/k)$. □

Room Exploration

Let $R_1, R_2, \ldots, R_r$ be the sequence of rooms that are visited in the algorithm. For each room R_i, let D_i be the maximum shortest path distance between any pair of points in R_i, and let n_i be the total number of traversable points in the room. For each room, the total number of previously explored points in the previously

explored rooms are taken into account when determining the difference in work between agents. As with perimeter scan, we will utilize $D_{\max} = \max_i D_i$. We will show that when the sum of work in all rooms explored exceeds $kD_{\max}$, a ratio 8 between the works of the agents is maintained. We define N_i as the sum of all the traversable points in all the rooms up to R_i, that is, $N_i = \sum_{j=1}^{i} n_j$. After the completion of the room R_i, each agent k has an associated work w_k^i.

We extend Lemma 10, to the more general case where agents have prior work.

Lemma 11. *If before the scan of room R_i, $i > 1$, the work of any two agents g and h differ in work $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$, then at the end of the room scan of R_i then the difference is maintained or a ratio 8 is achieved.*

Proof. Initially, only one agent gets active in R_i, suppose g. It has to be that the work of g before R_i, w_g^{i-1}, is at most the work of every other agent h. In its traversal, g gains at most $4D_i \leq 4D_{\max}$ additional work. Therefore, $|(w_g^{i-1} + 4D_i) - w_h^{i-1}| \leq |(w_g^{i-1} + 4D_{\max}) - w_h^{i-1}| \leq 4D_{\max}$. Thus, since our algorithm always picks an agent with the lowest work, as long as a single agent is scanning the room R_i, the maximum work difference of $4D_{\max}$ is preserved.

Now, that the algorithm sends two or more agents for scanning. The agents that are selected by the leader to scan the room must start with a ratio of work 4 within each other. These selected agents have also the lowest work at the time of their activation, with respect to all agents k, and each agent must have minimum total work of D_i. At any time during their scan, the active agents accumulate new work, but each agent at most doubles its work (according to our algorithm). Therefore, the ratio of work among those active agents does not exceed 8.

Between active and inactive agents, the difference of work is decreasing until the time when inactive agents get less work than the active, and the inactive become

active themselves. Until that happens the difference of work between active and inactive is $4D_{\max}$. After that, the ratio 8 is also maintained between active and inactive agents. Even if all agents get active, then they have a ratio 4 among them when they start scanning and will maintain a ratio 8 upon returning if one becomes inactive due to the completion of their area. Therefore, in all cases, if multiple agents start scanning either the work difference is bounded by $4D_{\max}$ or a ratio 8 between the works of the agents is maintained. □

Lemma 12. *If before the scan of room R_i, $i > 1$, the work of any two agents g and h have a difference in their ratio of work $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$, then at the end of the room scan of R_i then ratio 8 is maintained or regress to difference of $4D_{\max}$.*

Proof. Similar to the proof of Lemma 11, at the beginning only one agent scans R_i, say agent g, which gains at most $4D_i$ work. Any other agent h must have work $w_h^{i-1} \geq w_g^{i-1}$. If $w_h^{i-1} < D_i/4$, then, the ratio 8 between these two agents is not preserved, but there is a maximum difference of $4D_{\max}$ in their works.

If more than one agent becomes active for scanning, then similar to the proof of Lemma 11, the work ratio of 8 is maintained or the difference of works is bounded by $4D_{\max}$. □

Lemma 13. *At the end of scanning room R_i for any two agents it holds that either $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$.*

Proof. We will use induction. For the base case, $i = 1$, the claim holds immediately as initially no agent has any work so all are within $4D_{\max}$ and ratio 8. Suppose the claim holds up to the scanning of room R_{i-1}. By induction hypothesis, for any two agents g and h, $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$ or $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$.

If $|w_g^{i-1} - w_h^{i-1}| \leq 4D_{\max}$, then from Lemma 11, at the end of the scanning of R_i the difference of the works between two agents is either bounded $4D_{\max}$ or the ratio of their works is at most 8. Similarly, if $1/8 \leq w_g^{i-1}/w_h^{i-1} \leq 8$, then from Lemma 12, the work difference is bounded by $4D_{\max}$ or the ratio of their works is at most 8.

Therefore, in all cases it holds $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$, as needed. □

Lemma 14 (Room Work)**.** *Upon the completion of each room N_i the total work done up to that point is $O(N_i)$.*

Proof. From above, $N_i = \sum_{j=1}^{i} n_j$, and from Lemma 9, the work done in a room is $O(n_j)$, more specifically at most $6n$. The sum of work for all rooms becomes at most $6n_1 + 6n_2 + \ldots + 6n_i = 6N_i$ which is $O(N_i)$. □

Lemma 15. *Each agent performs $O(N_i/k)$ work when $N_i \geq 2D_{\max}k$.*

Proof. From Lemma 14 the total work from all the agents combined is $O(N_i)$, say $c_1 N_i$ for some constant c_1. Therefore, some agent has performed work at most $c_1 N_i/k$. From Lemma 13,for any two agents g and h we have, $|w_g^i - w_h^i| \leq 4D_{\max}$ or $1/8 \leq w_g^i/w_h^i \leq 8$. Suppose that g has the maximum total work, and h has the minimum total work, among all k agents.

If $|w_g^i - w_h^i| \leq 4D_{\max}$, then agent has g has no more work than $c_1 N_i/k + 4D_{\max} \leq c_1 N_i/k + 2N_i/k = O(N_i/k)$. If $1/8 \leq w_g^i/w_h^i \leq 8$, then agent g has no more work than $8c_1 N_i/k = O(N_i/k)$. Hence, in any case each agent's work is $O(N_i/k)$. □

In order to perform analysis of the time complexity of room scan, we examine the total time given the orthogonal area contains no holes. (Logarithms are base 2.)

Lemma 16 (Room Time)**.** *Assuming no holes, upon the completion of each room R_i the maximum total time up to that point is* $\sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k) + 4D_{\max}k$.

Proof. At the start of room scan, we have an initial swapping of work until every agent reach $4D_{\max}$ work, creating a total time of at most $4D_{\max}k$ for each agent to swap to be active. After this point, at the beginning of each room, as we perform slow start, the agents share work doubling the number of active agents each $4D_i$ time steps taking a total of $4D_i \log k$ time for all agents to become active. As there are no holes, the algorithm generates rectangular areas initially and all agents get an equal portion of the area. Now that all agents are active, and given that the work performed in a room is $6n_i$ (Lemma 9) we subtract the amount of work that has been performed during the sharing process and divide by the number of active agents, $k-1$ as one agent is leader, achieving the maximum total time for room completion $\sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k) + 4D_{\max}k$. □

Combination of Work

In this section we will combine Lemmas 4 and 14 to get the total work for all movements in the algorithm. We will also combine the total amount of work completed by an agent to get the total work done by each agent utilizing Lemmas 6 and 15.

For the combination of these Lemmas and to utilize Lemma 15 for work of agents in room scan we assume $N_i \geq 2D_{\max}k$. As $C_{\max}$ is along a single dimension and $D_{\max}$ is along two dimensions, $D_{\max} \geq 2C_{\max}$. From Lemma 4, we get that the total work during the perimeter is $O(kP_i + N_i)$, and from Lemma 14 we get that the total work during all room scans combined is $O(N_i)$. Therefore,

Corollary 1. *The total work done by all agents at the end of scanning room R_i is $O(kP_i + N_i)$.*

From Lemma 6 agents perform work $O(P_i + N_i/k)$ and from Lemma 15 each agent performs work $O(N_i/k)$. Therefore,

Corollary 2. *At the end of scanning room R_i, each agent performs work $O(P_i + N_i/k)$.*

Theorem 3 (Work Balancing)**.** *If $P_i = (N_i/k)$, then at the end of scanning room R_i, the total work of all agents is $O(N_i)$ and the work of each agent is $O(N_i/k)$.*

Proof. Since from Corollary 1 the total work is $O(kP_i + N_i)$, if $P_i \leq c_1(N_i/k)$ for some constant c_1, then $O(kP_i + N_i) = O(N_i)$. From Corollary 2, the total work performed by each individual agent is $O(P_i + N_i/k)$, with the above value of P_i this is $O(N_i/k)$. □

Combination of Time

Given the Lemmas 7 and 16 we will show the total time to complete the exploration.

Corollary 3. *The total time to complete exploration at the end of room scan is $P + 6\Gamma C_{\max} + 4D_{\max}k + \sum_{i=1}^{r}((6n_i - 4D_i \log k)/(k-1) + 4D_i \log k)$.*

Given this total time to complete the exploration we have certain areas in which the total time is $\Theta(N/k)$. Any exploration cannot do better than $\Omega(N/k)$ as this is all agent would be actively exploring during the entire exploration.For our exploration, we define a large room i is such that $4D_i \log k$ is of order $(6n_i - 4D_i \log k)/(k-1)$.

Theorem 4 (Parallel Exploration)**.** *When in an orthogonal area is made up of a majority of large rooms, and the terms P, $6\Gamma C_{\max}$, and $4D_{\max}k$ are $O(N/k)$, then time complexity is of $\Theta(N/k)$, which is asymptotically optimal.*

Chapter 2
Online Convex Decomposition

2.1 Introduction

In concave polygon decomposition, the goal is to divide the area into multiple convex polygons. As in Figure 2.1, we have a concave polygon on the left and then on the right, we have produced a convex decomposition on the polygon creating four convex areas (polygons) with the use of the interior lines. Traditional research focuses on reducing the number of convex polygons that the algorithm creates, and also on reducing the time the algorithm takes in order to complete.

Typically the solution to the problem is offline, where some compute node knows the whole polygon ahead of time and then solves the problem. Consider a polygon with perimeter length P that has n vertices out of which r are reflex. There exist decomposition algorithms that create a minimum number of convex areas, which depending on the input polygon it is either r or $r + 1$ convex areas.

However, in real life problems the whole polygon may not be known ahead of time, as in robot exploration problems. In this case, a robot is placed inside the polygon on the perimeter and then moves along the perimeter from vertex to vertex until it comes back to its starting point. Thus, the polygon's perimeter is discovered

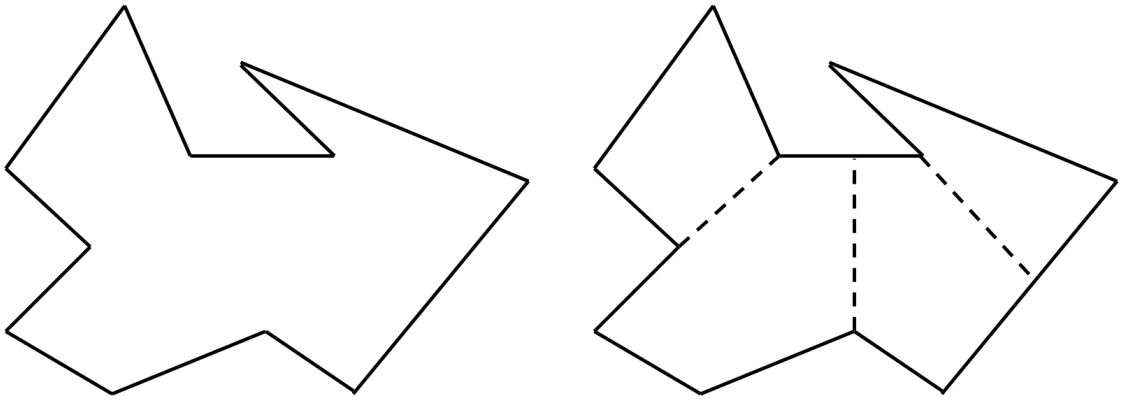

FIGURE 2.1. Applied Convex Decomposition

gradually. The goal of an online solution is to decompose the polygon into convex areas on the fly at the same time that its perimeter is being discovered.

For the online discovery efficiency, we consider the metric of minimizing the total length of the convex areas that are being created. In addition to P the metric includes the interior lines that are being created by the robots, which corresponds to the exploration distance traversed by the robots.

This algorithm gives $r+1$ convex areas. The areas are formed using the existing vertices of the polygon and it also creates new artificial points along the edges of the polygon. This algorithm gives a worst-case optimal number of convex areas. However, the total lengths for the newly created convex areas is $O(Pr)$ which may not be optimal, since P is a lower bound.

2.2 Related Work

In convex decomposition, there are multiple focuses for minimization. Minimum number of convex regions [4, 19, 30, 35], Time bound [17], proximity detection [21], and minimum non-overlapping [8]. Another focus has been triangulation which has a goal "to triangulate a non-degenerate simply polytope of n vertices into $O(n^2)$ tetrahedra" [7]. Convex decomposition has been used in pattern recognition and Star-Shaped has been utilized for guarding problem [16]. Similar to the guarding problem, there is another problem of hidden sets such that no pair of points in the area are visible [33]. Another convex decomposition has been used for Minkowski sums which have been used in "robot motion planning, assembly planning, computer-aided design and manufacturing, and marker making (cutting parts from stock material)" [1].

