5-AXIS COVERAGE PATH PLANNING WITH DEEP REINFORCEMENT LEARNING AND FAST PARALLEL COLLISION DETECTION

A Thesis
Presented to
The Academic Faculty

By

Xin Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science, College of Computing

Georgia Institute of Technology

May 2020

5-AXIS COVERAGE PATH PLANNING WITH DEEP REINFORCEMENT LEARNING AND FAST PARALLEL COLLISION DETECTION

Approved by:

Dr. Richard Vuduc, Advisor
School of Computational Science and Engineering, College of Computing
Georgia Institute of Technology

Dr. Thomas Kurfess
The George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

Dr. Ümit Çatalyürek
School of Computational Science and Engineering, College of Computing
Georgia Institute of Technology

Dr. Jeff Young
School of Computer Science
Georgia Institute of Technology

Dr. Thomas M. Tucker
Tucker Innovations Inc.
Tucker Innovations Inc.

Date Approved: April 20, 2020

To my parents: Dengxiu Zou and Yongzhen Chen,

for their sacrifice, dedication, and support.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my advisor, Richard Vuduc, for all his help and support. He is so wise, insightful, and patient. He taught me so many things, such as how to present an abstract idea and how to build self-confidence. Many of the ideas and thoughts he's offered turn out to be applicable in many places. He always helps me clarify my confusion patiently, no matter the question. He is also very nice and helpful. I still remember that when the scheduled room for my thesis proposal became unavailable due to a delay from a previous meeting, he directly put a note on the door directing others to another available room before I even noticed it. I am sincerely grateful for all of his guidance.

I would also like to thank my previous advisor, Karsten Schwan (deceased). He was very helpful and supportive. His keen interest in science and technology made him my role model forever. He will be missed.

I would like to thank the members of my Ph.D. thesis committee—Dr. Thomas Kurfess, Dr. Ümit Çatalyürek, Dr. Jeff Young and Dr. Thomas Tucker, for their help and valuable feedback. Their generous offerings of their time has been greatly appreciated.

I have been fortunate to work with many thoughtful and smart people: Costin Iancu, Matthew Wolf, Fang Zheng, Kun-Lung Wu, Ymir Vigfusson, and Douglas Blough. I am grateful for their wisdom, insightful ideas, valuable life experiences, which have influenced me a lot.

I also cherish my time with other talented graduate students in our lab, Srinivas Eswar, Michael Isaev, Zhihao Li, and Patrick Lavin. I have learned a lot from them, on both research and life in general. Because of these guys, graduate school is full of enjoyable memories that I will miss.

I am very grateful to my family, my wife and two lovely kids: Liting Hu, April Chen, and Angela Chen. They are the true sources of my happiness. I want to thank them for

their support.

Lastly, I want to thank my mother and father: Dengxiu Zou and Yongzhen Chen, for all your unconditional love. My father always had high expectations of me, which helped to push me forward without being lost. His hard-working spirit has been a constant source of motivation and encouragement. My mother is the person I owe the deepest gratitude. She has helped me take care of my two kids since they were born. I cannot imagine how much effort she has expended to support my family and me, undertaking it all while a stranger in a foreign country. This thesis would have been impossible without her support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The goal of coverage path planning problems is to generate an efficient path that can fully cover all reachable points in a given environment. It has numerous applications, including advanced manufacturing and robotics, which are two targets of our work. Most prior approaches to this problem are designed for specific features of a given application (e.g., how the robot can move and rotate), meaning they may "hard-code" properties or constraints of the application into the problem formulation or heuristics of a solution algorithm. While such methods can be very effective, they can also be hard to extend, as altering the formulation or constraints may require extensive changes to the solver algorithm and heuristics.

In this dissertation, we investigate a new generic method of constructing an efficient and collision-free paths for coverage path planning problems. It unifies and generalizes broad classes of cover path planning problems for two-dimensional environments into a general, unified, and automatically adaptive framework, which we refer to as *adaptive deep path* (AD Path). AD Path can improve path efficiency with respect to cost models that consider both path length and the number of turns, and can flexibly accommodate different problem configuration options, such as robot sizes and robot motion strategies (i.e., how the robot is allowed to move or rotate). This method helps programmers avoid reformulating the underlying problem or hard-coding heuristics from such configuration options, and through automatic exploration, can generate better paths as a result than previously proposed methods.

In moving from 2-D environments to 3-D ones, we consider two additional problems.

One is how to improve the performance of collision detection in the process of covering 3D objects, which is a well-known performance bottleneck of existing work. Previous approaches (e.g., tool path planning problems) tend to simplify the collision computations by simplifying the geometry of the robot or tool or assuming that object is a simple surface,

both of which are unrealistic in most practical applications. We propose a new method that allows for more general shapes, uses geometric insights to reduce computational costs, and exploits parallel computing via graphics co-processing (GPUs) to solve CD problems for more realistic geometries and much higher speeds than state-of-the-art methods. We show a proof-of-concept comparison against an existing commercial tool to show the efficacy of our approach.

Another problem in 3-D is how to construct an efficient path for covering the surface an arbitrary object, which finds applications in advanced manufacturing. We propose a novel algorithm that overcomes a key limitation of more conventional methods, which assume no collisions by simplifying the tool or robot geometry. We remove this simplification by incorporating geometric accessibility information when planning a path. Our proposed algorithm enables the optimization of both the path lengths and the cost of handling other constraints.

Taken together, this dissertation addresses several aspects of coverage path planning problems, making them more practical in both 2-D and 3-D planning environments with more realistic object geometries and cost metrics (e.g., path length, number of turns). As such, we believe it can motivate new work in robotic path planning tasks.

CHAPTER 1

INTRODUCTION

The classical *Coverage Path Planning* (CPP) problem concerns the task of constructing a path that *covers* (or touches or passes over) all points of a given environment, such as a 2D area or the surface of a 3D object, while avoiding obstacles. This task is integral to many applications in robotics and advanced manufacturing, such as vacuum cleaning robots [1, 2], painter robots [3], autonomous underwater vehicles creating image mosaics [4], demining robots [5, 6], lawn mowers [7], inspection of complex underwater structures [8], 3D printing [9, 10, 11], agriculture [12], search and rescue [13], aerial coverage [14, 15], physical simulations [16], motion planning [17, 18]; and virtual assembly [19], autonomous underwater vehicles (AUV) [20, 21, 22], to name just a few.

One of the main motivating examples of CPP that motivates our work comes from CNC milling applications shown in Figure 1.1. The left side presents an original object that is given as input, and the right side is our target object after applying a milling process. To automate this process, a common way is to simulate the process on a software to generate machining codes, and then apply the codes to a milling robot on real physical objects. The

Figure 1.1: An example of using CPP problem in CNC milling application that converts a cylinder stock into a 3D object (e.g.. king kong).

software simulates the process on virtual objects, while the milling robot applies the codes on physical objects. As shown in Figure 1.1, the virtual objects are on the left side of the two dashed frames, while the physical objects are on the right side of the two dashed frames. A key function of this software is to construct an efficient path to cover an arbitrary 3D object. The generated path corresponds to the output codes used for the milling process. Note that a milling process that fully achieves the conversion from a stock to a final object requires multiple steps of the CPP problems, where each step machines off materials proportional to the size of the robot, see details in Section 4.3.1.

1.1 Problem statement

The input is an environment represented by a bitmap. Figure 1.2 illustrates an example of the bitmap representing an environment. There are two types of space: the free space denoted by the empty area and the obstacle space denoted by the black area. The environment can be a 2D plane or a surface of a 3D object. The output is an ordered path that visits all free bits in the environment. The problem is to construct a path for a robot that covers all free space while avoiding the obstacles. Note that the bit does not corresponds to a pixel in the object model, but an area that can be covered by a robot without moving. The size of the robot determines the size of a single bit. The basic requirements of the problem are the following:

- Cover the free space in the environment
- Avoid obstacles
- Optimize the path efficiency.

This is an optimization problem, where the target is to maximize the path efficiency under the constraint that all bits have to be covered and the obstacles need to be avoided. It is a challenging problem because of its high computational complexity. Imagine that the robot is located at a random position. From there, it must choose a neighbor to visit.

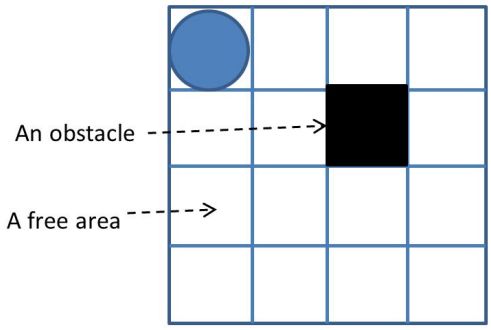


Figure 1.2: A bitmap representing an environment.

Covering a cell that has already been covered is allowed, for otherwise it is highly likely that the robot might get trapped in an area where all neighboring cells are covered. The complexity of constructing this type of an ordered path is, in general, exponential [23].

1.2 Problem Variations

Due to its high computational complexity, practical algorithms for CPP must be solved approximately. Such methodologies have a long history, and there are several problem variations that have been developed to simplify the problem and reduce its complexity. An important assumption made by these related problems is that all the free space in the environment can be represented by a graph, where each vertex or edge represents a group of cells that the robot needs to cover. This assumption is a simplification because there are many ways to perform this conversion. One example is the *boustrophedon cellular decomposition* (BCD)‘[24]. In BCD, the vertices in the graph are a collection of coarse-grained cells, where each cell has no obstacles and no overlaps with other cells. Other related problems are listed as follows.

- *Traveling salesman problem (TSP).* In TSP, one seeks the shortest path that visits all the vertices. It is a well-known NP-hard problem. LKH TSP [25] is an efficient library that generates a short path for TSP. Another variant of TSP is called the *covering salesman problem*. Instead of visiting each city (vertex) in TSP, the robot must

visit a neighborhood of each city [26]. Our CPP problem is different. It requires the robot to pass over all points, in contrast to visiting all the neighborhoods as in the covering salesman problem.

- *Chinese postman problem*. Here, the goal is to construct an Euler tour that visits all edges in the graph with the shortest tour length [27]. It differs from TSP in that it seeks to cover edges rather than vertices. As such, the edges in the graph correspond to the spaces in the environment that a robot needs to cover.

- *Lawnmower problem*. Here, one seeks a path to cut all of the "grass" in a region covered by grass. This problem is also NP-hard [28]. The difference between the lawnmower problem and CPP is that there are no obstacles in the lawnmower problem.

- *Piano mover's problem*. In this problem, one wants to find a collision-free path from a start to a target, which has been shown to be PSPACE-hard [29, 30]. The optimization is to reduce the path length.

- *The art gallery problem*. In this problem, one seeks the minimum number of guards (points) to cover all points in a polygonal gallery [31], which is also a well-known NP-hard problem. Instead of going through each point for to achieve full coverage, as in our CPP problem, this problem only needs to find a small number of guards where choosing a single guard corresponds to a coverage of the points visible to this guard.

- *The watchman route problem*. Here, the aim is to find the shortest route from a given point back to itself. This problem resembles the coverage goal in the art gallery problem, where all the points in the environment have to be visible to a chosen point in the route. Simple cases of the watchman route problem, such as covering the interior of simple polygons, can be achieved in polynomial time [32]. However, in general this

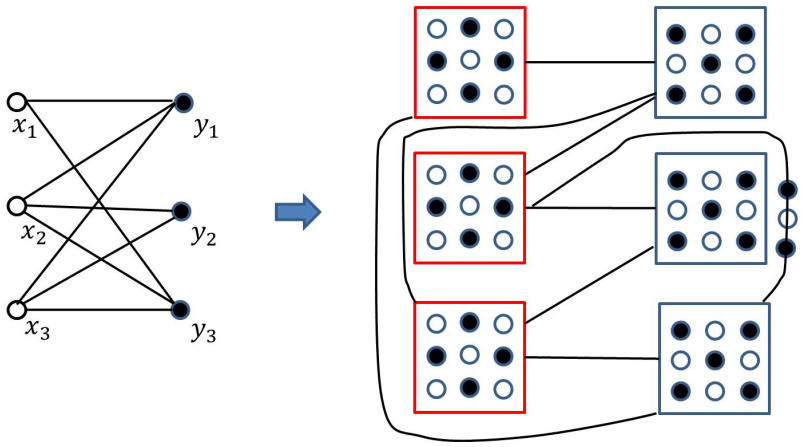


Figure 1.3: A planar bipartite graph G, with maximum vertex degree 3 as the first step of converting the law mowing problem to Hamiltonian Circuit problem.

problem is proved to be NP-hard [33]. The difference between the watchman route problem and the art gallery problem is that the watchman route problem optimizes the route length while the other minimizes the number of the chosen points on the route.

1.3 NP-hardness of the lawn mowing problem

Here we use the lawn mowing problem as an example of proving the NP-hardness of the CPP problem. Arkin proves that the lawn mowing problem for a connected polygonal region is NP-hard for the case of an aligned unit square cutter [28]. The proof makes use of the reduction from the (NP-hard) problem Hamiltonian Circuit in Planar Bipartite Graphs with Maximum Degree 3 to the problem of Hamiltonian Circuits on Grid Graphs, as used by Johnson and Papadimitriou [34].

The proof consists of two steps. Given a bipartite graph G that has n vertices with two types x, y, the first step is to construct a new graph $\hat{G}$, where the x vertices center at the white circles and the y vertices center at the black circles. In the left side of Figure 1.3 illustrates an example of bipartite graph with maximum degree 3 and the right side gives the constructed graph. In the constructed graph, the vertices are surrounded by the vertices

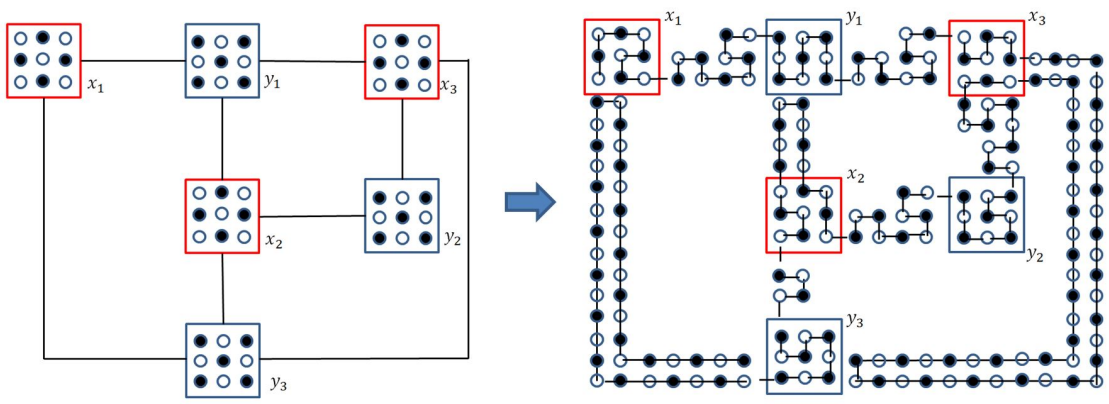


Figure 1.4: The second step of defining regions to be the union of all placements of all vertices and edges in the new graph $\hat{G}$.

of the other type, following the rules of traversing bipartite graph, either form x to y or from y to x. The edges are the same way. If the entrance vertex and the exit vertex on an edge are of the same type, a vertex of the other type needs to be added on the edge. The new graph $\hat{G}$ has m vertices, where $m = O(n)$. Therefore, $\hat{G}$ has a Hamiltonian circuit if and only if G has a Hamiltonian circuit.

The second step is to define the region R to be the union of all placements of all vertices in the new graph $\hat{G}$, where each vertex corresponds to a unit square. The right side of Figure 1.4 presents the corresponding region of the new graph $\hat{G}$. The left side of the Figure 1.4 is the same as the right side of the Figure 1.3, drawn in a different order. The path shown in the region R corresponds to a traversing order of the bipartite graph G: $\{x_1 -> y_2 -> x_3 -> y_2 -> x_2 -> y_3\}$. Therefore, the existence of a tour of length m on the vertices of the new graph $\hat{G}$ implies the existence of a lawn mower tour of length m.

On the other hand, a lawn mower tour of length m can mow all of R only if no point in the region is mowed more than once. This lawn mower path partitions the region R into non-overlapping strips (rectangles) of width 1. In this way, we can construct a graph with each rectangle corresponding to a vertex. With this graph, the lawn mower path implies that a lawn mower tour of length m induces a tour of length at most m in the new graph.

Besides these variant problems, there are some other approaches that propose to use a

combination of the problems mentioned above to solve the CPP problem. For example, in the coverage of 3D environments, the art gallery problem or the watchman route problem can be combined with the TSP problem or the Chinese postman problem [27]. The art gallery problem and the watchman route problem can be used to choose the candidate guards. If the robot needs to visit all points, each guard needs to generate a path from the points visible to the guard. If the robot takes the visibility as a coverage and only needs to cover the guards, a path that visits all guards can be optimized by applying TSP problem or Chinese postman problem.

1.4 Challenges

Despite the research on the CPP problem, there are still opportunities for significant improvement. Motivated by the requirement of our CNC milling application, this thesis, we focus on three perspectives: flexible metrics of evaluating a path, fast parallel collision detection and path optimization under the constraint of being collision-free.

Firstly, prior approaches usually focus on the optimization of one metric of path efficiency. Typically, there are two common metrics as the optimization target: minimizing the path length and minimizing the number of turns in the path. Some projects aims at reducing the path length [24, 15, 14, 35] , while other projects emphasize how to minimize the number of turns [36]. These approaches employ a static metric such that the design of the algorithm can "hard-code" the metric together with some other application-level policies, such as motion strategies on how the robot can move. This approach significantly restricts its application in a way that a small change in the application will make the "hard-code"invalid.

In reality, the metrics used to evaluate a path may vary widely, which depends on the characteristics of the robot or the tool. For the robot that moves fast in straight lines but rotates slow when changing the moving direction, a path that has a small number of turns is preferable, because each turns demands a rotate operation and thus consumes a long time.

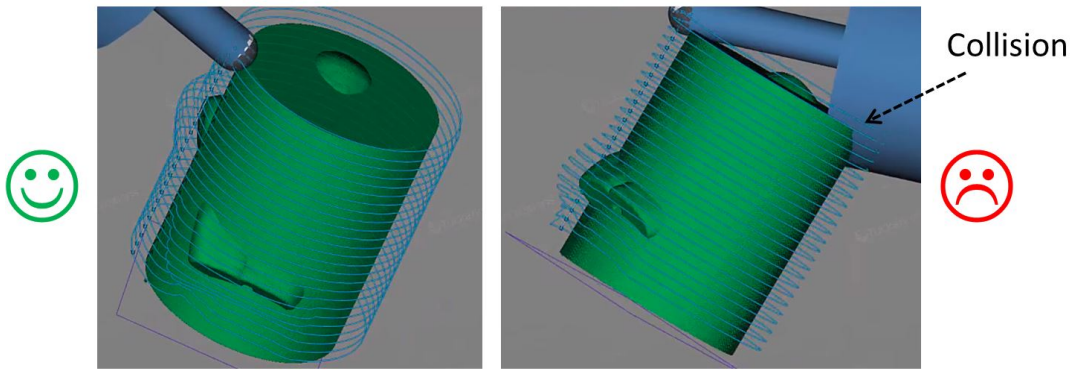


Figure 1.5: Two orientations of the milling robot on the same pivot point. The left orientation is collision-free. The right orientation causes a collision.

For the robot that has the opposite characteristics, a path that has a short path length is preferable, because the cost of moving in lines is higher than the cost of the rotating. Some applications require the support of various policies of motion. For example in 3D printing, a robot may select multiple machining tool sizes and employ various motion strategies, such as moving horizontally and vertically.

Secondly, in traditional problem setting, the robot is abstracted to a single point that covers an environment, ignoring the necessity of avoiding collision on path. In the process of covering an arbitrary environment, one of the basic requirements is to avoid obstacles. However, 2D and 3D environments pose different challenges. In a 2D environment, the robot can be abstracted to a point, because the only place the robot needs to touch is one end of the robot and the other parts of the robot are collision-free. The path generated for coverage just needs to bypass obstacles to avoid collision. However, the 3D case is more complex. Both the robot and the environment can have arbitrary shape. Any collisions between them are not allowed, except for the one that the end of the robot needs to touch the 3D environment for a coverage (in a 3D milling application, the cutting end of the robot needs to stay on the surface of the 3D object to machine the materials off, which is called the pivot point. This point is the only place that the robot is allowed to touch with the target object). Figure 1.5 presents an example of placing the robot on two orientations to cover the same point on the surface of the object. On the left, there is no collision while

a collision exists on the right side. The collision could either break the target object or damage the cutting robot, both of which are detrimental and crash the milling process.

However, in reality the robot has an arbitrary shape that is likely to collide with the target environment. Thus, we can not simply assume that the robot is just a tiny dot. Different from traditional problem setting, this thesis answers how to determine the accessible orientations to avoid collisions. Due to high resolutions of the target environment and high resolutions of the orientations (e.g.. a vast number of voxels in the target environment and a vast number of the orientations to check), the process of calculating the accessibility of orientations is computation-intensive and thus consumes a large amount of time. We present a new abstraction and a parallel algorithm that can significantly accelerate this process .

Last but not least, there is aa challenge in applying the constraint of avoiding collisions when constructing a path. The second challenge is about a fast way of determining accessible orientation, but how to use its output to construct a collision-free path is another problem. To facilitate autonomous path planning for complex environments or objects, previous approaches usually ignore the robot constraint of being collision-free. Conventional path planning methods either assume that the robot is under no constraint or focus on the optimization of the path length. The robot is abstracted into a single point. The cost of moving from one point a neighboring point is constant as a step, so the efficiency of the path is determined by the number of steps.

In practice, however, the constraint usually carries a high cost for safety reasons. In the case of 3D environment, a valid path needs to specify an accessible orientation to avoid collisions, assuming that the accessibility of the orientations is given as inputs. Special operations are demanded to construct a valid path, which is called 5-axis path (3-axis is the coordinates of the point in the environment in 3D space and 2-axis is the spherical coordinates of the robot orientations). If the specified orientations between two neighboring points are different but accessible on the two points, a special operation is called reorientation to guarantee the continuity of the path. If the specified orientations are different but

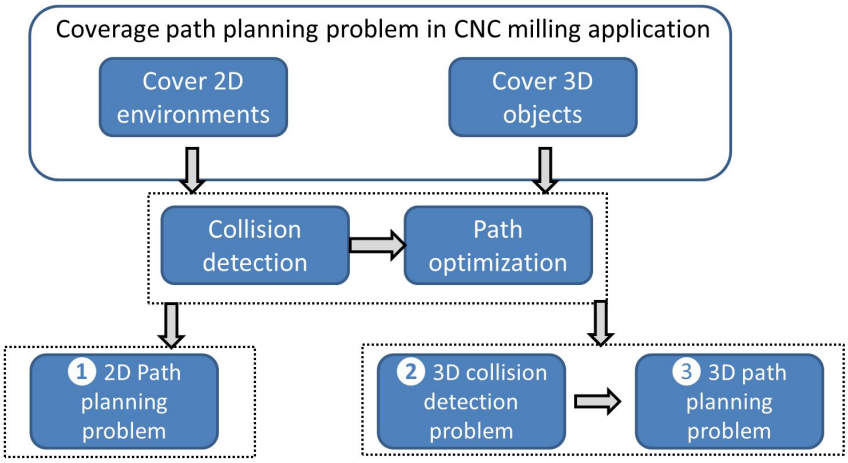


Figure 1.6: CPP problem is divided into three sub-problems.

inaccessible on one point, another special operation is to move the robot far away from the environment to a safe place, and then go back to the next point in the environment. Compared with the common operation that moving between two points on the environment using one accessible orientation, the special operations have a higher cost consuming more milling time. In short, ignoring the constraint would either produce an invalid path or result in a path with a higher cost.

1.5 Decomposing the problem

Figure 1.6 shows how we decompose our CPP problem into more basic building blocks, reepresented as layers from top-to-bottom.

In the first layer from top to down, we categorize our CPP problem into two types: 2D coverage path planning (2D CPP) and 3D object coverage path planning (3D CPP), where 2D CPP means that the target environment is in 2D and the robot stays in a 2D plane, and 3D CPP means the target environment is a 3D object and the robot covers the surface of the 3D object with collision-free orientations.

In the second layer of Figure 1.6, we have two steps for the two types of environments. To ensure that the generated path is valid, the first step is to do the collision detection to calculate the accessibility for all given orientations. The second step is to construct a path

that covers all free space and optimize the efficiency of the path.

The third layer has three problems that we discuss in detail in this thesis. Theoretically, both 2D environments and 3D environments need to go through the two steps in the second layer. However, avoiding collision is easy for the 2D environment, because we just need to check if the path intersects with the obstacle area in the 2d environment, which is given as an input. Chapter 2 describes how to bypass the obstacles to form a collision-free path. Thus, the first problem is to discuss the two steps together for the 2D CPP, which focus more on how to improve the efficiency of the path covering a 2D environment. The second problem is on the collision detection for the 3D objects. The third problem is on how to optimize the path efficiency for 3D CPP concerning the constraint of being collision-free. The three problems correspond to the three challenges mentioned above.

1.6 Contributions

This dissertation makes four primary contributions:

- For the 2D CPP, we propose a corner model used to decompose an arbitrary environment into cells. Different from the graph model, our model does not have the restriction of going from one vertex to another through an edge. It has the advantage of specifying the entrance and exit to a cell. It ends up yielding better paths than algorithms for graph models that lack entry/exit abstractions.

- For the 2D CPP, a deep reinforcement learning approach is employed to adapt various metrics from application as a more generic and modular framework for the path optimization. Different from prior approaches that employ hard-coded heuristics or other application-level specifics, our approach is easy to be extended to other scenarios or a new application.

- In the context of collision detection, we present an abstraction, called ICA. It significantly simplifies the operations required for collision test between a voxel and a

cylinder, by converting 3D operations to 2D operation. Different from conventional way of simplification, our method preserves accuracy without any approximations.

- To further accelerate CD on 3D object, we design a new parallel CD algorithm based on our ICA abstraction. It achieves a better performance through an efficient parallelization, load-balancing and optimizations of corner cases. Our evaluation shows that 99% of CD tests can be pruned.

- For the 3D CPP, we propose a new algorithm called max orientation coverage. It concerns a scenario under the constraint of avoiding collisions. It can improve path efficiency with respect to both the path length cost and the cost of dealing with the constraints. We evaluate our approach through extensive simulation studies on four CAD benchmarks against a state-of-the-art baseline.

1.7 Scope and Outline

Chapter 2 introduces a new approach of covering a 2D environment using deep reinforcement learning. Chapter 3 presents a new way of parallel collision detection. Chapter 4 proposes a new method that construct a collision-free path covering a 3D object. The three chapters correspond to the three sub-problems shown in Figure 1.6.

CHAPTER 2

COVERAGE OF A KNOWN 2D ENVIRONMENT

In this chapter, we present adaptive deep path, a new method that covers an arbitrary 2D environment and optimize the path efficiency supporting various metrics (e.g.. the path length and the number of turns) and various motion strategies, such as moving horizontally and moving vertically. Our new method enables a simple specification of CPP problem instances, rather than ”hard-coded” heuristics.

2.1 INTRODUCTION

The problem of 2D *coverage path planning* has a variety of applications, such as robotic vacuum cleaners for floor sweeping [2], 3D printing [9, 10, 11], agriculture [12], search and rescue [13], and aerial coverage [14, 15], among others. Given an environment consisting of free space and obstacles (i.e., occupied space), the goal is to compute a path that (a) sweeps all of the free space, thereby attaining *complete coverage*, (b) avoids obstacles, and (c) minimizes sweeping time. The general problem is NP-hard, and as such, demands approximate and heuristic algorithms [28].

Figure 2.1 is a simple example to illustrate some of the key issues that arise for the hypothetical task of a robot sweeping an indoor environment. Consider a robot that has finished sweeping the area on the left (the “covered region”) and stops at the bottom. For simplicity, suppose that the robot must now make a local decision in considering the cost of how to arrive at the next region.[1] Which path is better? Typically, there are two possible metrics to evaluate path quality: the path length [24, 15, 14, 35] and the number of turns on the path [36]. Approaches desiring short paths would prefer the first path (red), whereas those wanting fewer turns would prefer the second one (blue). However, it is hard to know

[1] “Local” is in contrast to solving the global optimization that considers all possible future moves.