Minimizing has been performed on simply polygons, to complex polygons with holes in the middle of the obstacle. With the introduction of holes in the polygon, the "exact convex decomposition is NP-hard" [16, 17]. In those without holes, the

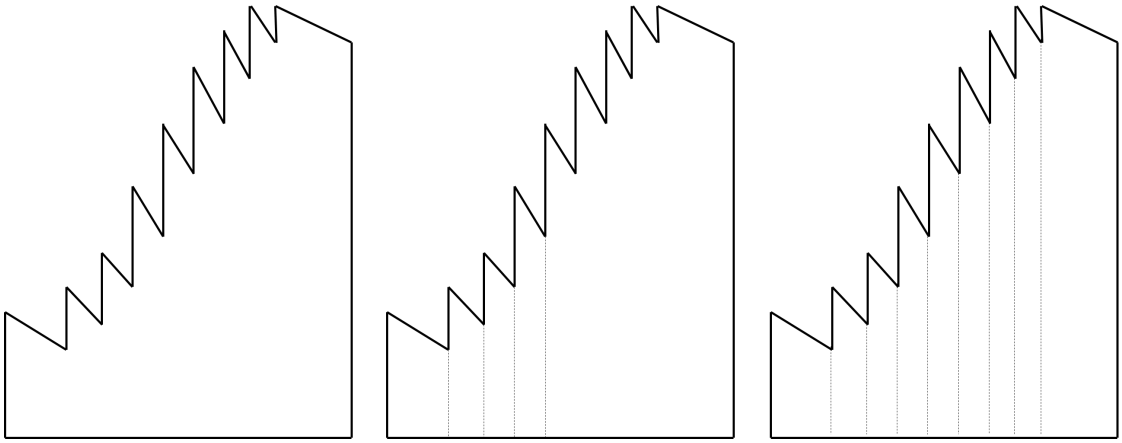

FIGURE 2.2. Algorithm One Convex Decomposition

minimum number of areas is the currently the number of reflex vertices. To achieve this, it takes matching reflex points together. In our paper we achieve the number of reflex vertices plus one because we do not attempt to match up reflex vertices, as in our first algorithm we want to complete online in a single pass.

With time bound research, the goal is to minimize the time necessary to complete the convex decomposition. In this, time bounds have been discussed with Steiner points, as well as without. Steiner points are points that are not the given vertices [17]. This allows convex polygons to close on line segments between points. We utilize the Steiner points for our algorithm but we do not attempt to match reflex points which gives us lower time result but in the average case one additional convex area. Without these points, it requires additional areas to perform the convex decomposition, but requires less time than the referenced Minimum convex decomposition with Steiner points [17].

2.3 Algorithm

Algorithm 5 performs the decomposition of a concave polygon into multiple convex polygons based on the number of reflex vertices that make up the concave polygon. The goal is to minimize the number of new convex areas.

Algorithm 5: Online Concave Polygon Decomposition

```
/* Algorithm for reducing Concave Polygon into multiple Convex
   Polygons                                                    */
```

Given total vertices $P = p_0, \ldots, p_{n-1}$;
Select $B \leftarrow p_0$;
while *some vertex of* P *is not visited* **do**
  $A \leftarrow$ previous vertex;
  $B \leftarrow$ current visited vertex;
  $C \leftarrow$ next vertex;
  while B *is marked* **do**
    Let B' be the closest vertex on last interior line $(\overline{B'B})$ that marked B;
    if $\widehat{ABB'}$ *is reflex* **then**
      Form new interior line $\ell \leftarrow \overline{BV}$ which is a line segment of $\overrightarrow{CB}$ such that V is either an existing visited vertex in P, or V is between two visited vertices of P;
      Remove from P convex area defined by $B, A, \ldots, V$;
      $A \leftarrow V$;
    Remove from P convex area defined by $B, A, \ldots, B'$;
    redefine $A \leftarrow B'$;
    if B *is not marked by another line* ℓ' **then**
      Unmark B;
  if $\widehat{ABC}$ *is reflex* **then**
    Form new interior line $\ell \leftarrow \overline{BV}$ which is a line segment of $\overrightarrow{CB}$ such that: V is either an existing vertex in P, or V is between two vertices of P, or V is the point where ℓ intersects another interior line ℓ' in P;
    if V *is an existing visited vertex in* P*, or* V *is a new vertex placed between two already visited vertices of* P **then**
      Remove from P convex area defined by $B, A, \ldots, V$;
      ℓ becomes part of new perimeter of P;
    else if V *intersects an interior line* $\ell' = \overline{XY}$ *of* P **then**
      Without loss of generality, suppose that X has been visited earlier;
      Remove from P convex area defined by $B, A, \ldots, X, V$ (the convex area includes ℓ and the part of ℓ' from X to V);
      The part of ℓ' from V to Y (endpoint of ℓ') remains as an interior line of P;
      The whole of ℓ and part of ℓ' from X to V become part of the new perimeter of P;
      V is a new vertex of P;
    else
      Mark V;
    If V was not an existing vertex of P then insert it as a new vertex in P;
  $B \leftarrow C$;

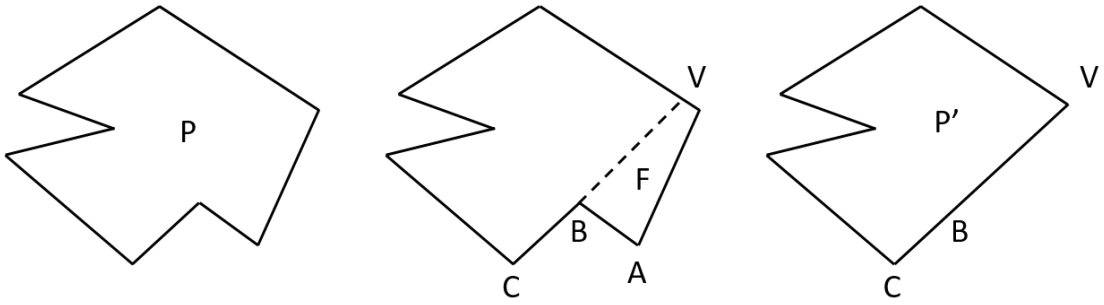

FIGURE 2.3. Closure of Convex Area

Beginning from an arbitrary vertex of the concave polygon P, the algorithm follows along the vertices on the perimeter until returning to the starting vertex. When it encounters a reflex angle, the algorithm removes it by creating an interior line from the angle vertex. The interior lines will define new convex areas. An interior line originates at a vertex and ends either at some point of the perimeter or at another interior line. If an interior line ends at the perimeter then it marks that point of the perimeter. A marked point of the perimeter will be treated as a perimeter vertex, which will be traversed while scanning the perimeter. If the marked vertex is created between two already visited points of the perimeter, then the marked point will not be visited in the future as this marked point will now create a convex polygon. Note that a marked point may coincide with an existing vertex of the perimeter, in which case the vertex is simply marked.

In a concave polygon P, the first reflex angle at vertex B reached will generate the first interior line ℓ. Let C be the next vertex (or marked point) after B in the perimeter. Line ℓ extends from the line segment $\overline{CB}$ internal to the polygon until it reaches the perimeter at some point V, Figure 2.3. Note that V is either another perimeter vertex or a point between two vertices along the perimeter. If V has been previously visited, then a convex area has been created and can be removed. This new convex area F is defined starting from V following the traversal in the perimeter to B and closing it with ℓ. This convex area F is removed from

the original polygon P, resulting to new polygon P'. Then the algorithm continues on P' from B. In this way, B is no longer the center of a reflex angle, and hence, the number of remaining reflex angles has been reduced by at least one. When the new convex polygon F is removed, the whole F is replaced by the line ℓ in P' and we assume that all the points on ℓ have already been visited for the next steps of the algorithm.

In general, the algorithm generates new convex areas (F), along interior lines (like ℓ) formed from reflex angles points (B) to a point across in the perimeter (V). The scenario that we discussed above corresponds to the case when V is a point that has already been visited and this results to removing a convex area (F), since all the reflex angle points in between V and B have been visited before, and hence processed. There are two more possibilities to consider, one is when V has not been visited earlier, and the other is when the line ℓ intersects with an existing interior line. We describe these cases next.

Suppose while visiting B, that ℓ does not intersect with an existing line. If V has not been traversed previously then we simply mark V. In this case, a convex area has not yet been created and the algorithm will continue to traverse to the next vertex C until another reflex angle is reached, or V has been reached. The interior line ℓ (defined from points C, B and V) is still being created anyway but is not used right away. This line ℓ will then be consider as an artificial perimeter boundary when it is intersected by other interior lines, as we describe next.

Suppose that during the traversal of the remaining perimeter P' a marked point B is reached. This marked point was created by one or more B' and the interior area may still be concave, as we can see from Figure 2.5. Through the traversal from Figure 2.5.a-2.5.b we have marked B in 2.5.c twice. Working backwards from the last B' that marked B we attempt to remove at least one convex area. This

process removes the area in Figure 2.5.c and then 2.5.d. If $\widehat{B'BA}$ is reflex, then like with the unmarked reflex angle we create a line ℓ' that extends from the line segment $\overline{CB}$ to V'. As all the vertices inside the area traversed from $B', \ldots, A, B$ have been traversed, $\widehat{ABC} > \widehat{B'BC}$. ℓ' creates two convex areas $V', \ldots, A, B$ and $B', \ldots, V', B$.

Let ℓ be the line that causes V to be marked. During the course of the algorithm, line ℓ might be intersected by other interior lines, due to other reflex angles possibly crossing the interior line ℓ to close a convex area earlier in the traversal. Let B' be the intersection point on ℓ closest to V (if no such intersection points exist then $B' = B$, the original point of the perimeter that defined ℓ). The convex area F that is removed can be traced to the closest interior point B' on the line ℓ. From B', to close the convex area, we continue to follow the closest interior point(s) B' following the other interior line ℓ' that intersected ℓ until the perimeter traversal reaches V.

To explain this better look at Figure 2.4. Beginning from Figure 2.4.a, when we visit B we detect that the angle $\widehat{ABC}$ is reflex. This generates the interior line ℓ intersecting the perimeter at V. Suppose that V has not been visited earlier. Then V is marked. As we continue the traversal, we reach reflex angle $\widehat{CDE}$, Figure 2.4.b. The reflex angle $\widehat{CDE}$ generates ℓ' intersecting ℓ at point B'. As ℓ is an interior line, a convex area that contains C is detected and it is removed; this convex area includes ℓ' and the part of ℓ from B' to B as the new perimeter, Figure 2.4.c.

The other part of ℓ from B' to V remains as an interior line. This process continues to the next reflex point, creating an interior line, Figure 2.4.c. We continue the traversal around the perimeter to V, Figure 2.4.d, detecting convex area F and removing it, Figure 2.4.e. We observe in Figure 2.4.f, the order in which the convex areas are made, 1, 2, and 3, and the respective interior lines generated.

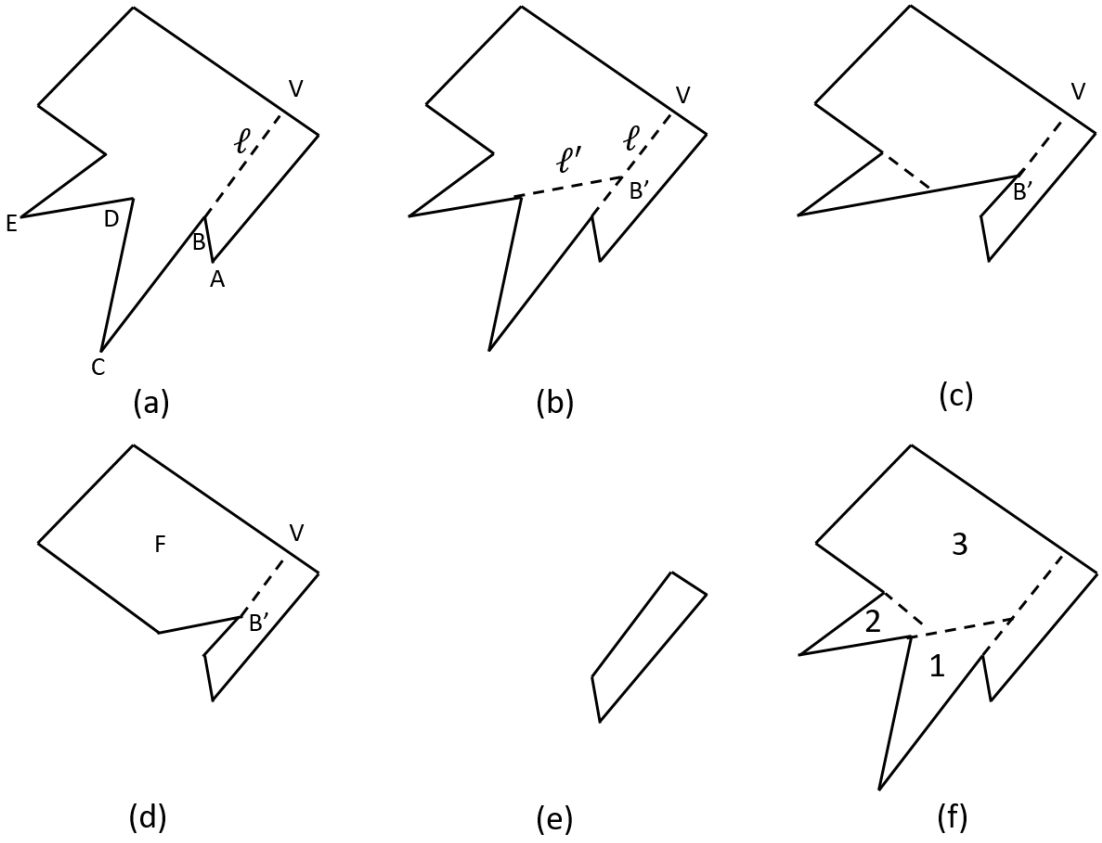

FIGURE 2.4. Closure when V not Visited and ℓ' Intersects ℓ

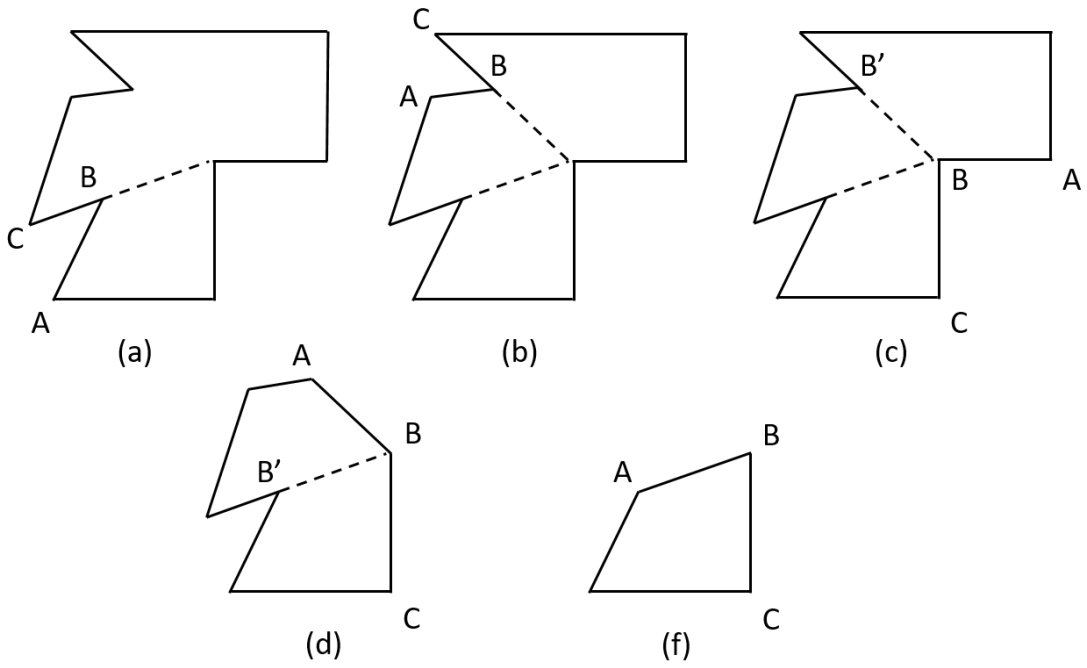

FIGURE 2.5. Closure with Multiple Reflex Marking

This process continues removing the convex areas until all of the points on the the perimeter of polygon P have been traversed, at that time, the total concave polygon has been reduced into multiple convex areas. Now we will walk through explaining Algorithm 5 to describe line by line how this algorithm traverses the polygon and creates the convex areas to be removed.