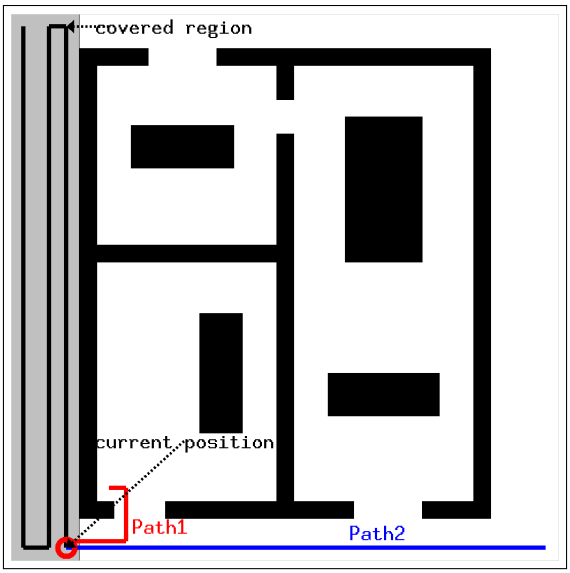


Figure 2.1: An example of a typical local choice during coverage path planning. A hypothetical robot has finished sweeping the free space on the left side (grey) and now arrives at the bottom ("current position"). It considers two choices, Path 1, which is shorter (red line), and Path 2, which has fewer turns (blue line). Which is better depends on the application.

which one will lead to a globally optimal path because doing so depends on many factors, like the speed of the robot, the motion trajectories, and the size of the robot.

Indeed, for any given instance of the coverage path planning problem, there are a variety of possible *configurations*. By configuration, we mean that a specific instance of the problem may involve application-driven variations or constraints. The choice of metric (e.g., path length vs. number of turns) is one example of a configuration option. For another, consider that a vacuuming robot usually supports diverse motion strategies [2], such as "random walk" and "wall-following." Thus, the path in a problem instance for this scenario should constrain the robot's motion accordingly. In 3D printing, a robot may select from multiple machining tool sizes and employ various policies of motion [37].

Among several examples of prior art for coverage path planning, one finds a variety of problem configurations [9, 12, 13, 14, 15]. The challenge is that these configuration options are typically "hard-coded" into the solution algorithm. That is, a proposed algorithm will consider some configuration, but adapting the algorithm to solve instances for other configurations may be tedious or difficult.

Our work concerns a general and unifying framework for solving instances of the coverage path planning problem under a variety of configurations. There are two key ideas.

The first idea is a new abstraction, which we call the *corner model* (Section 2.4.1). Earlier approaches divide the free space into regions, so that one may separately consider how to construct a path between regions (a hard problem, but now of smaller size) from how to construct the subpath inside a region (a simpler and cheaper problem). However, the prior *graph model* abstraction (Section 2.2.4) does not consider that one may choose to enter or exit a region at various points; by contrast, our corner model does so. Representing free space more accurately yields better paths.

The second idea is a more general and modular *framework*, based on *deep reinforcement learning*, which unifies the specification and selection of the problem configuration. That is, rather than an algorithm designer hard-coding the approximation algorithm or its heuristics for a specific configuration, our approach includes many configuration options, is extensible to new options, and can automatically *learn* how to select options without having to input *a priori* application-specific knowledge. For example, rather than hard-code an algorithm to reduce path length or number of turns, our framework can optimize either (or some combination) without having to redesign the overall solution method. Or, instead of assuming or tuning an algorithm for a particular motion strategy, our method can include any (or all) motion strategies as options and learn which is best in a particular scenario. Thus, posing coverage path problem instances becomes simpler.

Our overall contribution for 2D coverage path problems is the combined corner model and deep reinforcement learning framework, which together we call *AD Path*. The method is not without limitations, the most significant being the time to construct a path and the restriction to offline (rather than online) planning. Nevertheless, by unifying and modularizing methods, AD Path offers a flexible platform for addressing new coverage path planning scenarios while reducing the need for application-specific customization.

Additionally, we conduct several experiments to help validate and evaluate AD Path.

In particular, we test AD Path on four environments against a previously proposed ECC algorithm [24, 15, 14, 35]. We observe that our AD Path algorithm reduces total path length by 21.8 % and reduces the total number of turns by 38.6 % on average compared to ECC. With a different configuration and a small robot size, our algorithm produces a path that only has about 2.6 % length cost of connecting sweeping regions, which is very close to the best-known solution.

2.2 Background

In this section, we describe our problem statement, discuss possible ways to solve the problem. These ways usually have three stages: cellular decomposition, model abstraction and approximation algorithm.

2.2.1 Problem statement

Here is a statement of the overall path planning problem we wish to consider. First, the inputs are (a) an arbitrary environment including both obstacle space and free space, represented by pixels; (b) a single robot, represented by a circle, with its diameter equal to the robot size; (c) requirements or constraints on the robot motion; (d) the target cost metric, such as the path length or the number of turns. The output path must have the properties mentioned in the previous section.

2.2.2 A family of candidate solution methods

To solve this path planning problem, we consider a family of methods shown in Figure 2.2. The methods may be described by three stages, where each method corresponds to a particular choice of techniques from among the stages. The first stage uses *cellular decomposition* techniques to divide the arbitrary environment into a collection of coarse-grained cells, where each cell has no obstacles (Section 2.2.3). Doing so converts the problem of how to cover free space into one of covering all cells, where covering each cell be-

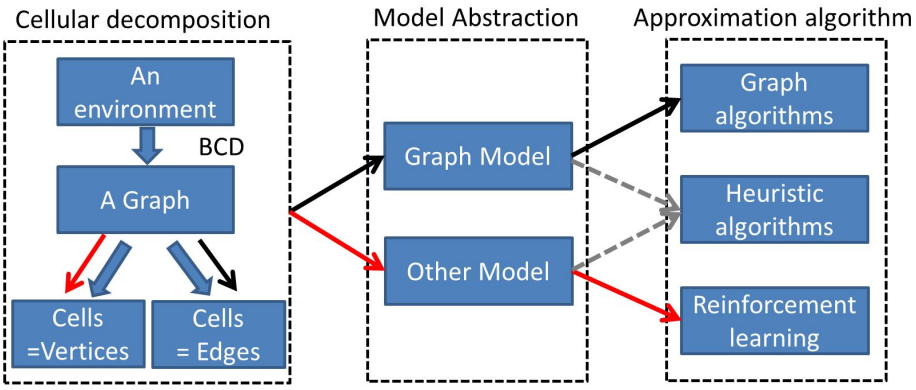


Figure 2.2: We consider a family of "three-stage" methods for the path planning problem. Different methods make different choices of techniques across the stages. Here, the choices underlying the ECC baseline method correspond to the black arrows; our AD Path, the red arrows.

comes a separate (and presumably simpler) subproblem. This approach can significantly reduce overall algorithmic complexity. The second stage builds an abstract model of the problem requirements and the metrics so the overall problem is computationally tractable (Section 2.2.4). The third stage constructs a path using some approximate and heuristic algorithm (Section 2.2.5).

2.2.3 BCD

Cellular decomposition is a technique that divides the free space into a collection of non-overlapping cells [38, 39, 24, 15]. The idea is that each cell is algorithmically "easy" to cover, since it consists only of free space, and thus the overall path planning problem is reduced to covering the cells. This approach should have lower complexity than covering the original space since there can be many fewer cells than points in the original space.

One type of Morse decomposition is BCD [24] that has been widely used in many applications. The following terms are used to describe how BCD works: *slice*, *cell*, *sweep direction* and *critical point*, which are commonly used by other methods as well. A slice is a sweeping line. A cell is an area that the agent can sweep following the slices in vertical back and forth motions. Sweep direction refers to the direction in which the slice is

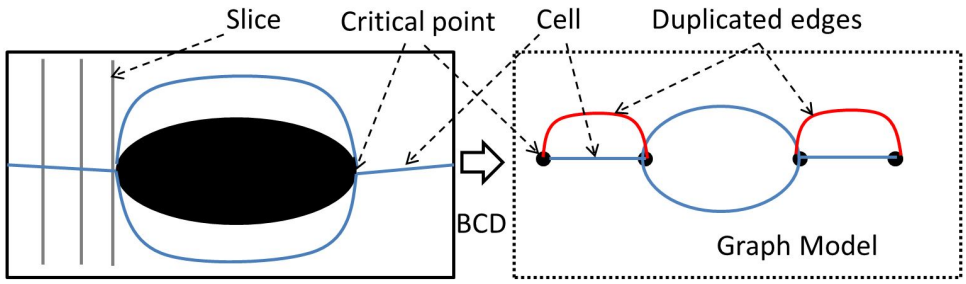


Figure 2.3: An example of applying BCD to convert an environment into a Reeb graph.

swept. A critical point represents a point on an obstacles that causes a change in the slice connectivity.

On the left side of Figure 2.3 illustrates an example of how BCD decomposes an surface. The vertical lines are the sweeping lines denoted as the slice. The connectivity of a slice only changes at the critical points on obstacles. A cell can be formed by connecting two neighboring critical points, such that inside each cell, no obstacle breaks the connectivity of any slices. Thus, the bits in each cell can be fully swept without a single revisit, using a back and forth motion.

The connectivity between neighboring cells are represented by a graph structure, named Reeb graph. The corresponding graph of the surface is given in the right side of Figure 2.3, where the nodes are the critical points and each edge denote a cell between two neighboring critical points. Note that all concave critical points are connected to exactly one cell, i.e., a node of degree one in the Reeb graph. Similarly, all convex critical points are connected to exactly three cells, i.e. a node of degree three in the Reeb graph.

BCD simplifies our optimization problem substantially in two ways: a boustrophedon motion can sweep a cell without causing any revisits, and a Reeb graph is used to represent the connection between cells. Following the connection, we can easily move from one cell to another so we can fully cover all free cells. In other words, a traversal of the nodes on the graph corresponds to an ordered path of visiting cells in the environment.

2.2.4 Model abstraction

The most common abstraction of coverage path planning problems is the *graph model*. It has two representative structures: vertices corresponding to cells or edges corresponding to cells. The Reeb graph uses the latter [38, 23, 24, 15], where visiting an edge corresponds to covering a cell and arriving at the critical point corresponds to a transition to another cell. The graph model always assumes that the path must follow the edges or the vertices in the graph, and also assumes there is a low cost moving from one cell to another passing through the critical point. Figure 2.3 is an example of cells represented by edges.

2.2.5 Approximation algorithms for path construction

At this stage, a path planner need only construct an order of covering the cells. This task is related to the variant problems as discussed in Section 1.2. Based on the graph model, many graph algorithms have been used to construct a path to guide robot exploration [40], mapping [41, 42], and coverage [24]. One of the commonly used algorithms is called ECC [24, 15, 14, 35], which is widely regarded as the state-of-the-art. It applies the Chinese Postman Problem algorithm [43]

Simulated annealing is also applicable [44] for routing problems and TSP [45, 46, 47], among others. It avoids local optima by judicious use of randomness.

Lastly, one may use a *reinforcement learning algorithm* to solve, approximately, a given optimization problem [48, 49, 50, 51]. The design of good heuristics or approximation algorithms for NP-hard optimization problems often requires significant specialized knowledge and trial-and-error. With the help of deep neural networks, reinforcement learning provides more opportunities to automatically learn heuristics that exploit the structure of the optimization problems. This method is the principle one underlying our approach.

2.3 Baseline algorithm

This section presents our baseline algorithm that employs BCD, graph model and Chinese postman problem algorithm. Lastly we conduct an initial study of the baseline algorithm to motivate our work.

Figure 2.4 presents the workflow of efficient complete coverage (ECC) algorithm [24, 15, 14, 35]. The two main steps are BCD and Chinese postman problem: BCD is to decompose a bitmap into Reeb graph, with each edge representing a cell, and Chinese postman problem is to find an Euler tour as the traversal order of cells. The output path contains two parts: how to cover a single cell and how to order the coverage of the cells.

2.3.1 Chinese Postman Problem algorithm

Many graph algorithms are used to guide exploration, mapping and coverage in the past. Based on the graph we obtain from BCD, we target the algorithms that can cover all edges in the graph with a small number of traversal times. A classic and relevant problem is the Chinese Postman Problem : find a shortest tour that traverses every edge at least once, which is proved to be NP-hard. The target of finding a shortest tour is consistent with our goal of reducing the time of visiting edges.

Another problem is on Euler tour, which is a circuit that covers every edge in a graph exactly once. If an Euler tour can be found in the graph with exactly once coverage, the requirement of traversing every edge at least once can be met. A Euler tour is a solution to the Chinese postman problem, since it guarantees exactly once edge-covering.

An prerequisite of the existence of an Euler tour in a graph is that all nodes should have even degree, which is demonstrated by Euler, where such graphs are called Eulerian Graphs. Even degree is mandatory because passing a node in a graph has to go in and out, corresponding to even times of traversing. If a graph is Eulerian graph, any Euler tour is a solution to the CPP. If a graph is not Eulerian graph, additional step is required.

2.3.2 Edge duplication

ECC uses a heuristic that duplicates some edges to let all vertices have even degree, and can thus guarantee *exactly one* visit of each edge to further reduce the cost of the traversal. In the case that the graph is not Eulerian, different strategies can be applied to determine which edges to be duplicated. Note that all of the duplicated edges are part of the cost of the Euler tour, based on the definition of Euler tour. The challenging is to how to choose the duplicated edges.

Linear programming is used to identify which edges to be duplicated. Mannadiar [52] proposes to apply integer linear programming to minimize the total cost of all possible duplicated edges for the first stage as the following: The target is to minimize:

$$z = \sum_{e \in E}^{\infty} (c_e x_e)$$

under the following constraints:

$$\sum_{e \in E} (a_{ne} x_e) - 2w_n = k_n, \forall n \in V;$$

$$x_e \in Z^+, \forall e \in E;$$

$$w_n \in Z^+, \forall n \in V;$$

where x_e is the total number of the duplicated edge of the edge e in the solution, c_e is the cost of the edge e. $a_{ne} = 1$ meas that the node n is connected to the edge e, otherwise $a_{ne} = 0$. $\sum_{e \in E}(a_{ne} x_e)$ is the number of all the edges connected to the node $n \in V$. w_n is an integer variable that forces $\sum_{e \in E}(a_{ne} x_e)$ to be odd for nodes with odd degree, and even for nodes with even degree. k_n is 1 for nodes with odd degree, and 0 otherwise. As shown in Figure 2.3, the right side is the updated graph after duplicating two edges. After the duplication, all nodes in the graph have even degree, so that an Euler tour can be found

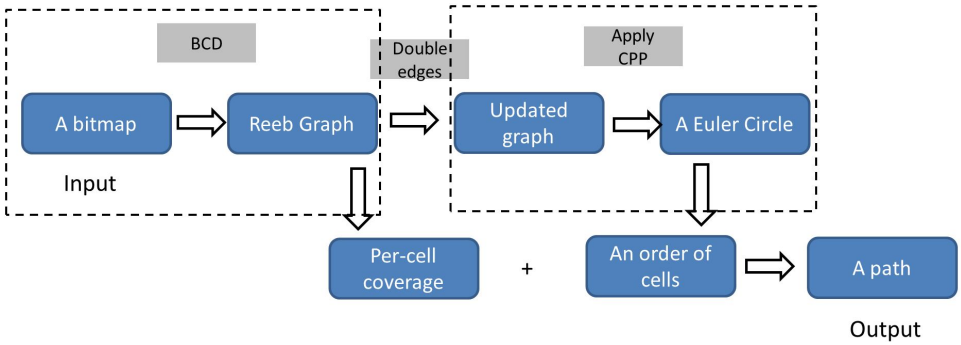


Figure 2.4: The workflow of ECC algorithm including BCD and CPP.

as the solution to the Chinese postman problem.

2.3.3 Avoid repeat coverage

Due to the edge duplication, the cells that have a duplicated edge are covered twice. To avoid this, cells corresponding to the duplicated edges are divided into non-overlapping cells. After this division, each cell only needs to be covered once to achieve a full coverage. However, this demands another heuristic on how to divide the duplicated cells to improve ultimate path efficiency. The heuristic taken by ECC is to divide wide and large cells instead of narrow and small cells. This is motivated by an assumption that narrow and small cells are likely to have a higher cost of connecting the divided cells than the wide and large cells. The edge cost of a cell is defined as c_e in the following equation.

$$c_e = \frac{cell\_width^2}{cell\_area} = \frac{cell\_width}{cell\_height}$$

In short, the solution to the Chinese postman problem consists of two stages: first to duplicate a set of edges to ensure even degree on all nodes in the graph, and then to construct a Euler tour on the duplicated graph. Two strengths of ECC algorithm are:

- It works on cells rather than bit and thus reduce the complexity significantly.
- The per-cell coverage is simple and efficient without costing a single revisit.

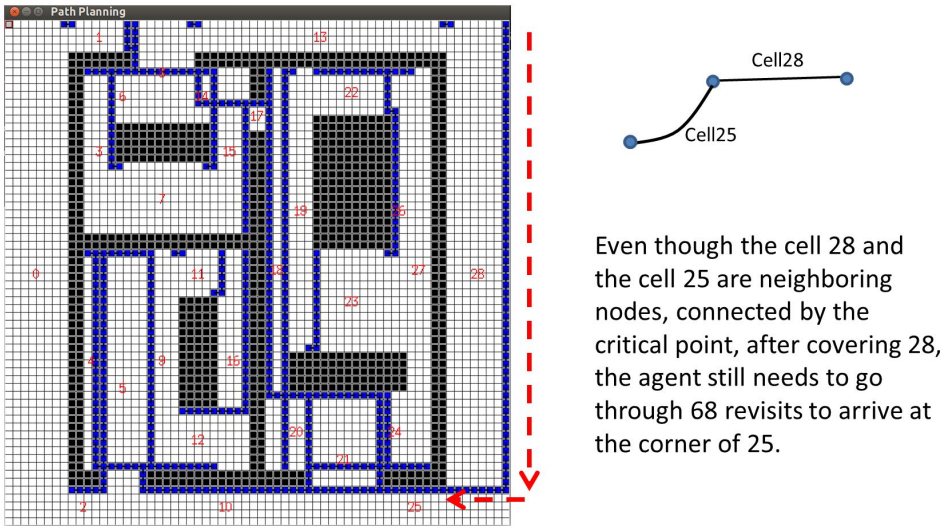


Figure 2.5: An example of applying ECC algorithm.

However, a weakness is that it ignores the cost of the connection between cells. After covering a cell, the agent needs to reposition itself to a corner of next cell.

Figure 2.5 gives an example of applying ECC algorithm. In the left side, the black area represents obstacles while the white area represents the free bits that needs to be covered. The number shown in the white area is the cell Id through BCD. The connected bits between cells in the traversal order generated from the Chinese postman problem are denoted with the blue path.

A specific connection between the cell 28 and the cell 25 is shown in the right-hand side of Reeb graph. The two edges representing the two cells are connected on the critical point in the middle. Following the order output from the solution to the Chinese postman problem, the agent goes to the closet corner of the cell 25 after covering the cell 28, with a cost of 68 revisits marked by the red dashed line. The key to this inefficiency lies in the cost function of construction of Euler tour. ECC algorithm assumes that two cells in the graph are "close", such as the cell 25 and the cell 28 connected by one critical point, indicating that the cost of going from one to another is small. However, this is not true. The distance between edges in the graph does not necessarily reflect the actual cost of connecting cells.

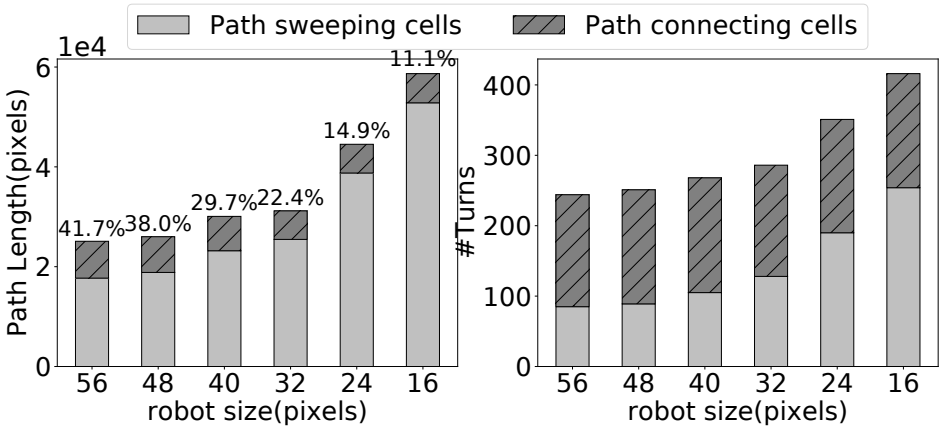


Figure 2.6: Cost analysis of the path generated by ECC algorithm on the 1024×1024 pixels described in Figure 2.1. Most commercial Roomba robots have a diameter of 13.5 inch. If the environment is mapped to be $1024 \times 1024\,\text{cm}^2$, the Roomba robot has a size of 34.3 pixels.

2.3.4 Initial Experimental study of ECC algorithm

To gain some intuition for how the ECC performs, we apply it to the environment shown in Figure 2.1 with various robot sizes. The results appear in Figure 2.6, assessing both the path length and the number of turns. There are two types of paths: the path sweeping the free cells and the path connecting cells to form a complete path. As the robot size decreases, the path length gradually increases (since the robot is smaller). Correspondingly, the number of turns also grows. Regarding the path length, the sweeping path has a constant cost, but the connecting path has a large potential to be reduced, especially for larger robots. Regarding the number of turns, both paths can be optimized to reduce the cost as we will show in Section 2.5, especially when the robot size is small.

2.3.5 Summary of the baseline algorithm

Our baseline algorithm is the state-of-the-art, called ECC [24, 15, 14, 35] as an improvement over the graph model, using Chinese Postman Problem algorithm. To generate an efficient path through this algorithm, three steps are prerequisite and heuristic. First, the environment needs to be formalized as a graph, where edges correspond to cells. Second,

some edges have to be duplicated. Third, the duplicated cells will be partitioned to avoid repeat coverage.

2.4 Our Approach: AD Path

The design of AD Path begins with BCD, as illustrated in the red arrows of Figure 2.2. It adds a new abstract, the *corner model* (Section 2.4.1), and a general unified framework based on deep reinforcement learning (Section 2.4.3).

2.4.1 Corner Model

In our corner model, we explicitly model the existence of four corners for each cell, as potential positions to enter and exit the cell. See Figure 2.7. Regarding the connection between cells, each cell can go to any other cell. Sweeping a single cell has two steps: covering all free elements of a cell and moving to another cell. This process iterates until all cells are covered. Correspondingly, we define two types of cost: (a) ***the cost of sweeping a cell***: $\mathrm{cost\_in\_cell}(\mathrm{entry}, \mathrm{exit})$, where $\mathrm{entry}, \mathrm{exit}$ are the enter corner and exit corner in the cell respectively; (b) ***the cost of connecting two cells***: $\mathrm{dist}(i, j)$, where i, j are the corner IDs indexed by cell ID. The cost has two parts: the path length and the number of turns. Since the length of the path covering all free space in the environment is constant, the cost on the path length in $\mathrm{cost\_in\_cell}$ is defined to be the number of the already visited pixels as *revisits*, not including the path length covering free pixels. Since the path length of the path connecting cells does not cover free space, $\mathrm{dist}(i, j)$ is the actual length of the path connecting any two corners. The cost on the number of turns counts the turns on the whole path.

Figure 2.7 illustrates a case in which a robot enters at corner 0 and exits at all 4 corners, where the path of the revisit is marked with red lines. Here, we assume that the robot mainly sweeps the free area in vertical motion following the slices in BCD. The robot can enter and exit at any 4 corners for a single cell, which has 16 possibilities in total. We can

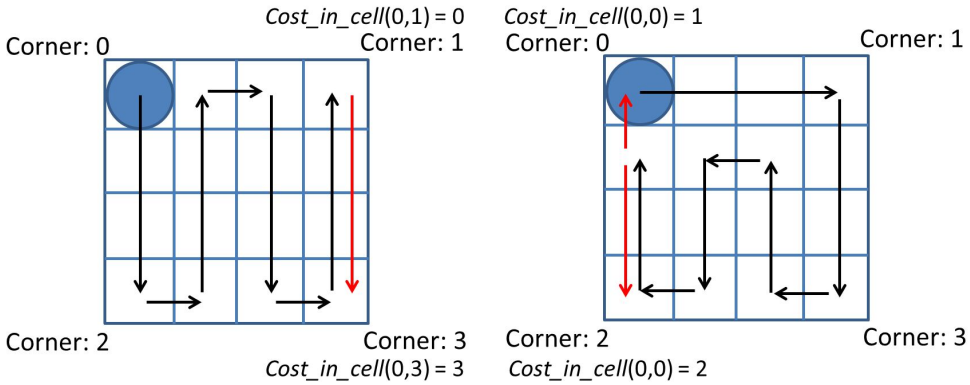


Figure 2.7: An example of $Cost\_in\_node$ that calculates the cost of sweeping a cell. We only consider the length of the path of covering already visited pixels as the number of revisits, represented by the red lines.

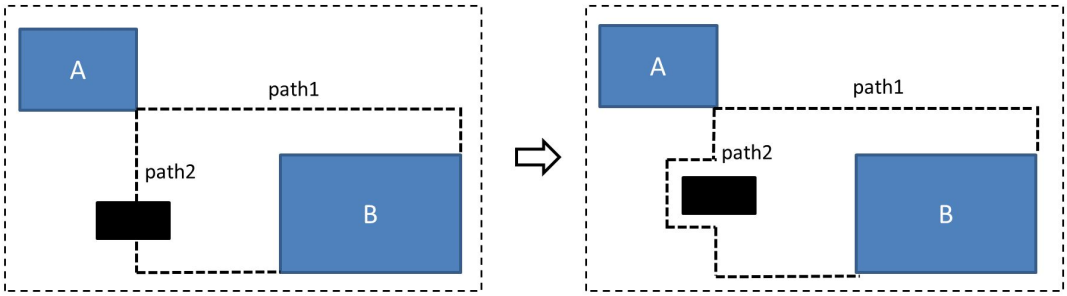


Figure 2.8: Illustration of calculating the cost of connecting two corners between two cells denoted by $\mathrm{dist}(i, j)$. The path "path1" avoids obstacles whereas "path2" intersects an obstacle. Thus, "path2" is converted to a bypassed.

see that where to enter and exit the cell is an important factor that influences the cost of sweeping a cell.

Different from the graph model that has to follow edges or vertices, our corner model supports any connections between two cells as long as there exists a valid path that avoids obstacles. Given two corners $p_1(x_1, y_1), p_2(x_2, y_2)$ from two cells A and B, we need to calculate the distance as the cost. Figure 2.8 illustrates an example of calculating the cost of connecting the two cells. Bypassing does not ensure that there must exist a valid path between any two points, which depends on the shapes of the obstacles. For those points where bypassing cannot find a valid path, we assign a maximum value (conceptually, ∞) as the cost.

Another difference between the two models is that our corner model gains a small

cost without any heuristics or approximations. We apply the Floyd-Warshall algorithm to all corners for the purpose of reducing the cost and reflecting the actual path in the environment by calculating the "shortest distance" for the cost of revisits and the number of turns. Although Floyd-Warshall is commonly used to find the shortest distance in a graph, our result does not guarantee that the distance is the "shortest" or incurs the minimal number of revisits, which is unnecessary in this case. Our aim is only to find a valid path that reflects the actual environment. The shorter the path is, the better our result will become.

2.4.2 Apply a new configuration

In the previous section, we made two assumptions about the configuration of how the robot moves. One assumption is that the robot needs to mainly follow the vertical motion when defining *the cost of sweeping a cell*. Another assumption is that the robot needs to move axis-aligned when defining *the cost of connecting cells*. This configuration implies that we aim to reduce both the path length and the number of turns.

We claim that our corner model is an adaptive framework that can optimize the cost of the path for application-specific configurations. Consider a new configuration that allows *both* vertical and horizontal motions to sweep the free space in the cell. With this configuration, the path length can be reduced further at the cost of increasing the number of turns. Figure 2.9 illustrates an example of how this configuration can further reduce the cost of sweeping cells. Similarly, if the application requires it, we can directly move from one position to another in a straight line by filling in $\mathrm{dist}(i, j)$ as the cost of connecting cells. We will evaluate the influences of these scenarios in our benchmark (Section 2.5.3). Our model supports other configurations as well, such as rotating the environment by an arbitrary angle, which might result in a more efficient path.