In Algorithm 5, while scanning the perimeter, we denote the previous vertex as A, current vertex as B, and next vertex as C. We first check if vertex B was marked by a previous reflex vertex being encountered and connected to an interior line. As multiple may have marked this we must work through each line that marked B. For this reason, while it is marked we must continue removing convex areas.

If this position is marked, then we use B' to represent the closest point to B along that interior line (note that B' is either a point of intersection of interior lines or it is a vertex in the original perimeter of P). If $\widehat{B'BA}$ is reflex, then we must create an interior line $\ell \leftarrow \overline{BV}$ which extends from $\overrightarrow{BC}$ and remove the sequence of

points $V, \ldots, A, B$ as they from a convex area.Now $A \leftarrow V$ to continue to remove the next convex area. The sequence of points $B, A, \ldots, B'$ form a convex area. As this is removed, $A \leftarrow B'$. If no more interior lines exist that marked B we remove the marking from B.

If $\widehat{ABC}$ is a reflex vertex, greater than 180 degrees, then this angle must be divided in order to remove the reflex vertex. New interior line ℓ extends from $\overline{CB}$ to a point V (new artificial vertex) such that V is at the intersection of ℓ with previously extended interior line ℓ' or V is a point at the perimeter. If V is at or between two explored vertices, then a convex polygon has been created that includes A and B. We an remove this convex area with ℓ becoming part of perimeter. Else if V is located on an interior line ℓ', we have generated a convex polygon that includes the vertices A, B, and B'. We now include ℓ and the part of ℓ' that runs from B to B'. We still have the interior line of ℓ' that runs from B' to V. Else If V is at a vertex or between two vertices that one or both vertices have not been previously visited then that position is marked. Otherwise, if B is non-reflex and marked, then previous vertex A, the previous reflex position B', and vertex after reflex C' are included in creating a convex polygon. Move to next vertex, $A \leftarrow B$, $B \leftarrow C$, and $C \leftarrow$ next vertex

As we show from implementing Algorithm 5 on the left image in Figure 2.2 we have nine total areas (1 more than the number of reflex vertices). The shortfall in this, is the length of the interior lines utilized to create the convex polygons, even though we only have to use a single line to remove the reflex vertex, the summation of the lengths of interior lines can be greater than that of the perimeter.

2.4 Analysis

We continue now with the analysis of the algorithm.

Lemma 17. *Suppose that upon visiting unmarked vertex B of reflex angle $\widehat{ABC}$ the interior line $\overline{BV}$ is formed. Then, the resulting angles $\widehat{ABV}$ and $\widehat{CBV}$ are non-reflex.*

Proof. As a line extends from $\overline{CB}$ to another vertex V such that $\overline{CB}$ and $\overline{CV}$ have the same slope. A 180 degree, non-reflex, angle $\widehat{CBV}$ has been created. As the maximum vertex is at most 360 degrees, the angle $\widehat{ABV}$ is less than or equal to 180 degrees, as $\widehat{ABV} + \widehat{VBC} = \widehat{ABC}$. Thus, the reflex vertex $\widehat{ABC}$ has reduced to two vertices $\widehat{CBV}$ and $\widehat{ABV}$ that are both non-reflex vertices. □

The new line $\overline{CV}$ that has been created is an interior line, from vertex B to V. Any vertex, where one involved line is an interior line, is called an interior vertex. When a future reflex vertex B' is reached its line $\overline{C'V'}$ is such that V' is on the perimeter of the polygon, or along the interior line $\overline{BV}$. When V' meets another interior line, interior vertices are created.

Lemma 18. *Upon visiting marked vertex B at least one convex area is removed.*

Proof. Let ℓ be the the most recent interior line that defines B as marked. Suppose that the closest point to B on interior line ℓ is B' (note that B' might be a point on the perimeter).

There are two cases:

i. $\widehat{B'BA}$ is a non-reflex angle: Consider the closed area $B', \ldots, A, B$. All reflex angles within this closed area have been removed through either Lemma 17 or if they were marked reflex through case ii below. Hence, this case creates one convex area, $B', \ldots, A, B$.

ii. $\widehat{B'BA}$ is a reflex angle: In this, as $\widehat{B'BA}$ is reflex, then the closed area $B', \ldots, A, B$ is not a convex and we must split it up into convex areas. Since

all the vertices in $B', \ldots, B, A$ have been traversed, C must be located outside of the area which creates angle $\widehat{ABC}$. The angle of $\widehat{ABC} > \widehat{B'BA}$ due to C being outside of the closed area. For this reason, we can utilize line segment $\overline{BV}$ of $\overrightarrow{CB}$ as this point V reduces $\widehat{CBV}$ to 180 degrees. (Note that V is on the perimeter, since according to the algorithm ℓ is the last interior line that marked B.) Thus, since B' must intersect at B, then $\widehat{B'BV} < 180$. From Lemma 17 we have shown that $\widehat{ABV} < 180$. Therefore, the areas $V, \ldots, A, B$ and $B', \ldots, V, B$ are convex areas and are removed.

□

Lemma 19. *After the algorithm visits point B that defines reflex angle $\widehat{ABC}$, then in the remaining perimeter, B defines a non-reflex angle.*

Proof. There are three cases when a reflex angle is encountered. The reflex angle can be either unmarked, or marked. If B is marked due to ℓ (the most recent internal line that marked B) then the angle $\widehat{B'BA}$, using the current B' on ℓ closest to B, is either non-reflex or reflex. Next, we examine the different cases.

i. B is unmarked: In this case, utilizing Lemma 17 creates an interior line ℓ that connects to point V and reduces the two subsequent angles $\widehat{ABV}$ and $\widehat{VBC}$ to non-reflex.

ii. B is marked and $\widehat{B'BA}$ is non-reflex: This case can repeat multiple times as B can be marked by one or more internal lines. As it is non-reflex, but marked, Lemma 18 case i removes the convex area $B', \ldots, A, B$ and sets $A \leftarrow B'$. If there more earlier internal lines that marked B then another B' may exist on such an internal line such that $\widehat{B'BA}$ could either be non-reflex, repeating case ii, or reflex which is handled by case iii below. If there are no

more interior lines, then we unmark B and $\widehat{ABC}$ is either reflex, handled by case i, or is non-reflex.

iii. B is marked and $\widehat{B'BA}$ is reflex: When this case is encountered, Lemma 18 case ii is utilized to reduce $\widehat{B'BA}$. This lemma creates a new interior line splitting the area into two convex areas, $V, \ldots, A, B$ and $B', \ldots, V, B$. After this process, either there is more marked non-reflex, or the remaining $\widehat{ABC}$ is non-reflex. We cannot have case iii repeat on a single marked B as the angle $\widehat{B'BA} > 180$ degrees so no other angle, marked or unmarked can be reflex.

□

Lemma 20. *When an interior line ℓ intersects a visited point then the resulting angles adjacent to ℓ and the other line are non-reflex.*

Proof. The interior line ℓ can land on a visited vertex the perimeter or between two visited vertices either on the perimeter or along another interior line, creating a visted point at the intersection.

i. When a visited vertex is encountered on the perimeter by interior line ℓ, it is known from Lemma 19 that if a reflex angle at that position existed, then it must have been removed through one of the cases in the proof of Lemma 19. Thus any interior line intersecting a visited vertex must produce two new interior angles that are non-reflex.

ii. Given the interior line ℓ intersects between two vertices D, and E of the perimeter or interior line, which generate a straight line $\overline{DE}$. This interior line splits $\overline{DE}$ making two interior vertices such that $\widehat{BVD} + \widehat{BVE} = 180$ degrees.

□

We are now ready to give the main correctness result.

Theorem 5. *The original polygon P is decomposed into at most $r + 1$ convex areas.*

Proof. Given Lemma 17 we have that if a reflex angle is reached an interior line ℓ is created removing the reflex angle. Each of these ℓ associate with the reflex angle that produced them. Each such interior line ℓ is converted to a perimeter line once. Lemma 19 shows the different ways to perform this and remove a convex area. Through this process, each reflex angle r produces a single line producing one convex area which we can associate to the reflex angle.

From here, we have to finalize the last area. Given Lemmas 19 and 20, in the last area all the reflex-vertices forming it have been visited, and hence, all the reflex angles have been removed and none of the interior vertices produce any new reflex angles. Through Algorithm 5 the traversal finishes visiting the remaining non-reflex vertices closing the last area which is convex.

This gives a maximum of $r+1$ convex areas, where r are due to the r reflex-angles and one more due to the last area.

□

We continue with the computational complexity analysis.

Lemma 21. *Given a polygon with n vertices, r of which are reflex, we can solve the convex decomposition problem in $O(nr)$ time.*

Proof. When a reflex vertex is reached, V must be determined. This involves where the ray $\overrightarrow{CB}$ and line segment $\overline{ab}$ of consecutive perimeter points or internal line intersect. The intersection can be determined in constant time. As there are n

vertices and at most r internal lines, a reflex vertex takes $(n + r)$ time to find an intersection V. This means, for a concave polygon with r reflex vertices, it takes $O((n + r)r)$ time. As $r < n$, $O((n + r)r) = O(nr)$. □

Lemma 22. *Total length of interior lines is Pr.*

Proof. Each interior line made from this algorithm, in the worst case, can be the length of the perimeter P. There are r interior lines to generate all the convex polygons, we get Pr total interior line length. □

Chapter 3

Rectilinear Convex Decomposition Minimize Interior Cut-Length

3.1 Introduction

As with Chapter 2, this chapter also is concerned with convex decomposition. However, in this chapter we put emphasis on reducing the interior cut-length given a rectilinear polygon. This is a non-trivial task in an online algorithm, due to not only having to make convex shapes but also having to maintain interior cut-length relative to the perimeter of the polygon. In order to do this with limited knowledge of the area, interior lines are only created for known convex areas. We introduce the idea of levels that begin with level 0 for the perimeter and increase with how the interior lines are generated, connecting a reflex point to the perimeter or through merging two reflex points. With the levels for interior lines we close a known convex area only when the parallel portion is of a lower level that that of the interior line(s) that generate the convex area. Through this, my algorithm produces $3r + \log_3(r) + 2$ total convex areas, given r reflex points. Minimizing cut-length has not been a focus on previous papers though is important to observe with the introduction of mobile agents. With this algorithm, a total of $2\log_3(r)+1$ levels that create a total of $P(2\log_3(r)+1)$ interior line lengths. This compares to $r+1$ total areas and Pr interior lengths from Chapter 2.

3.2 Algorithm

Algorithm 6 performs the decomposition of a rectilinear polygon into multiple convex polygons. The goal is to ensure the length of interior lines to be of the perimeter.