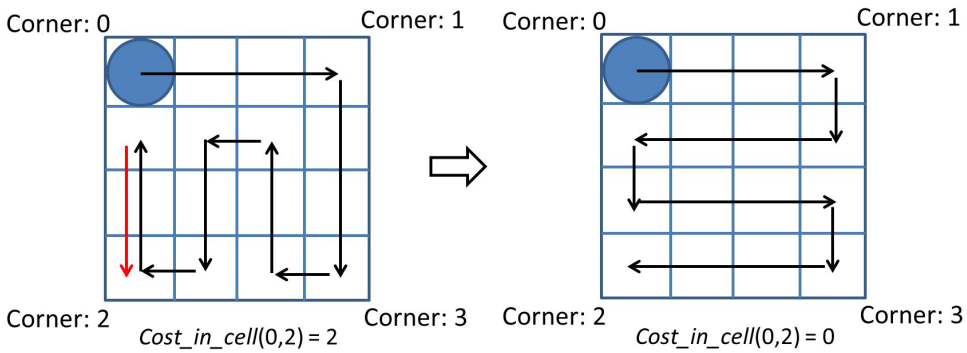


Figure 2.9: Allowing the robot to move both vertically and horizontally. Under this configuration, the length of the path sweeping cell, denoted by $\mathrm{cost\_in\_node}$, can be further reduced.

2.4.3 Deep reinforcement learning

Our AD Path algorithm considers the overall problem of constructing a full path of all cells as a combinatorial optimization problem. It varies the order in which cells are visited and incrementally constructs a path by adding cells through deep reinforcement learning. In our framework, the states, actions, and rewards are defined as follows.

- Each uncovered cell v is a potential action. If the action is taken, it is appended to the path.
- State S is a sequence of actions (cells) that have already been taken at the current step. It also corresponds to an ordered path of cells. A terminal state ($\hat{S}$) is that all cells have been chosen.
- The transition is a process that the robot takes an action v when facing a state S, and then arrives at a new state S'. Applying an action v corresponds to adding a cell to the current partial solution S. The transition is deterministic here.
- The reward for taking action v in state S is a function $r(S, v)$, which may be defined in terms of the cost $C(S, G)$ of being in state S given the graph G and the cumulative state $h(S)$, which represents the maintenance function corresponding to the sequence of all states up to (but excluding) S. (We define $h(S)$ more formally in Section 2.4.3.)

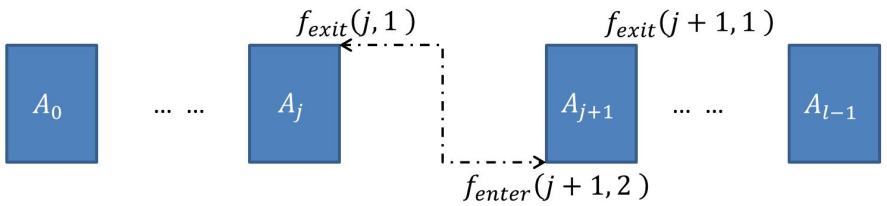


Figure 2.10: A path starts at cell A_0 and ends at cell A_{l-1}. Each cell has two types of cost functions: f_{enter} and f_{exit}.

A transition results in collecting a reward $r(S, v) = C(h(S'), G) - C(h(S), G)$, where $C(h(\emptyset), G) = 0$. As such, the accumulative reward (R) of a terminal state ($\hat{S}$) coincides exactly with the cost function of the $\hat{S}$.

- Policy $\pi(v|S)$ is a set of deterministic transitions that greedily maximizes the rewards or minimize the costs, which is $\pi(v|S) := \arg\max_{v' \in S} \hat{Q}(h(s), v')$.

Evaluation function for an arbitrary path

We need to define the evaluation function $Q(h(S), v)$ that evaluates the value of a possible action v on state S in Q-learning [53, 54]. In our case this function needs to evaluate a path that has an arbitrary number of cells with an arbitrary order.

We define two types of functions: f_{enter} and f_{exit}. The function $f_{\text{enter}}(j+1, c_{j+1})$ denotes the cost of the path starting at cell A_0 ending at the corner of cell A_{j+1}, where the entrance corner is c_{j+1} and the former part (from A_0 to the exit of A_j) remains unchanged. The function $f_{\text{exit}}(j+1, c_{j+1})$ denotes the cost of the path starting at cell A_0 ending at the corner of cell A_{j+1}, where the exit corner is c_{j+1} and the former part (from A_0 to the entrance of A_{j+1}) remains unchanged. Based on this definition,

$$f_{\text{enter}}(j+1, c_{j+1}) = \min(f_{\text{exit}}(j, c_j) + \text{dist}(c_{j+1}, c_j)) \tag{2.1}$$

$$f_{\text{exit}}(j+1, c_{j+1}) = \min(f_{\text{enter}}(j+1, c'_{j+1}) + \text{cost\_in\_cell}(c_{j+1}, c'_{j+1})), \tag{2.2}$$

where c' denotes another corner. Observe that f_{enter} depends on the f_{exit} of the last cell, which needs to use $\text{cost\_in\_cell}(\text{entry}, \text{exit})$ in the corner model, and f_{exit} depends on f_{enter} of the current cell, which needs to use $\text{dist}(i, j)$ in the corner model. Our algorithm only needs one traversal of the cell list from left to right. The corner that has the minimal cost of f_{enter} and f_{exit} is marked as the actual entrance and the exit. After determining these corners for all cells in the path, the minimal cost over 4 corners of f_{exit} on the last cell is considered as the cost of the given path.

State transition to form a complete path

In a transition, a maintenance function (or helper procedure) $h(S)$ is needed. It maps an ordered list S to a combinatorial structure that satisfies the constraint that the procedure returns a list with the minimal cost. In the case of taking an action v, it produces a new state S' by inserting a cell A_i to an ordered list $(A_0, A_1, ..., A_l)$ as illustrated in Figure 2.10. Given that this list has length l, there are $l + 1$ potential positions for the insertion of A_i corresponding to $l + 1$ possible new states. The helper procedure is to return the new state S' that yields the minimal cost by applying the evaluation function.

Learning algorithm

To achieve a faster learning convergence and a better result, we used a combination of n-step Q-learning [54] and fitted Q-iteration [53] similar to the greedy Q-learning algorithm [49]. The n-step Q-learning helps deal with the issue of *delay rewards*, where the final reward is only received far in the future. In our setting, the cost of sweeping a cell is only revealed after many cells swept. The fitted Q-iteration approach uses *experience*

Figure 2.11: The four tested environments from left to right: Cave; Multi-cells; Rural Quebec; Indoor.

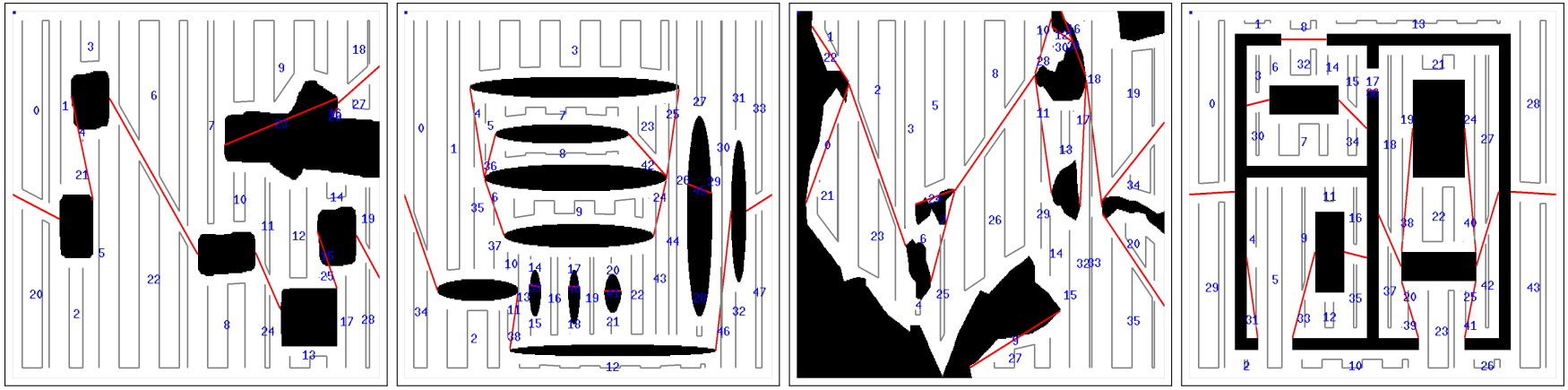


Figure 2.12: Apply BCD to decompose the environment into cells, where the red line is to partition cells for ECC resulting from the edge duplication and the grey lines represent the motion trajectories to sweep each cell.

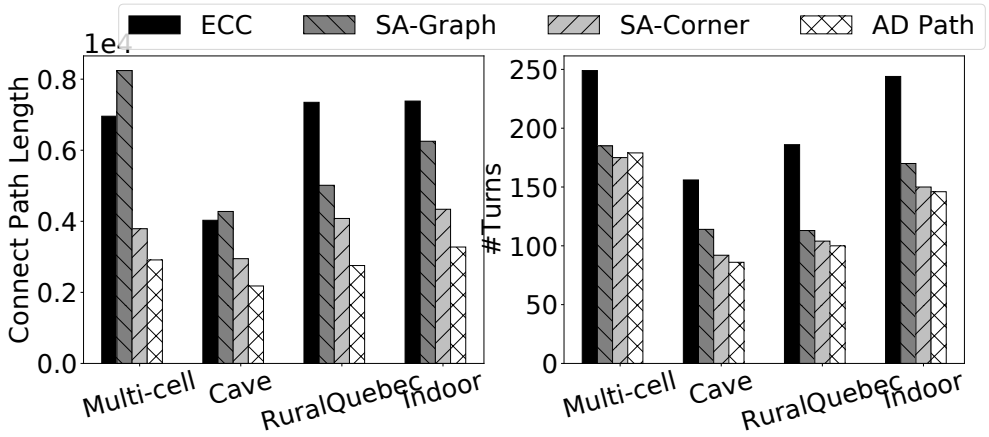


Figure 2.13: Efficiency results of the 4 candidate algorithm on the 4 environments, where the robot size is 56. The path length is only on the path connecting cells, but the number of turns is for the complete path.

replay to update the function approximator with a batch of samples. This also works in our setting by storing many partial paths in memory as the experiences and then apply stochastic gradient decent in batches.

The learning objective is to learn how to decide the order of covering cells based on the layout of the environment. For now, our deep reinforcement learning framework works like an optimization algorithm, focusing on improving the path efficiency for the given environment. Every new environment requires a training process as an optimization step, which takes several minutes. We mentioned before that our framework targets on *offline* path planning scenarios, where the application is not sensitive to the processing time. In our future work, we will train our model on a large number of environments so that the trained model can be reused and work for any environments.

2.5 Experimental Evaluation

The experimental validation mainly answers the following three questions: (a) How do the candidate algorithms perform in different environments, on both the path length and the number of turns? (b) By how much can AD Path improve on ECC, with various robot sizes? (c) What is the influence of new configurations? Can our algorithm adaptively adjust

the cost model as the configuration varies?

Implementation and configuration. The stages shown in Figure 2.2 describe a family of potential methods. One member of this family is the ECC algorithm, which is our baseline. Additionally, we also design two simulated annealing algorithms based on the graph model and our corner model for comparisons: SA-Graph and SA-Corner, representing simulated annealing algorithm using the graph model and our corner model, respectively. Similar to ECC, one heuristic used in the graph model is that robot choose a corner close to the critical point to move from one cell to another [15, 14].

For the learning algorithm and the deep neural network, we reuse the framework of combinatorial optimization on graphs and the parameters of neural network as described in [49, 55]. This framework is based on a representation called graph embedding [56]. The experiments are only run on CPU without using GPU.

Test environment and terms: We use four benchmark environments, which were used by others [35]. These environments appear in Figure 2.11, where black areas indicates obstacles and white areas the space to sweep. Figure 2.12 shows all the cells after applying BCD. The total path consists of two types of paths: the path sweeping cells and the path connecting cells. For brevity, in the following we usee *sweep* and *connect* for the two types of the paths, respectively.

2.5.1 Benchmark

We tested the 4 candidate algorithms on the 4 environments, and in Figure 2.13, report the path length and the number of turns. Since the length of the path sweeping cells is the same, only the length of the connect path is shown. SA-Graph has a shorter path length than ECC on the rural Quebec and the indoor environment, but has a longer path on the multi-cell and the cave environment. The reason is that the optimization of the SA algorithms involve randomness, so the heuristic of choosing a closer corner as the entry position of the next cell is not necessarily a stable or consistent decision. Comparing SA-Corner with

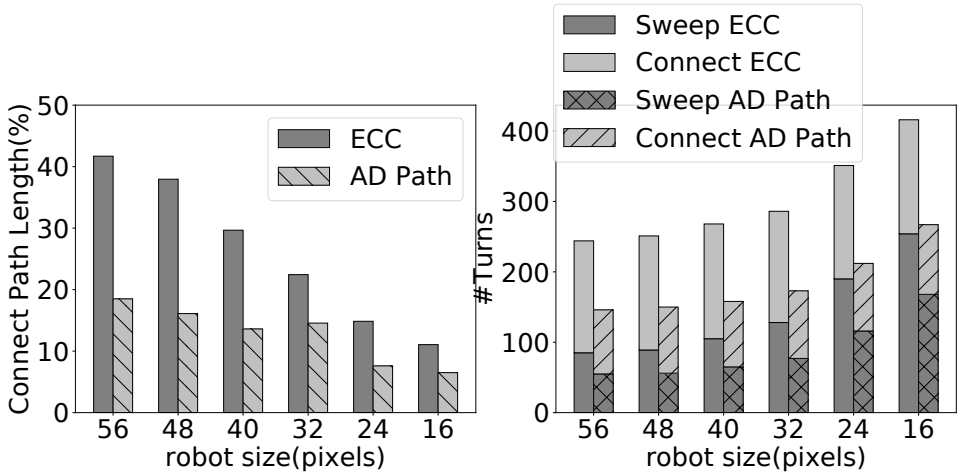


Figure 2.14: Efficiency results of varying the robot size on the indoor environment.The path length is only for the path connecting cells.

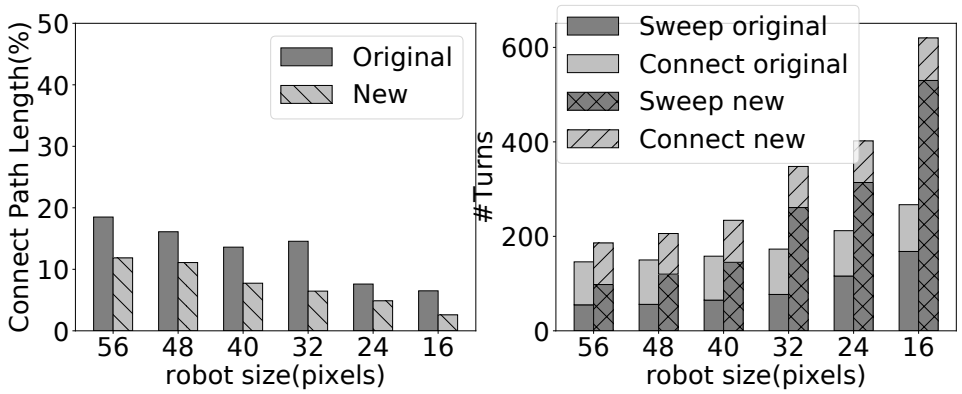
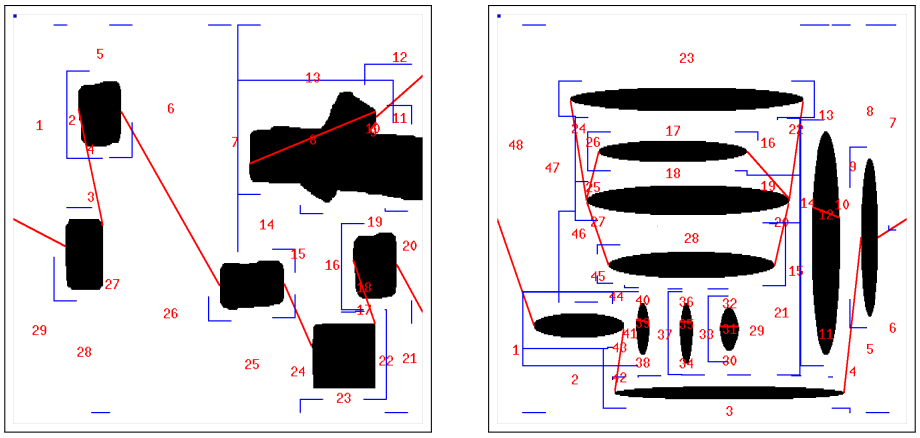


Figure 2.15: Efficiency analysis of our AD Path algorithm under two configurations as described in Section 2.4.2 on the indoor environment.

SA-Graph, we can see that it reduce the path length by 33.6 % and reduce the number of turns by 12 % on average of the 4 environments benefited from our corner model. AD Path obtains 26.6 % shorter path than SA-Corner due to the efficiency of our deep reinforcement learning approach. Compared with ECC, our algorithm reduces the connect path length by 55.5 % and reduces the number of turns by 38.6 % on average. Figure 2.16 and Figure 2.17 show the results of the ECC algorithm and our AD Path algorithm. Due to the training process in deep reinforcement learning, our approach AD Path requires a time of minutes for executions.



2.5.2 Varying robot sizes

The analysis depicted in Figure 2.14 shows how our AD Path algorithm performs with various robot sizes. The robot size is an important cost-factor because it influences not only the number of turns but also the position of entering and exiting cells. On the length of the connect path, our AD Path algorithm produces the path 48.7 % shorter than ECC on average of all the robot sizes; it decreases the number of turns by 39.4 % on average. The number of turns on the path of sweeping cells decreases because ECC has a larger number of cells to cover and has to make more turns, resulting from edge duplication. Also, note that our robot is considered to be a circle. It is possible that some pixels in corners can not be covered because the robot is not small enough to touch the corners. A smaller robot size would need to be specified to sweep the uncovered area.

2.5.3 Apply a new configuration

To further strengthen our efficiency analysis of configurations, we apply a new configuration that the robot can move both vertically and horizontally when sweeping a cell, which actually allows sacrificing the number of turns to reduce the path length. That is, this new configuration gives the robot more options for moving to reduce the path length. Figure 2.15 shows the result of the connect path length and the number of turns under the two configurations. On average of all the robot sizes, with the new configuration, our AD Path

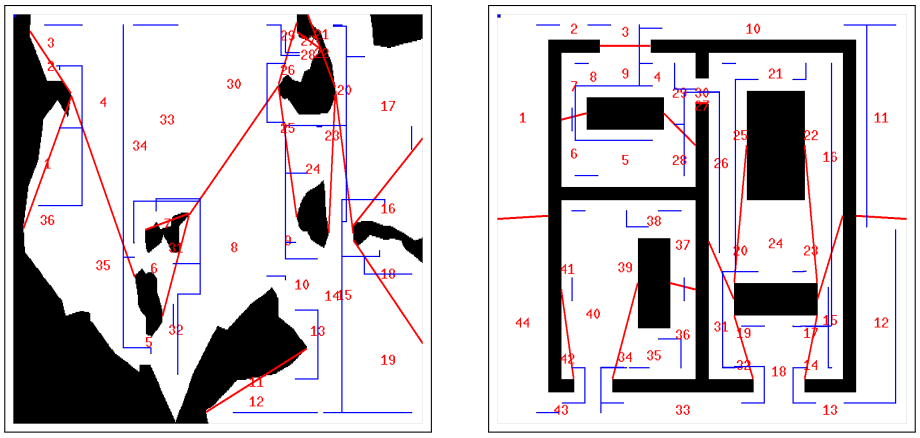


Figure 2.16: The path produced by ECC algorithm with robot size 32, where the red numbers denote the order of sweeping cells and the blue line represents the path connecting cells.

produces the path 43.6 % shorter than the one with the original configuration, at the cost of increasing the number of turns by 72.6 %. Figure 2.18 presents the path results under the new configuration. We can see that on the robot size 16, the connect path produced by our algorithm only has 2.6 % of the sweep path. Therefore, our AD Path algorithm can produce adaptive and efficient paths for various kinds of configurations.

2.6 CONCLUSIONS

By combining a better abstraction (our corner model, Section 2.4.1) and a more modular optimization framework (deep reinforcement learning, Section 2.4.3), our proposed AD Path method enables simpler specifications of coverage path planning problem instances. This capability frees users to explore more options for constructing low-cost paths by removing *a priori* decision-making. There are some limitations, including the cost of search, which prohibits the use of our method in online (as opposed to offline) planning scenarios and, therefore constitutes the most rich opportunity for future work.

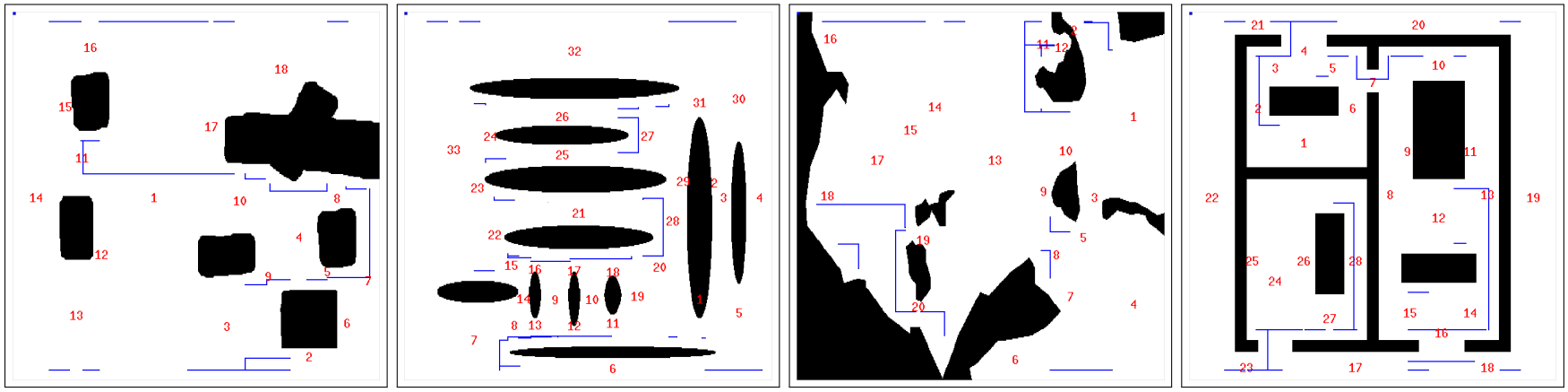


Figure 2.17: The path produced by our AD Path algorithm with robot size 32, where the red numbers denote the order of sweeping cells and the blue line represents the path connecting cells.

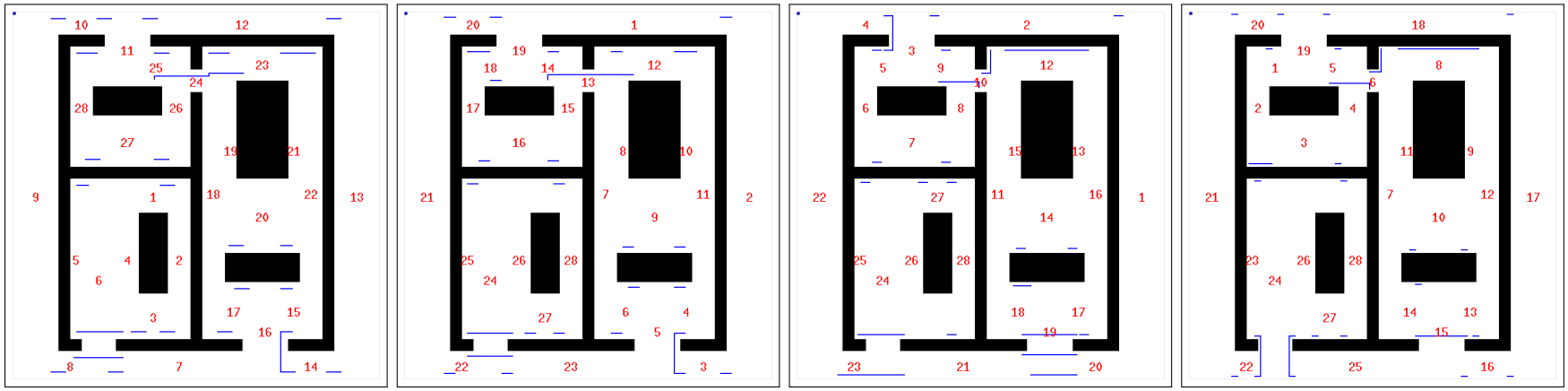


Figure 2.18: The path result of our AD Path algorithm under the new configuration, with various robot sizes from left to right: 40, 32, 24, 16. The blue lines denote the path connecting cells.

CHAPTER 3

FASTER PARALLEL COLLISION DETECTION

In this chapter, we move to the collision detection problem for the CPP of 3D objects. We present an abstraction, called ICA used to simplify the collision tests. We also proposed a parallel approach based on the ICA abstraction, called the aggressive inaccessible cone angle (AICA) method that, empirically, can prune as much as 99% of the intersection tests that would otherwise be required and improve load balance.

3.1 Introduction

We consider a CD problem that arises in the area of *CNC milling* [57], an application in advanced manufacturing. An example appears in Figure 3.1. There is a shape one wishes to cut starting from a block of material, such as the head from an initial cube of plastic (the left of Figure 3.1). The computational task is to construct a path that a cutting tool can make that eventually ends with the target object (e.g., the head), starting from the input (e.g., the block of plastic).

There are several possible CD methods, which are widely used in other settings, like CAD/CAM [10, 58, 59, 9]; computer animation, games, and physical simulations [16];

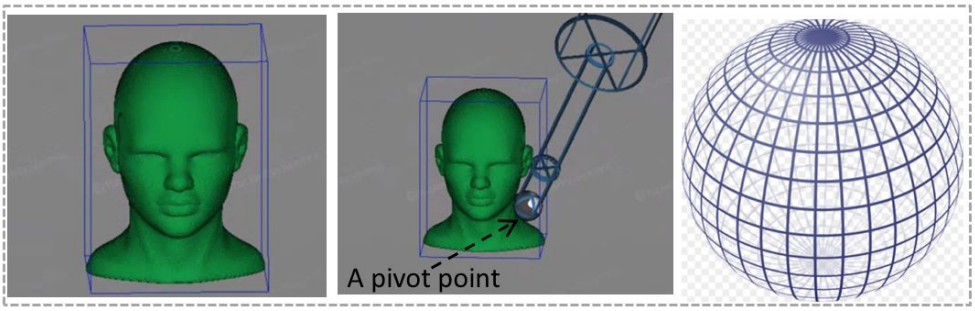


Figure 3.1: Inputs to the collision detection (CD) problem: a head object from CAD benchmark, a tool composed of bounding cylinders and a pivot point at the end of the tool. The orientation of the tool at the pivot is represented by a pair of angles in polar coordinates (φ, γ), where $\varphi \in (0, \pi)$ and $\gamma \in (0, 2\pi)$.

motion planning [17, 18]; and virtual assembly [19]. To improve the speed of CD, prior approaches have combined computer graphics analysis techniques with efficient parallelization. Such techniques include culling to prune redundant computation [60, 61], as well as algorithms that can exploit GPU features like visibility queries in the depth buffer, and frame buffers and fragment shaders [62, 16, 18]. But there are also efficient parallel CD methods for both general-purpose CPUs and GPUs [63, 17, 64, 65].

Underlying most of these approaches are three types of fundamental, computationally intensive operations: decompositions, rotations, and projections. We illustrate them in the bottom of Figure 3.4, in the case where one wishes to check whether a cylinder (i.e., one model of a tool) intersects with a box (i.e., part of the object being milled). Briefly, these operations are a sequence of geometric calculations that transform the input object into other representations, as explained in Section 3.2. These three operations also appear commonly in other types of basic geometric intersection tests, such as sphere-box intersection, box-box intersection, and cylinder-sphere intersection. Such tests are the basis of discrete collision detection (DCD) and continuous collision detection (CCD) algorithms in computer graphics [63, 64, 58, 65, 66, 67].