Given the vertices $P = p_0, p_1, \ldots, p_l$ such that we begin at $B = p_0$ and the last vertex p_l is connected to p_0. The traversal follows to the next vertex in P until

Algorithm 6: Rectilinear Minimize Interior Closure

```
/* Algorithm for minimizing interior lines in rectilinear polygon
   */
```
Given total vertices $P = p_0, \ldots, p_{n-1}$;
Select $B \leftarrow p_0$;
Every vertex and perimeter line segment is initialized at level 0;
S is initially an empty stack that will contain visited vertices that define reflex angles;
while *all* P *not visited* **do**
    $A \leftarrow$ previous vertex;
    $B \leftarrow$ current visited vertex;
    $C \leftarrow$ next vertex;
    if $\widehat{ABC}$ *is reflex* **then**
        `// Match with top of stack`
        MatchReflex();
        `// Push on the stack`
        Push B to stack S
    else if *at a point* B *with coordinates* (x, y) *such that* x *or* y *coordinates are the same with that of a reflex vertex* T *in stack* S **then**
        if *sum of the lengths of line levels less than level*(T) *is less than or equal to the length of interior line* **then**
            Form and interior line ℓ that connects (x, y') or (x', y) and T;
            Let U be the top of S;
            while $T \neq U$ **do**
                if *interior line* ℓ' *can be made from* U *to* ℓ **then**
                    create line ℓ' from U to ℓ;
                else
                    create line ℓ' from U perpendicular to ℓ
                $\ell' \leftarrow level(T) + 1$;
            Remove area traversed from (x, y') or (x', y) to B;
            Pop T from S;
    $B \leftarrow C$;
if S *is not empty* **then**
    traverse to point $V \leftarrow (x, y)$ that matches with top stack T;
    treat V as an artificial reflex;
    $\ell \leftarrow$ max level of all lines inside traversed area from T to V;
    pop T from S;

Algorithm 7: MatchReflex

```
   /* subroutine to match reflex points with top of stack          */
1  New ← true;
2  while New do
3  |  New ← false;
4  |  if S is not empty then
5  |  |  if B it at least the level of T at top of S and merge by at most two
   |  |   internal lines then
6  |  |  |  if B and T share the same x or y coordinate then
7  |  |  |  |  Create interior line ℓ that connects B and T;
8  |  |  |  |  ℓ ← max level of parallel line segments inside traversed area
   |  |  |  |   from T to B;
9  |  |  |  |  Remove area traversed from T to B;
10 |  |  |  |  B ← C;
11 |  |  |  else
12 |  |  |  |  create two interior lines that connect at V ∈ (x, y);
13 |  |  |  |  Lines are ℓ and ℓ′ such that B ∈ (x, y′) and T ∈ (x′, y) or vice
   |  |  |  |   versa;
14 |  |  |  |  level(V) ← max level of all parallel line segments of ℓ and ℓ′
   |  |  |  |   inside traversed area from T to B;
15 |  |  |  |  if ∠BVT is non-reflex in non-traversed area then
16 |  |  |  |  |  connect V to V′, such V′ that extends from BV⃗ or TV⃗;
17 |  |  |  |  |  B ← C;
18 |  |  |  |  else
19 |  |  |  |  |  Remove area traversed from T to B;
20 |  |  |  |  |  B ← V;
21 |  |  |  |  |  New ← true;
22 |  |  |  Pop T from S;
23 |  |  |  continue;
24 if B == V then
25 |  push V on S;
```

reaching a reflex point. When this is reached we utilize previously visited reflex point that was unable to close a convex area. If this is unable to happen, it will be matched with a later reflex point or to the perimeter. The internal line that are generated have less length that that of the parallel segment of the convex area's perimeter that was closed. Upon returning to p_0 there may be one final reflex point that needs to be connected. Once this has been performed, the concave polygon has been reduced into multiple convex areas.

Beginning from an arbitrary vertex of the concave polygon, the algorithm follows along the vertices on the perimeter until returning to the starting vertex. Levels are introduced such that all vertices on the perimeter are initially at level 0 and each time two vertices are matched up (with new internal lines) and create a new interior point (where the internal lines cross), the level of this point is one more than the current level. Two reflex angles can only be matched if the top of the stack of unmatched reflex points is a member of a lower level. A line connecting two points gets the maximum level of the points that define it or the parallel line in the closed area.

When the first reflex vertex B is encountered we check the stack, which is empty, and push B on the stack.

Upon reaching the next reflex vertex, B', we attempt to connect B and B' to generate at most one vertex V of an increased level if B and B' meet at a 90 degree angle, Figure 3.2.b, or otherwise, no reflex angle is produced, Figure 3.2.c. We remove any points that are between the traversal from the B' to B, as this is a new convex area F. The two lines used to generate V, ℓ and ℓ' are now added to the perimeter.

If the new point V is created it will be the first vertex of an increased level (level 1, if B and B' are at level 0). Now the perimeter traversal will continue from V,

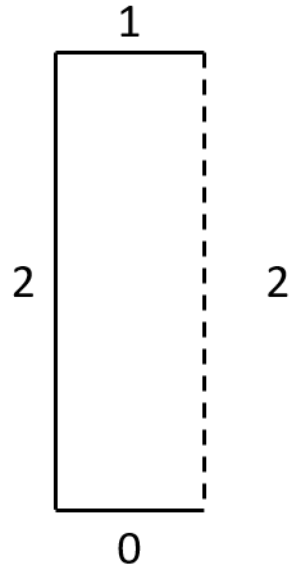

FIGURE 3.1. Levels of Lines with Attempted Closure

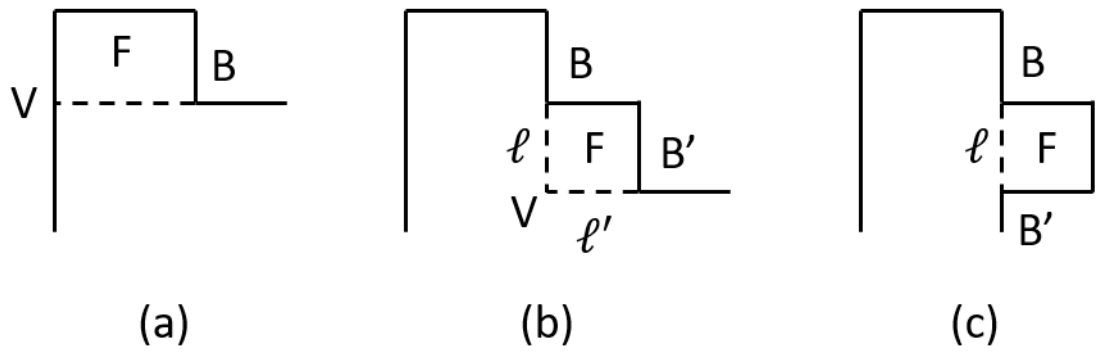

FIGURE 3.2. Possible Closures of Convex Areas

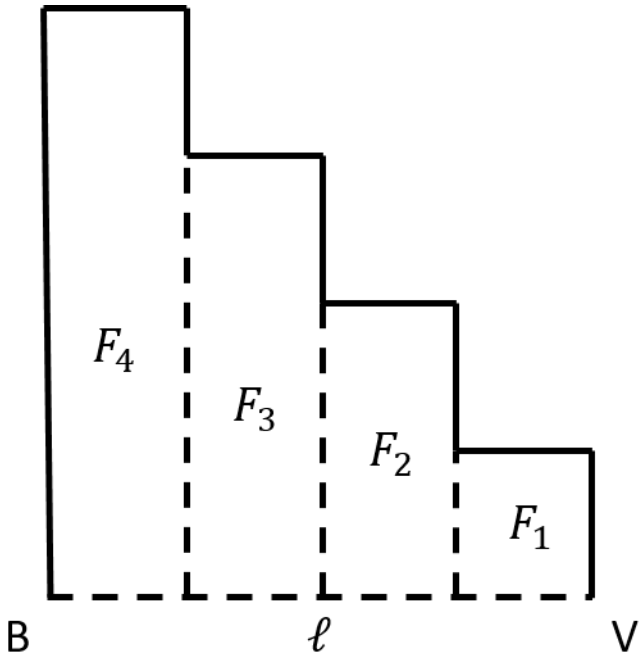

FIGURE 3.3. Connecting to Perimeter Generating Multiple Convex Areas

since it defines a new reflex angle. We only check if we can connect it to a lower level point on top of the stack, similarly to the way we did with the first vertex, or otherwise V is being pushed to the stack. We can only match reflex points of greater level with the top of the stack. If we cannot connect it to a perimeter point now, and it does not merge with another reflex point, then it will be connected to a perimeter point when V is reached, Figure 3.2.a.

When point V is reached, it may be that B that can connect to this point is not on top of the stack. We scan the stack for B and generate interior line ℓ to create line $\overline{BV}$. Since this is created, we must remove all the reflex points that exist in this closed area (those on top of the stack to B). To remove these, we pop them off one at a time and either connect them to ℓ, if the interior line can be made, or we connect it to the other end perpendicular to ℓ. Through this process, we generate multiple convex area, Figure 3.3. To be on the stack, we must have a lower level vertex than the previous top of stack. For this reason, generating areas

F_1, F_2, F_3, F_4, in Figure 3.3, we increase the level each time so the parallel portions of closed areas are always less than that of the newly created interior line.

We continue this process by checking if we can match reflex points. If this is not possible then we will push it on the stack to match with a future reflex, or perimeter point. When the original point has been reached it may be the the last reached reflex point remains on the stack. To remove it, we must traverse to the perimeter point that can generate line ℓ and merge them. At that point, the area no longer contains reflex angles and is entirely covered by convex areas.

We implement Algorithm 6 on a polygon, such as the one in Figure 3.4.a. Seen in Figure 3.4.b, we have matched some of the reflex angles with one another and one has been connected to a perimeter line. After creating two reflex vertices of level 1, we match up these vertices in order to make a level 2 vertex, Figure 3.4.c. The traversal continues around connecting a reflex angle to a perimeter point, after that perimeter point is reached and then performing the same on the level 2 interior vertex, Figure 3.4.c. When we return to the starting vertex, we have closed the last convex area and all reflex vertices are either matched with another reflex producing one new reflex with level one greater, eventually with a lower level line removing the reflex vertex.

Now we will walk through explaining Algorithm 6 to describe line by line how this algorithm traverses the polygon and creates the convex areas to be removed.

Beginning from an arbitrary vertex, continue to visit vertices until returning to beginning vertex. We generate empty stack S to contain any reflex point that has not been removed as well as its associated level. Every vertex is initialized into 0. Previous vertex is A, current vertex is B, and next vertex is C.

If $\widehat{ABC}$ is a reflex vertex, greater than 180 degrees, then this vertex must be divided in order to remove the reflex vertex. We then call MatchReflex to attempt

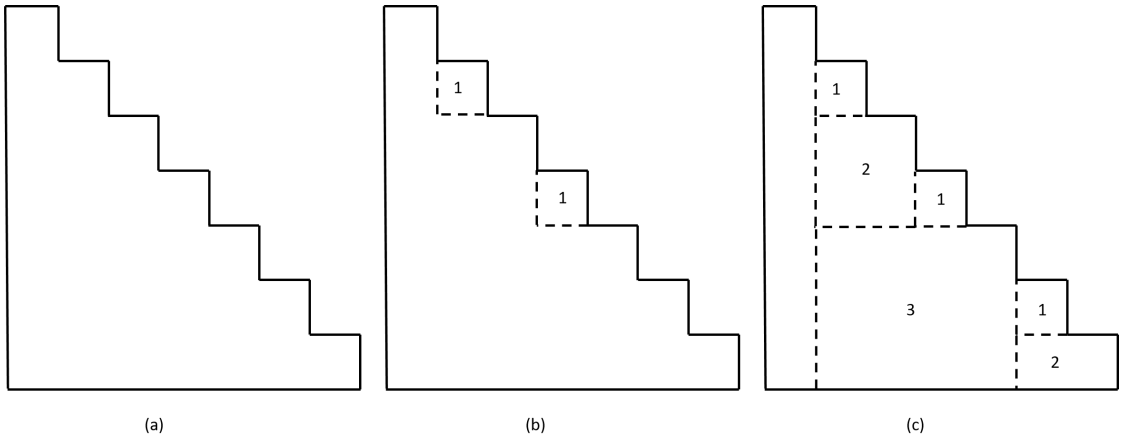

FIGURE 3.4. Algorithm Two Rectilinear Decomposition

to match with top of the stack for each new reflex generated. If we cannot match B to the top of stack, then we push B on top of the stack S.

Otherwise, if not at a reflex vertex, but at the x or y of a reflex vertex T in S then check if the sum of lengths of the parallel line segments with level less than that of T is greater than or equal to the length of the interior line ℓ. If this is true, then create the ℓ connecting (x, y) and T. As T may not be top of stack, we must pop top points off the stack until T is on top. To do this, we examine top of stack element U creating ℓ' from U to ℓ, if an interior line between U and ℓ can be created, else ℓ' is created in the other direction perpendicular to ℓ and popping U off the stack. Once T is on top of the stack we remove the area traversed from T to B and pop T from S. Finally, if neither at a reflex or a position on a line that matches the x or y of a reflex vertex, then move to next vertex C. After all the points have been traversed, it may be such that the last reflex still needs to be removed. To remove this we must traverse around to the point V on the perimeter. Treating V as an artificial reflex, we make the connecting line ℓ the max of all lines in the closed area from T to V. Then pop T off the stack and the concave decomposition is completed.

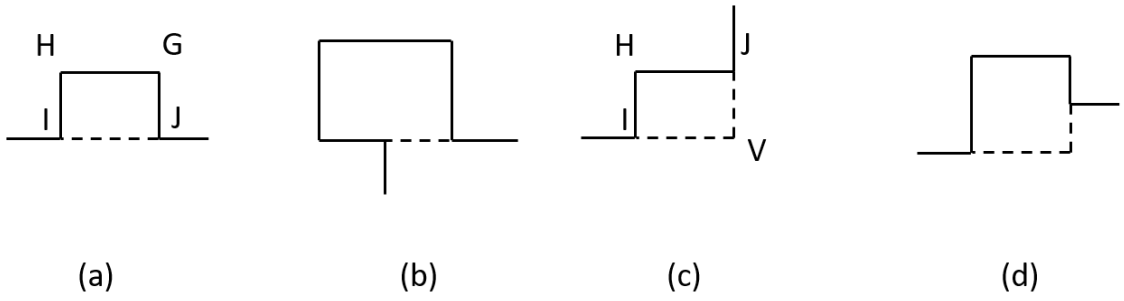

FIGURE 3.5. Ways to Close Convex Areas

For the subroutine, MatchReflex, we loop while we create a new reflex vertex to attempt to match again. if there is a reflex point in S and if so, then if B is at least level of top of stack T and separated by a non-reflex point. B and T can now be matched either with one line ℓ if the share x or y coordinate, or with two interior lines meeting at point V. This point V is one level greater than B or V. In both cases we remove the area traversed from B to T and pop T off the stack. When V is created we move to V as this is reflex to match with top of S or push it on the stack. The line(s) made , ℓ to close area are maximum level of all closed perimeter lines plus one. If a vertex V is made and is not matched with the top of the stack, then push V on S.