However, for CD, cylinder-box intersection tests dominate and may be sped up considerably. We do so using a novel abstraction, called the *ICA*, that eliminates a high percentage of the usual cost of CD tests for decompositions, rotations, and projections. We further accelerate this method via a parallel algorithm that we call the *AICA* method. Prior methods to test for the intersection between two objects relied on a general *bounding volume hierarchy (BVH)* and fine-grained volumetric representation of the cutting tool. However, in our application we can replace this representation by a simpler collection of bounding volumes, like the bounding cylinders suggested in Figure 3.1, thereby making the three fundamental operations cheaper. This simplification also suggests a new way to express the computation, yielding a method that has smaller "constant factors" and is easier to load-balance. Our ICA abstraction could be extended into a new primitive and, thereby, applied in other

Figure 3.2: The output of one CD test is an accessibility map (AM). A map at (m, n)-resolution is discretized uniformly into $m \cdot n$ points, with each point denoting some (φ, γ) orientation. Here, the map is shown as a grid whose vertical axis corresponds to φ and whose horizontal axis corresponds to γ. Schematically, a black point indicates a collision between the voxel and the tool when oriented at (φ, γ), and a white point means no collision.

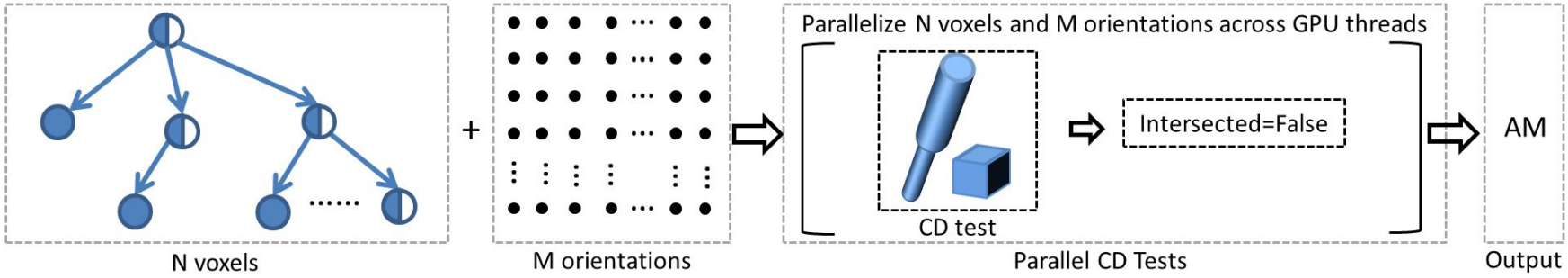


Figure 3.3: Schematic of the baseline parallel scheme to calculate the AM. It begins with the target (left), stored as a adaptive (non-uniform) volumetric octree with N voxels, and the $M = m \cdot n$ discrete orientations of the tool to check for collisions. Each (GPU) thread considers one orientation and executes Algorithm 2, which traverses the octree to see if that orientation causes an intersection with any voxels. (Algorithm 2 does not need to visit all voxels if it detects early in the traversal that no intersection is possible.)

CD contexts that involve rotational operations (Section 3.6).

In brief, the main claimed contributions of this paper are the ICA abstraction, the AICA parallel algorithm, and an empirical validation thereof. [1] The basic ICA abstraction allows us to reason about the object over all orientations in a computationally compact way, thereby reducing the number of operations and checks than prior art. When using an adaptive volumetric octree to store the target object (e.g., the head of Figure 3.1), we observe that as many as 99% of the CD tests can be eliminated on a variety of complex input geometries. We have prototyped our approach in a version of the commercial SculptPrint software package. Our AICA method can be over $23\times$ faster than a baseline approach that uses 3D projection, and nearly $4.8\times$ faster than another novel method we present. In absolute performance, AICA enables the checking of 4096 orientations for an object represented by 27 million voxels in just 18 milliseconds on a recent GPU.

3.2 Background and Motivation

In this section, we begin with a problem statement, the requisite background on spatial data structures involved in this problem, and summarize the baseline algorithms and an initial experimental study as our motivation.

3.2.1 Problem Statement

. The inputs of our CD problem are (a) a 3D object, which is the *target* (e.g., the head); (b) another 3D object, which is the *tool*; (c) a pivot point upon which one end of the tool is fixed; and (d) the set of (discretized) orientations of the tool to consider. The output is an AM that indicates whether or not each orientation leads to a collision between the two objects. Figures 3.1 and 3.2 illustrate these inputs and outputs.

To efficiently detect collisions, we will assume the setup of Figure 3.3. The target

[1] An initial sketch of the ICA appeared earlier in an unrefereed work [57]. However, the previously proposed calculation of ICA is incorrect. In this paper, we first give a correct description and then present a parallel scheme, neither of which appeared before.

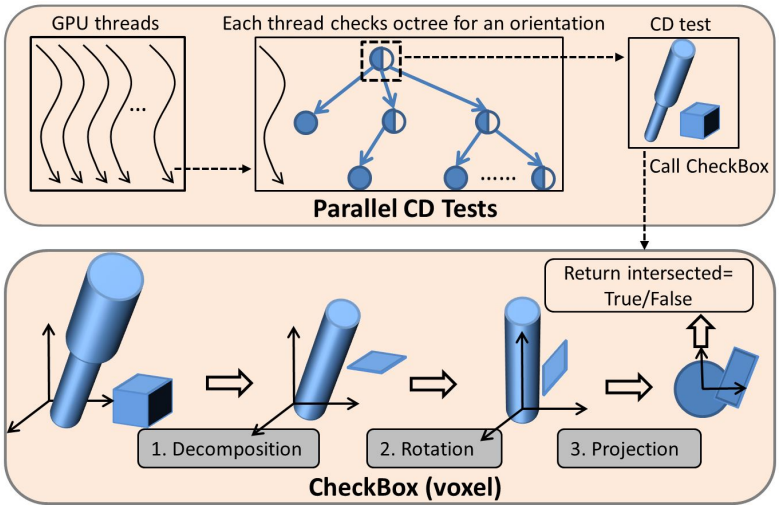


Figure 3.4: Our baseline algorithm that performs the parallel CD tests by calling CHECKBOX, which involves the three steps that cost 216 operations.

object is represented by a high-resolution volumetric (voxel-based) adaptive octree and the tool object is enclosed within a collection of simple bounding cylinders. Both octree and bounding volumes (BVs) are spatial data structures widely used in many applications [65, 68, 37, 9, 58, 60]. We denote the total number of voxels (root + interior + leaves) in the target object by N, and use M for the number of discrete tool orientations to check. We will consider single GPU-parallel algorithms, where the basic building unit of computation is a CD test, which checks if a given orientation intersects with a given voxel; the adaptive octree will allow both the baseline algorithm and our improved schemes to dynamically prune CD tests when no collisions are possible.

3.2.2 Baseline Algorithm.

Figure 3.4 illustrates our baseline algorithm, which is GPU-parallel. Each GPU thread considers an orientation, traversing the octree to determine whether that orientation yields any intersections with the target. (The code that each thread runs is CHECKOCTREE, shown in Algorithm 2.) During its traversal, the thread performs a CD test at each voxel it traverses, assessing whether the tool at the given orientation intersects the voxel. The intersection calculation is performed by a subroutine referred to as CHECKBOX, which consists mainly

of three computationally intensive geometrical operations: *decomposition*, *rotation*, and *projection*. The decomposition step decomposes the tool into one or more cylinders, the voxel into 6 faces, and each face into 4 line segments. Secondly, rotation changes the coordinate frame so that the cylinder becomes axis-aligned, which greatly simplifies some subsequent calculations and requires 9 elementary operations (e.g., scalar arithmetic). Thirdly, the projection step projects the geometries from 3D space to 2D. In total the algorithm CHECKBOX executes at most $N_c \cdot 6 \cdot 4 \cdot 9 = 216 \cdot N_c$ elementary operations.

3.2.3 Spatial data structures

. Octrees and bounding volumes (BVs) can accelerate the performance of CD by pruning candidate checks. We assume that the target object is in octree where a leaf voxel represents the smallest unit of space that the target occupies. For CNC milling, this representation is especially convenient because it efficiently supports dynamic topology changes such as volume offsetting [65, 68, 37, 9] as well as efficient parallelization. BVs are another type of spatial data structure widely used in many applications. Frequently used types of BVs include bounding spheres, AABBs, and oriented bounding boxes (OBBs), among others [58, 60]. Given the roughly cylindrical shape of tools in milling, we use bounding cylinders to model the tool. Despite this assumption of cylinders, our proposed algorithm can be easily extended to bounding boxes as well (see Section 3.6).

3.2.4 Initial experimental study.

To gain some intuition for how this baseline performs, consider the following experiment on a NVIDIA GTX 1080Ti GPU (see Section 3.5.1 for hardware details).

Suppose we generate an AM with the baseline algorithm. Figure 3.5 shows the execution time as we vary either the object resolution ($N = k^3$ voxels on the x-axis of the left subplot) or accessibility map resolution ($M = l^2$ along the x-axis of the right subplot). Even though increasing the object resolution sharply increases the number of voxels in the

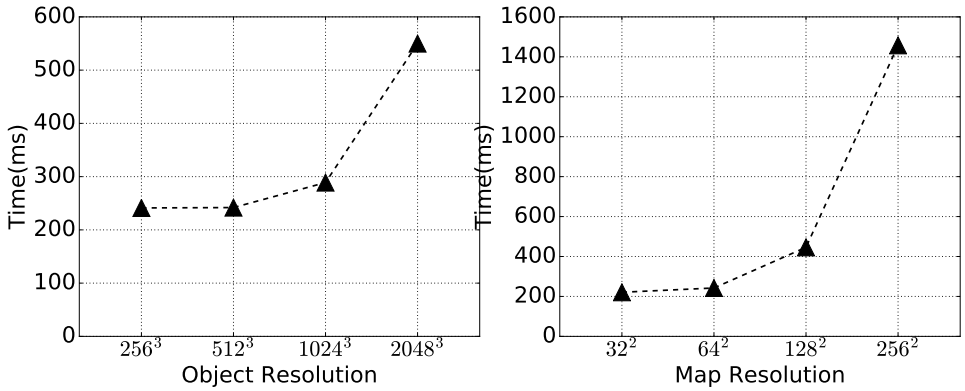


Figure 3.5: Execution time of varying the object resolution in the head model (map size is 64^2) and varying the map resolution (object resolution is 1024^3).

octree representation, the largest observed increase in execution time is a factor of two (2) when increasing the number of voxels by eight (1024^3 to 2048^3 grid). This scaling behavior is sublinear in resolution because the octree induces a pruning of possible checks. By contrast, when the map resolution increases from 128^2 to 256^2, a $4\times$ increase in cells, the corresponding execution time also increases by the same factor. This observation is also not surprising as the total amount of work is proportional to the number of orientations being tested. For relatively low-resolution accessibility maps (e.g., 32^2 or 64^2), the execution time appears flat; that behavior is due to the number of threads of work being less than or comparable to the number of physical execution cores (3,548 CUDA cores on this particular system). However, that absolute time is high enough to prohibit real-time CD. (Real-time is not required in CNC milling but can be in other graphics problems.)

Our paper focuses on the performance improvement of the CD test between cylinders and voxels. It contains two parts: ICA abstraction in Section 3.3 and our parallel algorithm AICA in Section 3.4. ICA aims to reduce the cost of a single CD test. AICA aims to improve parallelism and load balance.

3.3 ICA abstraction

We improve the baseline by first making it more work-efficient, namely, by reducing the high cost of the three basic geometrical operations of decompositions, rotations, and projec-

tions. Our approach is an abstraction, the *inaccessible cone angle* (ICA), which simplifies the 3D operations into 2D equivalents.

3.3.1 Spherical approximation

First, consider the following approximation, designed to reduce the complexity of a CD test: replacing a voxel by a sphere.

A general strategy to make CD tests cheaper is to *axis-align* the objects, that is, perform the calculations in a coordinate frame where one or more axes align "naturally" to one of the objects. Since it is rare that the two objects of an intersection test are simultaneously axis-aligned, we need to axis-align one object and rotate the other. In our method, we choose to axis-align the cylindrical tool because projecting the side surface of a cylinder is more complex than projecting the face of a voxel.

However, calculating the new coordinates of the voxel's geometric elements (e.g., faces, edges) in this new coordinate frame can still be high. In 2D, an axis-aligned line segment becomes skewed after a rotation, and so does a square. In 3D, this problem worsens because a voxel has multiple (six) faces.

By contrast, a sphere would be naturally neutral to axis-alignment regardless of how it is rotated. Consequently, the complexity of an elementary intersection test involving the sphere would be invariant to its rotation. We could, therefore, approximate the voxel by, say, a circumscribed sphere. Doing so truncates the corners, but it is possible to resolve the inaccuracy introduced by this approximation. We explain how in Section 3.3.3.

3.3.2 Inaccessible Cone Angle (ICA)

Suppose we are given (i) a spherical object approximated by a voxel, (ii) a tool composed of several cylinders, and (iii) a position where one end of the tool is fixed. Our goal is to calculate all the *inaccessible* orientations, that is, orientation angles at which the tool collides with the spherical object. Any remaining orientations are *accessible*. With this

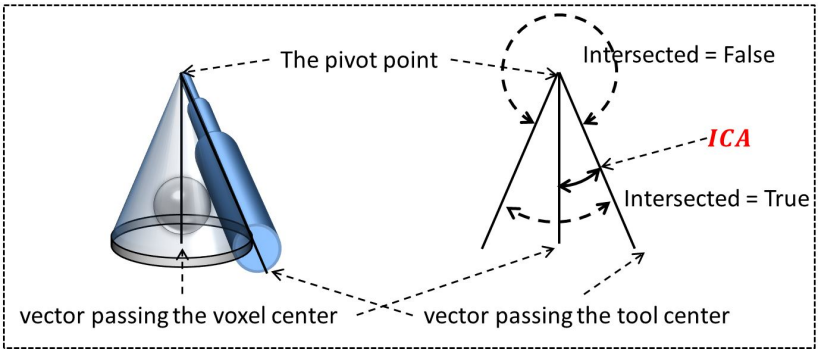


Figure 3.6: (*Left*) How a cone is formed with the tool cylinders exactly touching the surface of the sphere in 3D. (*Right*) How ICA is formed to check the intersection in 2D.

setup, we propose the concept of an *inaccessible cone angle*, or ICA, which represents the possible region of intersection between the tool and the voxel.