In the next Chapter, we will discuss the changes that had to be made to this algorithm in order to generalize to any concave polygon.

3.3 Analysis

Lemma 23. *When the top of stack point I is at most the level of the current reflex point J, and internal lines (one or two) connecting them can be formed, then a new closed area is formed and the top of stack I is popped. Moreover, the levels of the newly created lines are higher than the levels of the parallel edges of the closed area, and the total length of the newly created lines is matched with the corresponding length of lines at lower level.*

Proof. There are two cases for this, the first, is that the reflex angles match with a single straight interior line ℓ generating no new reflex angles. Second case, the reflex angles match with two interior lines ℓ and ℓ' generating a new reflex angle.

i. *Reflex angles I and J match with a single interior line ℓ:*

In this case, I was at the top of the stack and at least two non-reflex points H and G between, as shown in Figure 3.5.a and 3.5.b. In the case of Figure 3.5.a, the newly visited J is connected to I through lines $\overline{IH}$, $\overline{HG}$ and $\overline{GJ}$. As I was on the stack, it cannot be that another reflex remains between these two reflex or it would have been removed from the stack already. This means that the line $\ell = \overline{IJ}$ is the same length as the perimeter line $\overline{HG}$ of the closed area that is parallel to ℓ, and moreover the length of the line ℓ is exactly the length of $\overline{HG}$. In the other possible case, similar to Figure 3.5.b, the line length of ℓ is less than the parallel line on the closed rectangle which is at a lower level.

ii. *Reflex angles I and J match with two interior lines ℓ and ℓ':*

Here, I was at top of the stack, and J is connected along the perimeter by multiple non-reflex points connecting to I. As I and J will merge at a 90 degree angle internally at point V we create two internal lines that meet at V such that $\ell = \overline{IV}$ and $\ell' = \overline{VJ}$. Given the case similar to Figure 3.5.c, I and J are separated by a single non-reflex point H. In this case the length of $\overline{IH}$ is equal to $\overline{JV}$ and the length of $\overline{HJ}$ is equal to $\overline{IV}$. As V is one level greater than I and J the lines $\overline{IH}$ and $\overline{HJ}$ are lower level than $\overline{JV}$ and $\overline{IV}$. In the other possibility, shown in Figure 3.5.d, one more non-reflex points can exist on the perimeter between I and J, however, the lengths of the along the perimeter will have greater length than the length of ℓ and ℓ' combined.

□

Lemma 24. *Given a level 0 reflex point B on the stack, that is not matched with another reflex point, it will be able to close a rectangular area F with a level 1 line ℓ from B to a level 0 reflex point V.*

Proof. Without loss of generality, let Z be the top left corner of F created by ℓ. Suppose for the sake of contradiction that ℓ has level at least 2.

Then it must be such that the ℓ' parallel to ℓ in F is at least level 1 (according to the algorithm), hence was not part of the original perimeter. We will show below that such an ℓ' cannot be created.

There must exist B' that connect to another point, V', either perimeter or reflex, generating ℓ'. For B' to be pushed on the stack, as it is the at least the level of B it must be such that they cannot connect through at most two interior lines. This means that B' cannot exist on the parallel line of ℓ to be pushed on the stack. We examine the following cases.

- Reflex point B' exists on left perpendicular to ℓ of F:

  Through Algorithm 6, B' is outside or along a perpendicular line to ℓ of F in order to produce ℓ' to traverse the entire parallel line to ℓ. Upon reaching Z, following the algorithm, it must be that all reflex points have either been matched, through Lemma 23 or, they have been connected to the perimeter removing all from the stack. Hence, B' cannot exist outside F on the left perpendicular of ℓ.

- Reflex point B' exists on right perpendicular to ℓ of F:

  In this case, interior lines can be made, but it would take at least three to close an area, which is unable to close. In this scenario, it is impossible to make a line across, as through Algorithm 6, lines can only be made when a V

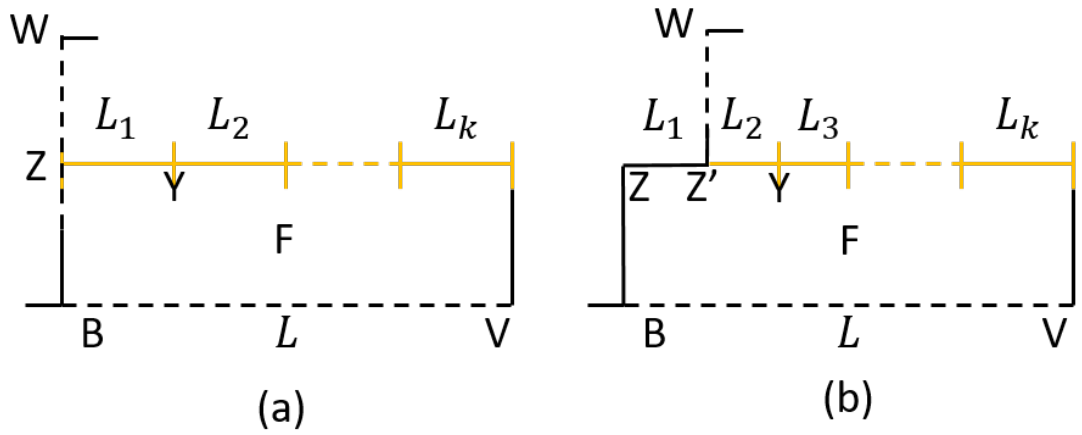

FIGURE 3.6. Reflex Removal Through Perimeter Connection

is reached after the reflex point is added to the stack. B' has been visited after all the points between B and Z have been visited so a line ℓ' that parallel to ℓ cannot be created. In this case, upon reaching V, following the algorithm, we generate perpendicular ℓ' that is a segment of the right perpendicular of ℓ in F

Through both of these cases, it is such that the parallel to ℓ' is level 0 and ℓ is level 1. □

Lemma 25. *Given a reflex point B on the stack, which is not matched with another reflex point, it will be able to close with line ℓ from B to a perimeter point V, creating rectangular areas (at least one) such that the level of ℓ is greater than the maximum level of any segment in the parallel line of ℓ in those rectangular areas.*

Proof. When reflex point B was initially encountered, we could not match it with the reflex point on top of the stack, because the stack was empty or top of stack was at a greater level so it had to be placed on the stack. Continuing the traversal, we eventually reach a perimeter position V that shares the same x or y coordinate with B.

B and V can be connected with a single straight line ℓ. Suppose that line ℓ is of level L, which implies that the level of B is $L-1$. In case B was not at the top of the stack, we must now pop off elements from the stack up to reflex point B which creates multiple rectangular areas as shown in Figure 3.3. Without loss of generality, we will focus one of those rectangular areas (i.e. $F = F_L$). For simplicity assume that ℓ is the lower edge of F and V is its right end, as shown in Figure 3.6.

Let Z be the point at the top left most corner of F (3.6). Parallel to ℓ on F we have line ℓ', with one endpoint being Z. The line ℓ' consists of segments $\ell_1, \ell_2, \ldots, \ell_k$, with respective levels $L_1, L_2, \ldots, L_k$ (Figure 3.6), such that the end points of each segment are perimeter points (either original perimeter points or artificially created at level 1 and higher). To have these line segments, there have to be visited reflex angles defining them. We need to prove that $L_j < L$, for all $1 \leq j \leq k$.

Suppose that a line l_i was created such that both of its end points were original reflex points of the initial perimeter of level 0. Then according to Lemma 24, the level of ℓ_i is 1. If such ℓ_i exists, then it must be that $L > 1$, since if $L = 1$, then the level of B is 0, and one of the endpoints of ℓ_i would have been matched with B to create a different closed area than F, giving a contradiction.

To continue, suppose that at least one of the ℓ_i was created due to some artificially created reflex point during the course of the algorithm which was not part of the original perimeter (i.e. the level of Y is at least 1). Let Y be the leftmost such artificial reflex point on ℓ which is on some ℓ_{i_Y}. For the sake of contradiction, suppose that $L_i \geq L$.

Without loss of generality, we focus on two cases, $\ell_{i_Y} = \ell_1$ and $\ell_{i_Y} = \ell_2$ as shown in Figures 3.6.a and 3.6.b, respectively. The analysis of these two cases is general enough to cover all the other possible positions of ℓ_i on ℓ'.

i. $\ell_{i_Y} = \ell_1$ *(Figure 3.6.a):*

Let W be the first point along $\overrightarrow{BZ}$ where the current perimeter turns for the first time to the right of Z.

In this case, ℓ_1 connects to Z and Y. There are three possibilities for this case; Y is created from reflex points above and to the right, Y is created from reflex points above and to the left (namely Z is to the left), and $W = Z$.

a. Y is created from reflex points above and to the right:

   In this case, after Y is created, gaining level at least $L - 1$ it connects to a point Z. Through Algorithm 6, if Z is a reflex point connected from below and to the left, then we know that by W we reach a point that matches its x coordinate, and through the algorithm we would pop off all reflex points to Z generating that many convex areas. Thus, Z is not a reflex point when W is reached, which comes before Y in the traversal, so ℓ_1 is not created at all.

b. Y is created from reflex points above and to the left:

   In this case, Y connects directly to Z and another reflex point above Y. For this to be possible, Z must be a reflex point on top of the stack when that other point is reached. As shown in case [a.], when W is reached Z no longer can be reflex and as such, could not merge with another reflex point. Y cannot be generated in this way.

c. $W = Z$:

   In this case, as since W is the point that turns to the right of Z and Y connects to Z, then it must be such that Y was not artificial.

Through this, Y cannot connect to Z in any of these cases thus ℓ_1 is not generated. Also, if Y was greater level than B it must be that Y would have merged with B as it was assumed to be the first artificial vertex and if any other reflex existed between W and Y then they would have merged creating a closer vertex along ℓ' at a higher level, causing a contradiction for Y being the first artificial reflex (this process continues until all other reflex points are popped finally merging with B).

ii. $\ell_{i_Y} = \ell_2$ *(Figure 3.6.b):*

Let $\ell_1 = \overline{ZZ'}$. Let also W be the first point along the parallel of $\overrightarrow{BZ}$ that goes through Z' where the current perimeter turns for the first time to the right of Z'.

Then, ℓ_2 connects to Z'. Moreover, Z and Z' must exist along the original perimeter as Y is the first artificial reflex point created along ℓ'. To begin, from the traversal from B to Z it could be similar to the first case from B to W such that all the reflex points between them have been removed. There are two cases for Z', where its level is equal to B or the level of Z' is less than B.

a. Level Z' = Level B:

Through this case, as Z' is at a level equal to B, it must be that Level $B = 0$. For this, it must be that B is on top of the stack, due to all other reflex being removed upon reaching Z, and Z' would have had to connect to B thus ℓ would not be created.

b. Level $Z' \neq$ Level B:

In this case, Z' is level less than B, it is then that Z' is pushed on top of the stack with B as the next element. As Z' is a lower level than B then

there are different cases similar to case [i.] are possible; Y is created from above and to the right, Y is created from above and to the left, and $W = Z'$.

1. Y is created from reflex points above and to the right:

   Through this case there exists a reflex point above Y that is on top of the stack when the reflex point to the right of Y is on top of the stack. Similar to case [i.][a.] upon reaching W all reflex points have been removed between Z' and W putting Z' on top of the stack. As Z' is on top of the stack, the next reflex either one that was earlier than the one above Y or the one that is above Y.

   A. next reflex point after W appears before the one above Y:

      Given that Z' is level 0 and on top of the stack, the next reflex points that is encountered, after visiting W will be able to merge with Z'. Through this, an artificial reflex point is created along ℓ_2 that would connect to Z'. This is a contradiction as Y is the next reflex point to connect to Z'.

   B. next reflex point after W appears above Y:

      In this case, given that Z' is level 0 and top of stack, as with [A.] it has to be that the next encountered reflex point will merge with Z'. In this, Z' connects with the reflex point above Y. Through merging these points, the artificial reflex point crated along ℓ_2 would be Y. This is a contradiction as it was assumed that Y was created from above and to the right, but due to this, Y is created from above and to the left.

2. Y is created from reflex points above and to the left:

   In this case, Z' is one of the reflex points merged to create Y. As the point above Y was the current point in the traversal, to merge with Z', it must be such that Z' is on top of the stack. When merged, Z' is popped from the stack, making B top of stack and creating Y. In this case, Y is the level of ℓ_2, namely,at least L. As B is at most $L-1$ and B is on top of the stack, Y would have merged with B, which is impossible.

3. $W = Z'$:

   As in case [i.][c.] if $W = Z'$ then it must be that Y is not artificial, which contradicts the assumption that Y is artificial.

Given the above cases, it cannot be such that Y is a higher level than B and ℓ_{i_Y} to be created. These cases are repeated and for all ℓ_i up to ℓ_k.

□

Theorem 6. *At the end of the traversal there are no remaining reflex points on the perimeter.*

Proof. We only need to show that the stack of points that define reflex angles at the end of traversal will be empty. There are three possible cases for the stack being empty at the end of traversal; no reflex point added to stack, reflex angles are matched, we match top of stack with later explored perimeter point, or the last reflex is handled by Algorithm 6.

i. No reflex point added to stack: From Lemma 25 case i, if all reflex points are able to connect with the perimeter upon reaching the reflex point, then the stack is always empty.

ii. Reflex angles are matched: Given Lemma 23 we have two possible cases. For Lemma 23 case i, we pop to top of the stack off and no new points are pushed on the stack. In Lemma 23 case ii, we pop the top of the stack off and generate an increased level reflex point V. With this, we then attempt to match V with the next element utilizing case ii, connect it to the perimeter with case i. If these are not possible, then push V on the stack and later match (case ii) or connect to the perimeter (case iii).

iii. Matching top of stack with later explored perimeter point: Through Lemma 25 case ii, we have pushed a reflex point on the stack and it has yet to be matched with another reflex. In this case, we connect the point with the perimeter and pop the point from the stack.

iv. The last reflex could not be matched and we could have previously reached the perimeter which we cannot close. Due to this, we have to handle this after finish the traversal by Algorithm 6. This is because we must finish traversing all the points and re-reach the perimeter point that matches the last reflex in order to close the convex area.

□

Lemma 26. *A reflex point at level $L \geq 2$ for its creation it requires at least three reflex points at level $L-2$ or lower.*

Proof. Consider a reflex point ϕ at level $L \geq 2$. Then, from Algorithm 6, ϕ must have been created was created by matching two reflex points θ_1 and θ_2, where θ_2 was visited or created more recently than θ_1 such that:

- θ_2 was at level $L-1$. Then the creation θ_2 had to use two reflex points at level $L-2$ or lower, since the creation of a reflex always increases the level,

and those were different than θ_1. Thus, three points at level $L-2$ or lower were needed.

- θ_2 was at level $L-2$. Consider the closed area F which was defined when ϕ was created. On the perimeter of F there must be a line segment ℓ with level $L-1$. The line segment ℓ must have been created when we closed some reflex point ζ at level $L-2$ with a straight line to the perimeter across. Thus, three points at level $L-2$ or lower were used.

□

Theorem 7. *There are at most* $2\log_3(r)+1$ *levels and in total* $3r+2\log_3(r)+2$ *convex areas created.*

Proof. From Lemma 26, every two levels reduce the total number of reflex points by a third. Through this process we have at most $2\log_3(r+1)$ levels until there are no more reflex points.

From level 0, there are r reflex points and connecting them to the perimeter produces $r+1$ convex area. For arbitrary even level $i \geq 2$ there are at most $r/3^i$ reflex points that produces $r/3^i+1$ convex areas. Given there are at most $2\log_3(r+1)+1$ levels, the total convex areas from even levels is $\sum_{i=0}^{\log_3(r+1)}(r/3^i+1)$ which is at most $2r+\log_3(r+1)+1$.

In level 1, there are at most $r/2$ reflex points which produces $r/2+1$ convex areas. An arbitrary odd level $j \geq 3$ there are at most $r/(2\cdot 3^{j-1})$ reflex points that produces $r/(2\cdot 3^j+1$ convex areas. The total convex areas from odd levels is $\sum_{j=1}^{\log_3(r+1)+1}(r/(2\cdot 3^{j-1})+1)$ which is at most $r+\log_3(r+1)+1$.

Summing the even and odd levels we produce a at most $3r+2\log_3(r)+2$ convex areas.

The final reflex point is a special case as this one may not be matched during the first traversal to each point. In this case, we produce one additional level giving a total of $2\log_3(r+1)+1$ levels. This does not change the total number of convex areas created as this reflex will only be used once and fits in one of the two summations above.

□

Given that P is the length of all the perimeter lines.

Lemma 27. *Any level has at most P total interior line length.*

Proof. From Algorithm 6 levels reached through merging reflex points produces lines ℓ of the greatest level of all perimeter lines in the closed area, thus it is at most the length of the perimeter contained in the closed area. Through Lemma25, levels reached through connecting reflex point to perimeter with interior line ℓ is at most the length of the perimeter line parallel to ℓ inside the closed area which is at most the perimeter of the closed area. Since P is the length of the perimeter, the summation of interior lines utilized to generate convex each level is at most P. □

Lemma 28. *The total length of all interior lines is $P(2\log_3(r)+1)$.*

Proof. From Theorem 7 there are at most $2\log_3(r)+1$ levels and since each level has at most length P, Lemma 27, the total length of all interior lines is $P(2\log_3(r)+1)$.

□

Chapter 4

Arbitrary Convex Decomposition Minimize Interior Cut-Length

4.1 Introduction

In order to solve the problem utilizing the Algorithm 6, from Chapter 3, we view the arbitrary reflex angle as three rectilinear reflex angles. To do this, given an arbitrary reflex angle $\widehat{ABC}$, we draw a square around the reflex point B with reflex points A', B', and C', Figure 4.2. We then minimize the size of the square infinitesimally until A', B', and C' share the same space as reflex point B.

Through this, it is such that each reflex point now takes at most three interior lines. It is also such that, without loss of generality, any line that runs vertical or horizontal can be either original perimeter or an interior line while any other line must be such that it was original perimeter as interior lines are only made horizontally or vertically.

Given that three reflex points are considered to be at one original reflex point like B each must push three reflex points on the stack, A', B', and C'. In the process of traversing the perimeter, it can be that one or two of the reflex points that were pushed on the perimeter at B were able to reduce the angle to less than reflex. In the worst case, three reflex points will generate a separate straight line connecting to points on opposite ends of B so the angles on the other end of this line must sum to less than 180 degrees.

When a reflex point B connects to the perimeter, assuming that B is not at top of the stack, multiple convex areas are created, namely number of items on the stack up to and including B plus one (Figure 4.3). As with the rectilinear solution, it must be such that the reflex angles are increasing from the top of the stack, thus, if their level is greater than 0, the reflex point is created by a horizontal and

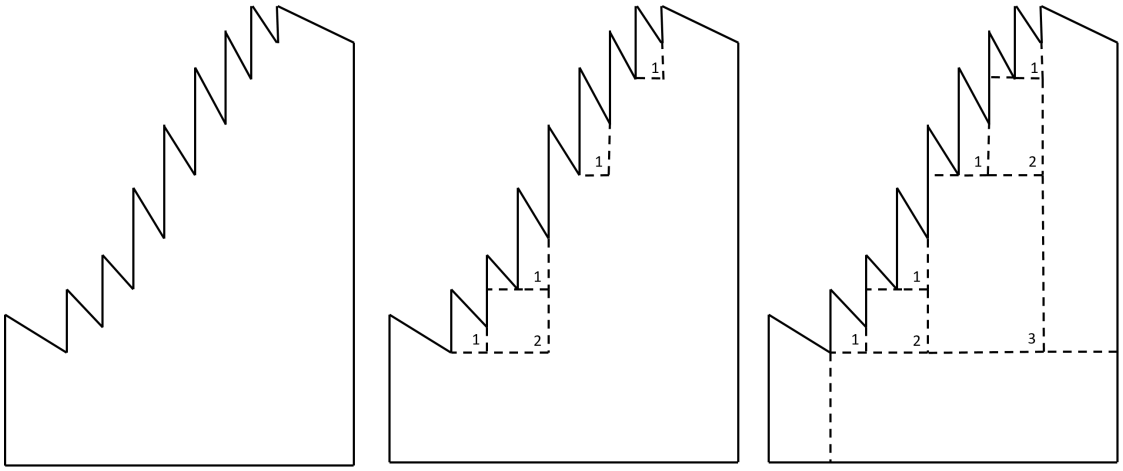

FIGURE 4.1. Convex Decomposition to Minimize Cut-Length

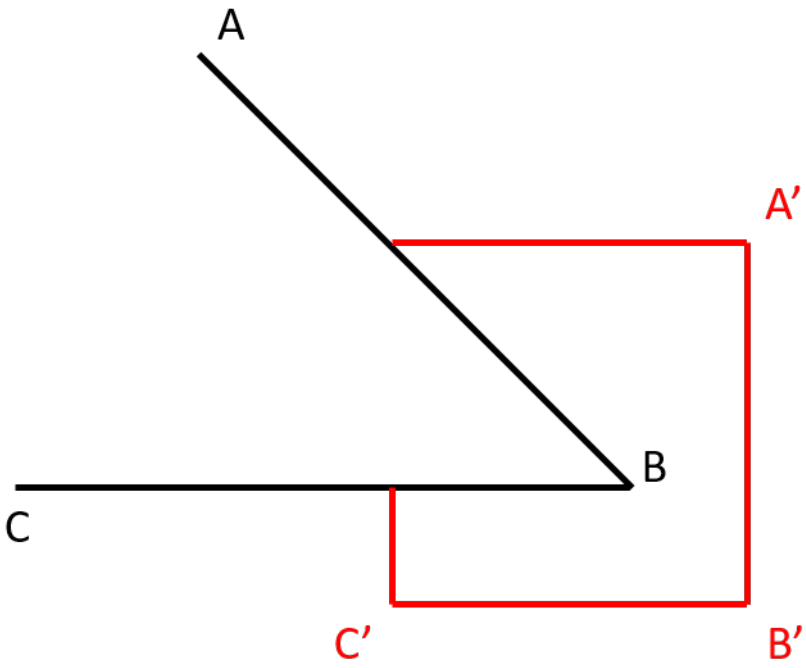

FIGURE 4.2. Reduction from Arbitrary Concave to Semi-Rectilinear

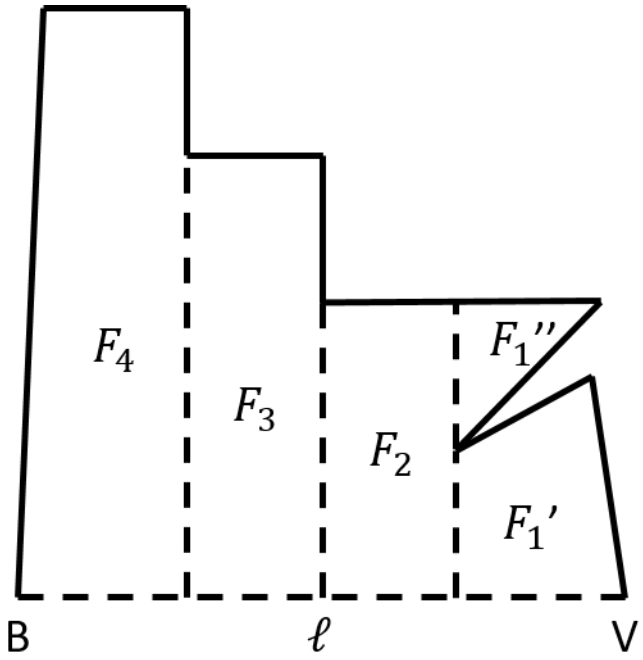

FIGURE 4.3. Generating Multiple Convex Areas with Arbitrary Angles

vertical line, F_4, F_3, and F_2. Thus, only the top of stack element can be level 0 creating at most 2 convex areas, F_1' and F_1''.

Through this reduction, we will prove that, as each reflex counts as 3 reflex, we now have $3r$ reflex giving $9r + \log_3(3r) + 2$ convex areas and interior line length of $P(2\log_3(3r) + 1)$.

4.2 Analysis

Through the performed reduction to the Algorithm 6, we must revisit the elements of the analysis in order to show they still hold and the amount of convex areas and interior line lengths. Unlike with 3.5 there are more than 4 possible ways to close a convex area, but we still are limited to one generating one or two interior line segments to close the area.

With this reduction, we no longer have the case that at least one non-reflex is between, but now it is such that there do not need to be any reflex between, Figure 4.4.

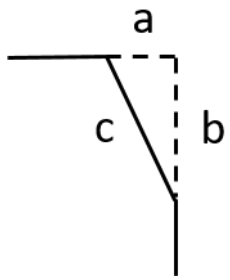

FIGURE 4.4. Convex Area Closed without Non-Reflex Contained

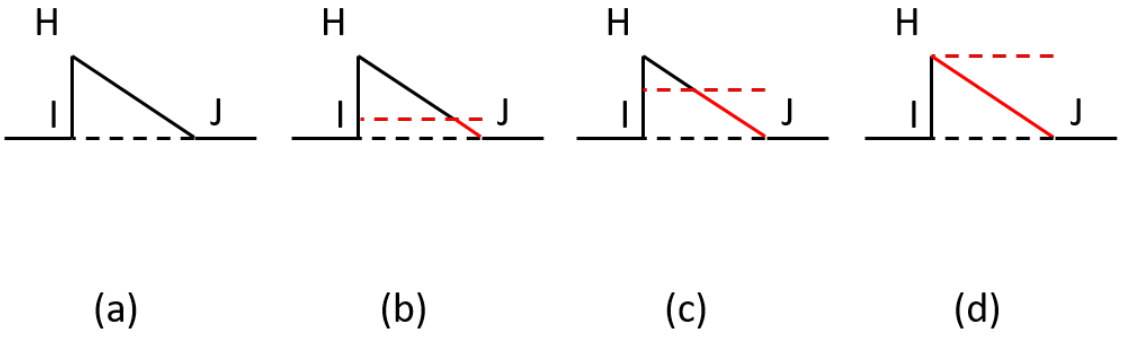

FIGURE 4.5. Determining "Parallel" Line Segment(s) of Convex Area

Lemma 29. *When matching two reflex angles the interior lines are less than* $\sqrt{2}c$ *such that* c *is the length of the perimeter of closed area*

Proof. Through this, we have such that the two interior lines a and b with perimeter line c in the worst case follows the equation $a^2 + b^2 = c^2$ which gives that $c = \sqrt{a^2 + b^2}$. As we know that a and b are less than c, $\sqrt{a^2 + b^2} < \sqrt{c^2 + c^2} = \sqrt{2}c$ □

When determining what is "parallel", as the area is no longer rectilinear, we must redefine what is parallel. In order to do this, as we can have multiple "parallel" line segments with different angles, we extend the interior line across the closed area, Figure 4.5. Through performing this, any perimeter line encountered is "parallel".

Lemma 30. *When the top of stack point* I *is at most the level of the current reflex point* J*, and internal lines (one or two) connecting them can be formed, then a new closed area is formed and the top of stack* I *is popped. Moreover, the*

levels of the newly created lines are higher than the levels of the "parallel" edges of the closed area, and the total length of the newly created lines is matched with the corresponding length of lines at lower level.

Proof. There are two cases for this, the first, is that the reflex angles match with a single straight interior line ℓ generating no new reflex angles. Second case, the reflex angles match with two interior lines ℓ and ℓ' generating a new reflex angle.

i. *Reflex angles I and J match with a single interior line ℓ:*

   In this case, I was at the top of the stack and at least one non-reflex points H and G between as now there are arbitrary angles. As I was on the stack, it cannot be that another reflex I' remains between these two reflex as I' would have been removed from the stack already either through merging with I or with the perimeter before J was encountered. This means that the line $\ell = \overline{IJ}$ is at most the length of the segments of the perimeter that are encountered through moving ℓ across the closed area F, represented by Figure 4.5.

ii. *Reflex angles I and J match with two interior lines ℓ and ℓ':*

   Here, I was at top of the stack, and J is connected along the perimeter by multiple non-reflex points connecting to I. As I and J will merge at a 90 degree angle internally at point V we create two internal lines that meet at V such that $\ell = \overline{IV}$ and $\ell' = \overline{VJ}$ and one is Horizontal and one is Vertical. Through this, in the worst case we have no non-reflex points in the area generating interior lines that are at most $\sqrt{2}c$ such that c is the length of the perimeter line, Figure 4.4.

□

Lemma 31. *Given a level 0 reflex point B on the stack, that is not matched with another reflex point, it will be able to close a convex area F with a level 1 line ℓ from B to a level 0 reflex point V.*

Proof. Without loss of generality, let Z be the top left point of F created by ℓ. Suppose for the sake of contradiction that ℓ has level at least 2.

Then it must be such that the ℓ' "parallel" to ℓ in F is at least level 1 (according to the algorithm), hence was not part of the original perimeter. We will show below that such an ℓ' cannot be created.

There must exist B' that connect to another point, V', either perimeter or reflex, generating ℓ'. For B' to be pushed on the stack, as it is the at least the level of B it must be such that they cannot connect through at most two interior lines. This means that B' cannot exist on the "parallel" line of ℓ to be pushed on the stack. We examine the following cases.

- Reflex point B' exists on left perpendicular to ℓ of F:

  Through Algorithm 6, B' is outside or along a perpendicular line to ℓ of F in order to produce ℓ' to traverse the entire "parallel" line to ℓ. Upon reaching Z, following the algorithm, it must be that all reflex points have either been matched, through Lemma 30 or, they have been connected to the perimeter removing all from the stack. Hence, B' cannot exist outside F on the left perpendicular of ℓ.

- Reflex point B' exists on right perpendicular to ℓ of F:

  In this case, interior lines can be made, but it would take at least three to close an area, which is unable to close. In this scenario, it is impossible to make a line across, as through Algorithm 6, lines can only be made when a V is reached after the reflex point is added to the stack. B' has been visited

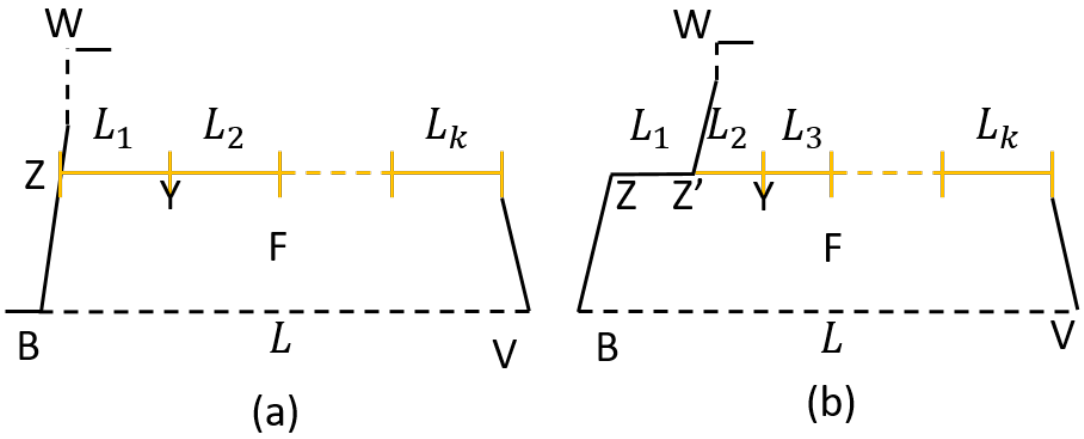

FIGURE 4.6. Reflex Removal Through Perimeter Connection of Arbitrary Polygon

after all the points between B and Z have been visited so a line ℓ' that "parallel" to ℓ cannot be created. In this case, upon reaching V, following the algorithm, we generate perpendicular ℓ' that is a segment of the right perpendicular of ℓ in F

Through both of these cases, it is such that the "parallel" to ℓ' is level 0 and ℓ is level 1. □

Lemma 32. *Given a reflex point B on the stack, which is not matched with another reflex point, it will be able to close with line ℓ from B to a perimeter point V, creating convex areas (at least one) such that the level of ℓ is greater than the maximum level of any segment in the "parallel" line of ℓ in those rectangular areas.*

Proof. When reflex point B was initially encountered, we could not match it with the reflex point on top of the stack, because the stack was empty or top of stack was at a greater level so it had to be placed on the stack. Continuing the traversal, we eventually reach a perimeter position V that shares the same x or y coordinate with B.

B and V can be connected with a single straight line ℓ. Suppose that line ℓ is of level L, which implies that the level of B is $L-1$. In case B was not at the top of the stack, we must now pop off elements from the stack up to reflex point B which creates multiple rectangular areas as shown in Figure 4.3. Without loss of generality, we will focus one of those convex areas (i.e. $F = F_L$). For simplicity assume that ℓ is the lower edge of F and V is its right end, as shown in Figure 4.6.

Let Z be the point at the top left most corner of F (4.6). "Parallel" to ℓ on F we have line ℓ', with one endpoint being Z. The line ℓ' consists of segments $\ell_1, \ell_2, \ldots, \ell_k$, with respective levels $L_1, L_2, \ldots, L_k$ that can be of arbitrary slope (Figure 4.6), such that the end points of each segment are perimeter points (either original perimeter points or artificially created at level 1 and higher). To have these line segments, there have to be visited reflex angles defining them. We need to prove that $L_j < L$, for all $1 \leq j \leq k$.

Suppose that a line l_i was created such that both of its end points were original reflex points of the initial perimeter of level 0. Then according to Lemma 31, the level of ℓ_i is 1. If such ℓ_i exists, then it must be that $L > 1$, since if $L = 1$, then the level of B is 0, and one of the endpoints of ℓ_i would have been matched with B to create a different closed area than F, giving a contradiction.

To continue, suppose that at least one of the ℓ_i was created due to some artificially created reflex point during the course of the algorithm which was not part of the original perimeter (i.e. the level of Y is at least 1). Let Y be the leftmost such artificial reflex point on ℓ which is on some ℓ_{i_Y}. For the sake of contradiction, suppose that $L_i \geq L$.

Without loss of generality, we focus on two cases, $\ell_{i_Y} = \ell_1$ and $\ell_{i_Y} = \ell_2$ as shown in Figures 4.6.a and 4.6.b, respectively. The analysis of these two cases is general enough to cover all the other possible positions of ℓ_i on ℓ'.

i. $\ell_{i_Y} = \ell_1$ *(Figure 4.6.a):*

Let W be the first point along $\overrightarrow{BZ}$ where the current perimeter turns for the first time to the right of Z.

In this case, ℓ_1 connects to Z and Y. There are three possibilities for this case; Y is created from reflex points above and to the right, Y is created from reflex points above and to the left (namely Z is to the left), and $W = Z$.

a. Y is created from reflex points above and to the right:

   In this case, after Y is created, gaining level at least $L - 1$ it connects to a point Z. Through Algorithm 6, if Z is a reflex point connected from below and to the left, then we know that by W we reach a point that matches its x coordinate, and through the algorithm we would pop off all reflex points to Z generating that many convex areas. Thus, Z is not a reflex point when W is reached, which comes before Y in the traversal, so ℓ_1 is not created at all.

b. Y is created from reflex points above and to the left:

   In this case, Y connects directly to Z and another reflex point above Y. For this to be possible, Z must be a reflex point on top of the stack when that other point is reached. As shown in case [a.], when W is reached Z no longer can be reflex and as such, could not merge with another reflex point. Y cannot be generated in this way.

c. $W = Z$:

   In this case, as since W is the point that turns to the right of Z and Y connects to Z, then it must be such that Y was not artificial.

Through this, Y cannot connect to Z in any of these cases thus ℓ_1 is not generated. Also, if Y was greater level than B it must be that Y would have merged with B as it was assumed to be the first artificial vertex and if any other reflex existed between W and Y then they would have merged creating a closer vertex along ℓ' at a higher level, causing a contradiction for Y being the first artificial reflex (this process continues until all other reflex points are popped finally merging with B).

ii. $\ell_{i_Y} = \ell_2$ *(Figure 4.6.b):*

Let $\ell_1 = \overline{ZZ'}$. Let also W be the first point along the "parallel" of $\overrightarrow{BZ}$ that goes through Z' where the current perimeter turns for the first time to the right of Z'.

Then, ℓ_2 connects to Z'. Moreover, Z and Z' must exist along the original perimeter as Y is the first artificial reflex point created along ℓ'. To begin, from the traversal from B to Z it could be similar to the first case from B to W such that all the reflex points between them have been removed. There are two cases for Z', where its level is equal to B or the level of Z' is less than B.

a. Level Z' = Level B:

Through this case, as Z' is at a level equal to B, it must be that Level $B = 0$. For this, it must be that B is on top of the stack, due to all other reflex being removed upon reaching Z, and Z' would have had to connect to B thus ℓ would not be created.

b. Level $Z' \neq$ Level B:

In this case, Z' is level less than B, it is then that Z' is pushed on top of the stack with B as the next element. As Z' is a lower level than B then

there are different cases similar to case [i.] are possible; Y is created from above and to the right, Y is created from above and to the left, and $W = Z'$.

1. Y is created from reflex points above and to the right:

   Through this case there exists a reflex point above Y that is on top of the stack when the reflex point to the right of Y is on top of the stack. Similar to case [i.][a.] upon reaching W all reflex points have been removed between Z' and W putting Z' on top of the stack. As Z' is on top of the stack, the next reflex either one that was earlier than the one above Y or the one that is above Y.

   A. next reflex point after W appears before the one above Y:

      Given that Z' is level 0 and on top of the stack, the next reflex points that is encountered, after visiting W will be able to merge with Z'. Through this, an artificial reflex point is created along ℓ_2 that would connect to Z'. This is a contradiction as Y is the next reflex point to connect to Z'.

   B. next reflex point after W appears above Y:

      In this case, given that Z' is level 0 and top of stack, as with [A.] it has to be that the next encountered reflex point will merge with Z'. In this, Z' connects with the reflex point above Y. Through merging these points, the artificial reflex point crated along ℓ_2 would be Y. This is a contradiction as it was assumed that Y was created from above and to the right, but due to this, Y is created from above and to the left.

2. Y is created from reflex points above and to the left:

   In this case, Z' is one of the reflex points merged to create Y. As the point above Y was the current point in the traversal, to merge with Z', it must be such that Z' is on top of the stack. When merged, Z' is popped from the stack, making B top of stack and creating Y. In this case, Y is the level of ℓ_2, namely,at least L. As B is at most $L-1$ and B is on top of the stack, Y would have merged with B, which is impossible.

3. $W = Z'$:

   As in case [i.][c.] if $W = Z'$ then it must be that Y is not artificial, which contradicts the assumption that Y is artificial.

Given the above cases, it cannot be such that Y is a higher level than B and ℓ_{i_Y} to be created. These cases are repeated and for all ℓ_i up to ℓ_k.

□

Theorem 8. *At the end of the traversal there are no remaining reflex points on the perimeter.*

Proof. We only need to show that the stack of points that define reflex angles at the end of traversal will be empty. There are three possible cases for the stack being empty at the end of traversal; no reflex point added to stack, reflex angles are matched, we match top of stack with later explored perimeter point, or the last three reflex are handled by the reduction of Algorithm 6.

i. No reflex point added to stack: From Lemma 32 case i, if all reflex points are able to connect with the perimeter upon reaching the reflex point, then the stack is always empty.

ii. Reflex angles are matched: Given Lemma 30 we have two possible cases. For Lemma 30 case i, we pop to top of the stack off and no new points are pushed on the stack. In Lemma 30 case ii, we pop the top of the stack off and generate an increased level reflex point V. With this, we then attempt to match V with the next element utilizing case ii, connect it to the perimeter with case i. If these are not possible, then push V on the stack and later match (case ii) or connect to the perimeter (case iii).

iii. Matching top of stack with later explored perimeter point: Through Lemma 32 case ii, we have pushed a reflex point on the stack and it has yet to be matched with another reflex. In this case, we connect the point with the perimeter and pop the point from the stack.

iv. The last reflex, three reflex points on stack, could not be matched and we could have previously reached the perimeter which we cannot close. Due to this, we have to handle this after finish the traversal by Algorithm 6. This is because we must finish traversing all the points and re-reach the perimeter point(s) that matches the last reflex in order to close the convex area.

□

Lemma 33. *A reflex point at level $L \geq 2$ for its creation it requires at least three reflex points at level $L-2$ or lower.*

Proof. Consider a reflex point ϕ at level $L \geq 2$. Then, from Algorithm 6, ϕ must have been created was created by matching two reflex points θ_1 and θ_2, where θ_2 was visited or created more recently than θ_1 such that:

- θ_2 was at level $L-1$. Then the creation θ_2 had to use two reflex points at level $L-2$ or lower, since the creation of a reflex always increases the level,

and those were different than θ_1. Thus, three points at level $L-2$ or lower were needed.

- θ_2 was at level $L-2$. Consider the closed area F which was defined when ϕ was created. On the perimeter of F there must be a line segment ℓ with level $L-1$. The line segment ℓ must have been created when we closed some reflex point ζ at level $L-2$ with a straight line to the perimeter across. Thus, three points at level $L-2$ or lower were used.

□

Theorem 9. *There are at most* $2\log_3(3r)+1$ *levels and in total* $9r+2\log_3(3r)+2$ *convex areas created.*

Proof. From Lemma 33, every two levels reduce the total number of reflex points by a third. Through this process, as we have three points pushed on the stack per reflex point r, we have at most $2\log_3(3r+1)$ levels until there are no more reflex points.

From level 0, there are r reflex points, pushing three points on the stack, and connecting them to the perimeter produces $3r+1$ convex area. For arbitrary even level $i \geq 2$ there are at most $3r/3^i$ reflex points that produces $3r/3^i+1$ convex areas. Given there are at most $2\log_3(3r+1)+1$ levels, the total convex areas from even levels is $\sum_{i=0}^{\log_3(3r+1)}(3r/3^i+1)$ which is at most $6r+\log_3(3r+1)+1$.

In level 1, there are at most $r/2$ reflex points, pushing $3r/2$ points on the stack, which produces $3r/2+1$ convex areas. An arbitrary odd level $j \geq 3$ there are at most $3r/(2\cdot 3^{j-1})$ reflex points that produces $3r/(2\cdot 3^j+1$ convex areas. The total convex areas from odd levels is $\sum_{j=1}^{\log_3(3r+1)+1}(3r/(2\cdot 3^{j-1})+1)$ which is at most $3r+\log_3(3r+1)+1$.

Summing the even and odd levels we produce a at most $9r+2\log_3(3r)+2$ convex areas.

The final reflex point is a special case as this one may not be matched during the first traversal to each point. In this case, we produce one additional level giving a total of $2\log_3(3r+1)+1$ levels. This does not change the total number of convex areas created as this reflex will only be used once and fits in one of the two summations above.

□

Given that P is the length of all the perimeter lines.

Lemma 34. *Any level has at most $\sqrt{2}P$ total interior line length.*

Proof. From Algorithm 6 levels reached through merging reflex points produces lines ℓ of the greatest level of all perimeter lines in the closed area, thus it is at most the length of the perimeter contained in the closed area. Through Lemma32, levels reached through connecting reflex point to perimeter with interior line ℓ. The length of interior lines of any convex area are at most $\sqrt{2}c$ such that c is the length of perimeter of the area, Figure 4.4. From this, since P is the length of the perimeter, when $c = P$ we get that the total length of all convex areas is at most $\sqrt{2}P$. □

Lemma 35. *The total length of all interior lines is $\sqrt{2}P(2\log_3(3r)+1)$.*

Proof. From Theorem 9 there are at most $2\log_3(3r)+1$ levels and since each level has at most length $\sqrt{2}P$, Lemma 34, the total length of all interior lines is $\sqrt{2}P(2\log_3(3r)+1)$.

□

Conclusion

This thesis proposes multiple algorithms to perform exploration on various geometric areas. Each of these algorithms are designed in such a way that robots can perform the task. Each algorithm I have developed is an online algorithm that does not initially know the configuration of the geometric area and completing the task in a single pass. In each of these, the algorithms are asymptotically optimal (Chapter 1, achieve average (Chapter 2), or introduce a concept previously not researched (Chapters 3 and 4).

In Chapter 1, the algorithm developed was a parallel algorithm with slow start, for multiple agent to perform parallel exploration with a limited communication range. Through the introduction of unknown holes in the area, the challenge becomes more difficult to maintain equal work for all robots. As the area and the holes are unknown, the robots begin by traversing until they can close a room to explore. Once there, the room exploration algorithm of leader with slow start was developed to minimize the cost of returning to the leader. This process of slow start reduces the repeated work of the robots through only activating enough agents that the room can support. Through exponential back-off with a reset to communicate with the leader, communication costs are decreased as well as the cost of returning to communicate. With my algorithms, I achieve an overall performance of $O(N/k)$ agent work in asymptotically optimal time $\Theta(N/k)$.

As I began into convex decomposition, with Chapter 2, I developed an online algorithm that is able to achieve $r+1$ total convex areas. Through this algorithm, it can be achieved with a robot that is exploring the perimeter creating a single interior line per reflex point removing them upon reaching them only taking a single traversal around the perimeter. As there is the better case, namely making r total areas, it is not possible in an online algorithm as it involves connecting

multiple reflex points together while still using a single line. To perform this, it would take knowledge of the existing reflex to make a decision.

Although this achieves $r+1$ convex areas, the problem is the amount of movement that the robot would have to traverse. In this algorithm, the interior cuts are up to Pr total length, which would take a robot to traverse twice to move to the other end and back. This means that to complete convex decomposition, in total the robot would traverse $P(2r+1)$. This was the inspiration to minimize the interior cut-length to reduce the length of the interior lines as the robot at least needs to traverse around the perimeter.

Minimizing cut-length, as in Chapter 3 and 4, has not been previously done. This is an important area as we develop more reliance on robots in order to minimize the total work while also not generating too many additional convex areas. To perform minimize cut-length, I began with solving the problem on a rectilinear polygon. Although this generates $3r + 2\log_3(r) + 2$ rooms, a little over three times that of Chapter 2, I achieve total interior length $P(2\log_3(r) + 1)$.

To truly to compare minimize cut-length with convex decomposition, the algorithm from Chapter 3 must be reduced to be performed on arbitrary polygons, Chapter 4. This reduction results in three times the number of reflex points pushed on the stack. This causes a total of $9r + 3\log_3(3r) + 2$ convex areas with interior line length of $\sqrt{2}P(2\log_3(3r) + 1)$. Therefore, the total traversal for a robot to complete would take $P + 2\sqrt{2}P(2\log_3(3r) + 1)$.

My future work is to implement the algorithms. Through this, I am able to observe the average case run time. Not only do I want to implement the algorithms through programming, but also implement it with some robots to better observe the time complexity.

Next, I plan on to research the algorithm from Chapter 1 with the introduction agent failure. With agent failure, during room scan an agent can fail when active. In doing so, the leader would have to notice the agent did not return in the expected time and send an inactive agent to perform the failed agent's task. During this research, there must be at least one agent that does not fail and the aim would be to minimize the repeated work.

I would also remove some of the constraints to this algorithm, such as being orthogonal and the rectangular holes, to be able to explore arbitrary concave polygons. Without the holes being rectangular, it becomes a more difficult task as they no longer have a bounded aspect ratio. In no longer being orthogonal, it becomes more difficult to determine when a "room" should be made as well as performing room scan on it as the angles are no longer 90, 270, or 360 degrees.

With Convex Decomposition, Chapter 2, my future goal is to reduce the cut-length while maintaining the $r + 1$ convex areas. Doing this in an online manner requires more knowledge about the surrounding area in order to remove the reflex instead of performing a straight cut to remove the reflex but still must make a single cut to remove the reflex.

In the last Chapters, 3 and 4, my future goal would be to minimize the amount of convex areas generated while maintaining interior cut-length. After this, I would like to relax the constraints further in order to have continuous areas such that there are curves and not hard angles. The difficulty with this scenario is that I need to be able to approximate convex as the curvature makes it so the algorithm would generate areas that must allow for an error of concavity.

References

[1] Pankaj K. Agarwal, Eyal Flato, and Dan Halperin. Polygon decomposition for efficient construction of minkowski sums. *Computational Geometry*, 21(1-2):3961, 2002.

[2] Vaibhav Agrawal. Multi robot area exploration using modified frontier algorithm. *International Journal of Computer Engineering and Applications*, IX, May 2015.

[3] Tauhidul Alam, Leonardo Bobadilla, and Dylan Shell. Minimalist robot navigation and coverage using a dynamical system approach. In *IEEE Robotic Computing*, 2017.

[4] Chanderjit L. Bajaj and Tamal K. Dey. Convex decomposition of polyhedra and robustness. *SIAM Journal on Computing*, 21(2):339364, 1992.

[5] Nicola Basilico and Francesco Amigoni. Exploration strategies based on multi-criteria decision making for searching environments in rescue operations. *Autonomous Robots*, 31(4):401–417, 2011.

[6] M. A. Batalin, G. S. Sukhatme, and M. Hattig. Mobile robot navigation using a sensor network. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 636–641 Vol.1, April 2004.

[7] Bernard Chazelle and Leonidas Palios. Triangulating a nonconvex polytope. *Discrete Computational Geometry*, 5(5):505526, 1990.

[8] Herbert Edelsbrunner, Arch D. Robison, and Xiao-Jun Shen. Covering convex sets with non-overlapping polygons. *Discrete Mathematics*, 81(2):153164, 1990.

[9] Pooyan Fazli. Fault-tolerant multi-robot area coverage with limited visibility. In *ICRA 2010 Workshop on Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees*, 2010.

[10] Pooyan Fazli, Alireza Davoodi, and Alan Mackworth. Multi-robot repeated area coverage. *Autonomous Robots*, 34.4:251–276, 2013.

[11] Paola Flocchini, Matthew Kellett, Peter Mason, and Nicola Santoro. Map construction and exploration by mobile agents scattered in a dangerous network. In *IEEE International Symposium on Parallel and Distributed Processing*, 2009.

[12] Joshua Freeman. Robot assisted wireless sensor network for monitoring and detection of explosives in indoor environment. 3, May 2011.

[13] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. 61:12581276, December 2013.

[14] Andrea Gasparri, Bhaskar Krishnamachari, and Gaurav S. Sukhatme. A framework for multi-robot node coverage in sensor networks. 52:281–305, April 2008.

[15] Julian Hoog, Stephen Cameron, and Arnoud Visser. Role-based autonomous mulit-robot exploration. In *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009.

[16] J. Mark Keil. Polygon decomposition. *Handbook of Computational Geometry*, page 491518, 2000.

[17] Mark Keil and Jack Snoeyink. On the time bound for convex decomposition of simple polygons. *International Journal of Computational Geometry Applications*, 12(03):181192, 2002.

[18] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. 3:24– 33, November 2004.

[19] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polygons. *Computational Geometry*, 35(1-2):100123, 2006.

[20] Ahmad Masoud and Ali Al-Shaikhi. Time-sensitive, sensor-based, joint planning and control of mobile robots in cluttered spaces: A harmonic potential approach. In *54th IEEE Conference on Decision and Control (CDC)*, 2015.

[21] Bin Mu and Spiridon Bakiras. Private proximity detection for convex polygons. *Tsinghua Science and Technology*, 21(3):270280, 2016.

[22] Christian Ortolf and Christian Schindelhauer. Online multi-robot exploration of grid graphs with rectangular obstacles. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pages 27–36. ACM, 2012.

[23] Nunzia Palmieri, Xin Yang, and Salvatore Marano. Coordination techniques of mobile robots with energy constraints. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2016.

[24] A. Pierson, L. C. Figueiredo, L. C. A. Pimenta, and M. Schwager. Adapting to performance variations in multi-robot coverage. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 415–420, May 2015.

[25] Dip Ray, Amit Mandal, Somajyoti Majumder, and Sumit Mukhopadhyay. Human-like gradual multi-agent q-learning using the concept of behavior-based robotics for autonomous exploration. In *IEEE International Conference on Robotics and Biomimetrics*, 2011.

[26] Joshua Reich and Elizabeth Sklar. Robot-sensor networks for search and rescue. January 2006.

[27] Arles Rodriguez, Jonatan Gomez, and Ada Diaconescu. Foraging-inspired self-organisation for terrain exploration with failure-prone agents. In *IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*, 2015.

[28] Nicholas Roy and Gregory Dudek. Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations. *Autonomous Robots*, 11(2):117–136, September 2001.

[29] Sajad Saeedi, Liam Paull, Michael Trentini, Mae Seto, and Howard Li. Group mapping: A topological approach to map merging for multiple robots. *IEEE Robotics and Automation*, 21(2), 2014.

[30] Imran Sharif, Debasis Chaudhuri, Naveen Kumar Kushwaha, Ashok Samal, and Brij Mohan Singh. An efficient algorithm to decompose a compound rectilinear shape into simple rectilinear shapes. *Turkish Journal Of Electrical Engineering Computer Sciences*, 26:150161, 2018.

[31] Rahul Sharma, Daniel Honc, Frantiek Duek, and Gireesh Kumar T. Frontier based multi robot area exploration using prioritized routing. In *30th European Conference on Modelling and Simulation*, pages 25–30, June 2016.

[32] Wheihua Sheng, Qingyan Yang, Jindong Tan, and Ning Xi. Distributed multi-robot coordination in area exploration. *Robotics and Autonomous Systems*, 54, 2006.

[33] T.c. Shermer. Recent results in art galleries (geometry). *Proceedings of the IEEE*, 80(9):13841399, 1992.

[34] Alberto Veseras, Thomas Wiedemann, Christoph Manss, Lukas Magel, Joachim Mueller, Dmitriy Shutin, and Luis Merino. Decentralized multi-agent exploration with online-learning of gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[35] Joe Warren, Scott Schaefer, Anil N. Hirani, and Mathieu Desbrun. Barycentric coordinates for convex sets. *Advances in Computational Mathematics*, 27(3):319338, 2006.

[36] Jasmine Xavier and Kumari Shantha. Multi robot coordination in area exploration based on a fixed linear chain bluetooth communication. *Australian Journal of Basic and Applied Sciences*, pages 362–373, 2013.

[37] Y. Zhou and A.N. Zincir-Heywood. Intelligent agents for routing on mobile ad-hoc networks. In *Conference on Communication Networks and Services Research*, 2004.

Vita

Wyatt P. Clements was born October 18, 1990 in Lewes, Delaware. He earned a bachelors of art degree in mathematics and computer science and a bachelors of science degree in web development at Thiel College May 2014. In August 2014 he came to Louisiana State University to pursue graduate studies in computer science. During his studies at Louisiana State University, he was as a subreviewer for SPAA 16, ALGOSENSORS 16, SSS 16, IPDPS 17, and ICDCN 18. In Fall 2017 he was the instructor for Computer Science I for Majors where he taught thirty students. He is currently a candidate for the degree of Doctor of Philosophy in computer science, which will be awarded August 2019.