Figure 3.6 gives a general example on how a cone is formed in considering a potential collision. On the left, there is the tool, a target sphere, and a pivot point, with vectors from the pivot through the centers of the sphere and the tool. Observe that the tool may touch the sphere at many points, but that the angle between the tool vector and the target vector is constant in all cases. The set of all orientations for which the tool surfaces touch the sphere forms a cone, which we refer to as the *inaccessible cone*: all tool directions within the cone will yield a collision (intersected=**True**), while directions outside are collision-free (intersected=**False**).

The inaccessible cone is associated with an angle between two vectors, one passing through the center line of the tool and the other passing through the center of the voxel. This angle is the ICA, calculated as a 2D value; see Figure 3.6 (*right*). The ICA is the largest angle at which the tool collides with the sphere, or, conversely, the smallest angle at which the tool does not do so.

To calculate an ICA, one must determine at which points a circle might just touch a given rectangle (tool). Figure 3.7 illustrates how to do so. A cross-section of a cylinder that passes its center line is a constant rectangle, and a cross-section of a sphere that passes its center is a constant circle, so these 2D geometries are used to represent the original 3D objects. Our idea starts by logically expanding the size of the rectangle by the radius of

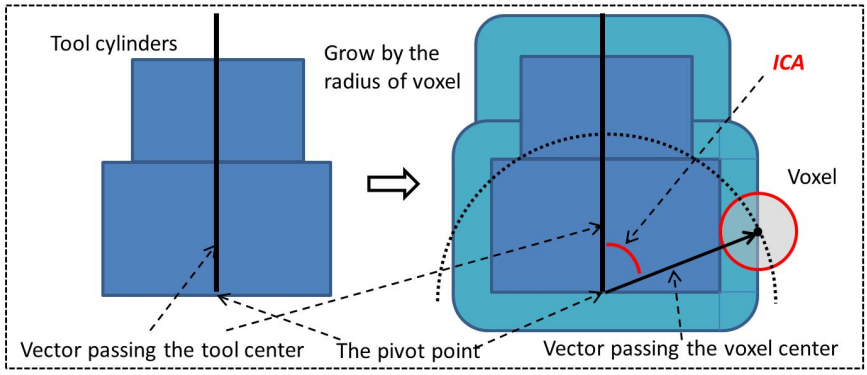


Figure 3.7: The tool cylinders and the voxel are simplified into rectangles and a circle. The voxel's ICA value is calculated as the maximum angle that the circle touches the surface of rectangles.

Algorithm 1 CHECKICA

Input: tool as cylinders, orientation S_i, pivot point p, voxel

procedure CHECKICA(cylinders, S_i, p, voxel)
    $r \leftarrow$ radius of sphere *within* the voxel
    $v \leftarrow$ the center of the voxel
    $\text{ica}_1 \leftarrow \text{GETTOOLICA}(\text{cylinders}, p, v, r)$
    $\text{ica}_2 \leftarrow \text{GETTOOLICA}(\text{cylinders}, p, v, \sqrt{3}r)$
    $\text{vector}_1 \leftarrow$ the center line of tool at orientation S_i
    $\text{vector}_2 \leftarrow$ the vector passing p and c
    angle $\leftarrow$ the angle between vector_1 and vector_2
    if angle $\leq \text{ica}_1$ **then**
        return intersected $=$ **True**
    else if angle $\geq \text{ica}_2$ **then**
        return intersected $=$ **False**
    else ▷ **Corner case**
        return CHECKBOX(cylinders, S_i, p, voxel)
    end if
end procedure

the voxel. Then, fixing the distance between the pivot point and the center of the voxel, it determines all points along an arc, centered at the pivot, that intersect the expanded rectangle. These points are *crossed points*. A crossed point might be located at any point on the border of the expanded rectangle, whether it be on the side, the bottom, or the corner. Crossed points correspond to centers of voxels whose circumscribed sphere just touches the original rectangle.

3.3.3 CHECKICA algorithm to preserve accuracy

The preceding procedure approximates a voxel by a sphere. The resulting CD test may, therefore, yield false-positive reports of accessibility. For instance, if the tool intersects with a corner of the voxel—a literal "corner case"—the approximation will report "accessible" because that corner is outside the sphere. This case is detrimental in CNC milling, where any collision could damage the target part or tool.

The algorithm CHECKICA in Algorithm 1 covers such cases. It considers *two* spheres at each voxel, one inscribed within the voxel (Sphere_1) and one circumscribed about the voxel (Sphere_2), as shown in Figure 3.8. Each of these spheres yields an ICA value. Then, it verifies the following two conditions by comparing the two ICA values as in line 9 and 11. Lastly, if the angle lies between the two ICA values, which is a ***corner case*** in line 13, we cannot verify the intersection using only the definition of ICA; therefore, we must fallback to the original CHECKBOX described in Section 3.2. If in the *absence* of a corner case, the cost of invoking CHECKICA is $N_c{\cdot}2{\cdot}5 + 3 = 10{\cdot}N_c + 3$ operations, where $10{\cdot}N_c$ is the cost of calculating ICA and 3 is the cost of verifying the intersection with the ICA values. Here 2 means the 2 spheres; N_c denotes the number of cylinders in the tool; 5 means there are 5 components to check for each rectangle (cylinder).

One question is how often the CHECKICA algorithm might need to invoke the baseline CHECKBOX, or what is the probability that we encounter a corner case. We define the ***ICA efficiency*** as the fraction of intersection tests that do *not* resort to calling CHECKBOX.

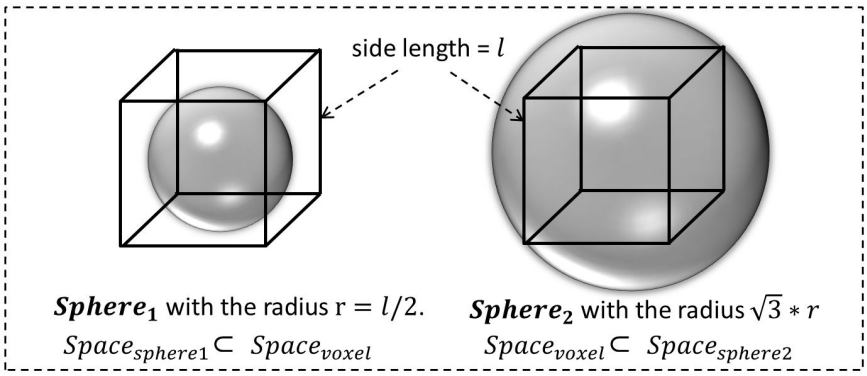


Figure 3.8: Two spheres are constructed for each voxel. For sphere_1, its surface is tangent to the 6 sides of the voxel, and for sphere_2, the voxel's 8 corners are on its surface.

That is, an efficiency of zero means we always call CHECKBOX, and a value of 1 means we never need to call CHECKBOX.

Figure 3.9 derives a theoretical estimate of ICA efficiency, in a simple setting where the cylinders are approximated by a straight line and there are an infinite number of orientations to check. ICA efficiency is inversely proportional to $(\mathrm{r}/\mathrm{dist})$, where r is the radius of a voxel, and dist is the distance from the pivot point to the center of the voxel. In practice, for most voxels the distance should be much larger than the radius, resulting in a higher ICA efficiency than the minimal value: the pivot point must be outside the 3D object, and a point inside the object must result in a collision. The relation between the distance and the radius are crucial to ICA efficiency. As the resolution of the target object increases, the voxel will have a smaller r, thereby yielding a higher ICA efficiency. Thus, ICA efficiency benefits naturally from high-resolution representations.

The concept of an ICA confers several benefits. First, using the ICA in elementary tests does not require any decomposition, since it represents the entire tool. Secondly, the ICA value is independent of the given orientation of the tool. Thus, regardless of the number of orientations a test needs to check, we need only compute the ICA once. Thirdly, the ICA does not require any expensive rotations or projections thanks to the spherical approximation. Compared with CHECKBOX, CHECKICA reduces the overall cost from $216N_c$ operations to $10N_c + 3$ operations, a roughly 20-fold decrease.

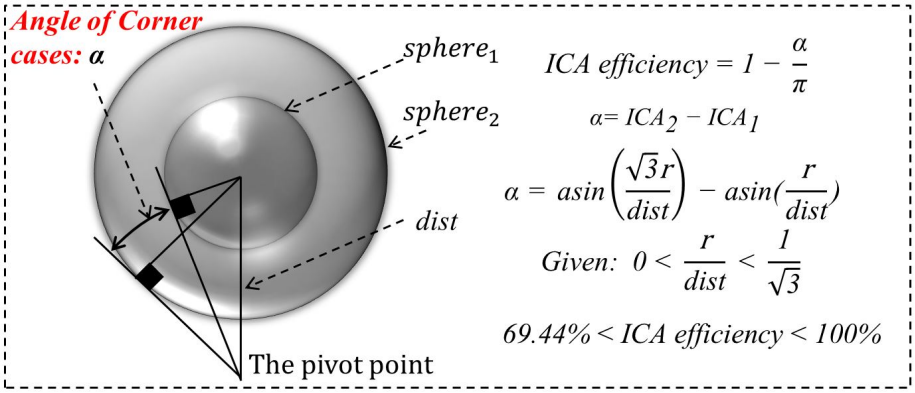


Figure 3.9: Theoretical ICA efficiency analysis. We assume that the sizes of the cylinders do not influence the ICA value and the tool becomes a straight line.

3.4 Design of Parallel AICA

Our approach, AICA, consists of two stages: parallel ICA calculation and parallel CD tests. These are illustrated in Figure 3.10. The inputs to the first stage are the target octree and tool geometry, and the output is a memoized table storing that holding some precomputed ICA values for the upper-levels of the tree. The number of levels is a tunable parameter, described below. In the second stage, each logical GPU thread again checks one orientation; however, when it performs a CD test and calls CHECKICA, the CHECKICA algorithm can now use the memoized table to look up the precomputed ICA values instead of computing them on-the-fly. Any yet-to-be-computed ICA values are computed on-demand. The rest of this section describes our approach to GPU thread mapping, parallelization of the CD tests, and reduction of costly corner cases.

3.4.1 GPU threads mapping

For the parallel ICA calculation, we compute the values of the voxels on the top S levels in octree. Each thread corresponds to a voxel, yielding a highly efficient SIMDization.

For the parallel CD test, given the workload with the N voxels and the M orientations, there are two natural parallelization strategies. One is to partition the octree among threads, and then each thread processes M orientations. The other is to map each orientation to a thread and then each thread will traverse the octree, as with the baseline algorithm. We use

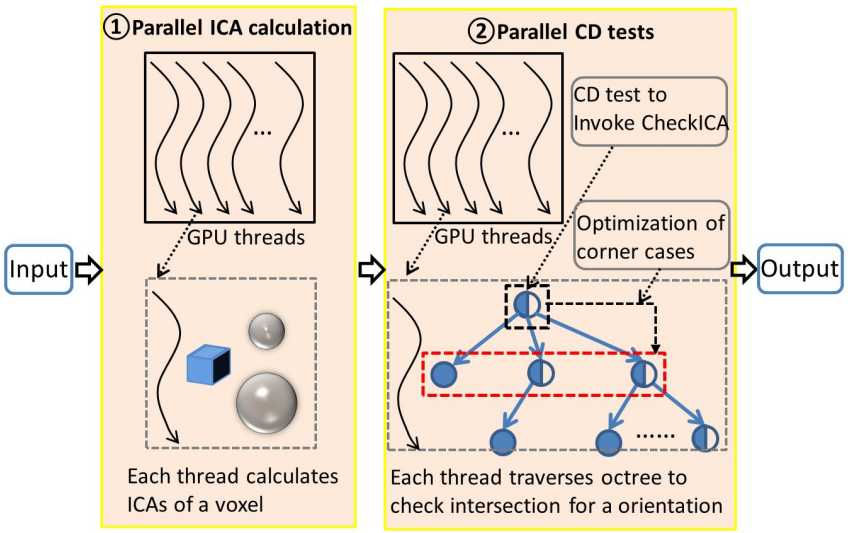


Figure 3.10: Overview of aggressive inaccessible cone angle (AICA) with two stages: parallel ICA calculation and parallel CD tests.

Algorithm 2 CHECKOCTREE

Input: tool, orientation S_i, pivot point p, octree

```
1:  procedure CHECKOCTREE(tool, S_i, p, octree)
2:      stack ← {voxels at the top level of octree}
3:      while !stack.IsEmpty() do
4:          voxel = stack.pop()
5:          intersected = CHECKBOX(tool, S_i, p, voxel)
6:          if !intersected then
7:              continue
8:          else if intersected & voxel.IsLeafNode() then
9:              return True
10:         else
11:             stack.push_back(voxel.children())
12:         end if
13:     end while
14:     return False
15: end procedure
```

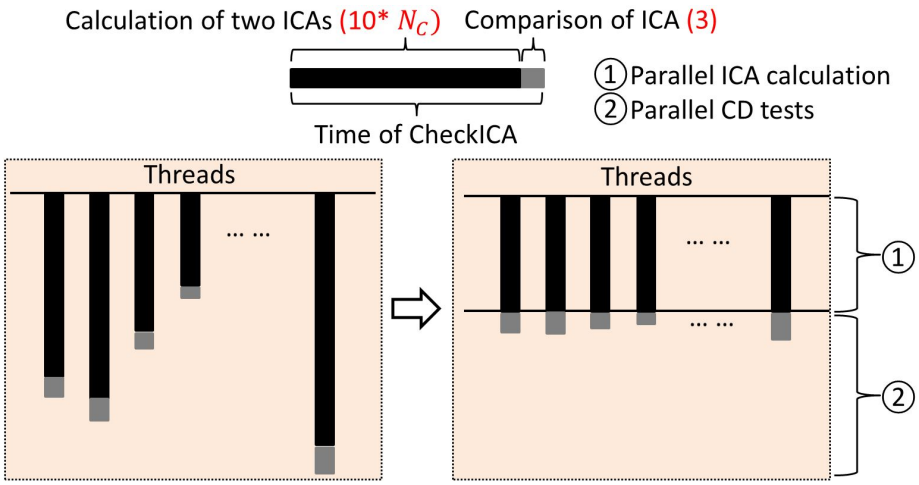


Figure 3.11: The parallel ICA calculation mitigates load imbalance and improves the performance, by saving the cost of redundant ICA calculation and efficient parallelization.

the latter, for two reasons. One is that it enables more aggressive exploitation of the adaptive octree for pruning. Finding an interior voxel node that does not intersect with the tool can avoid any calculations on all of its descendants; similarly, a solid voxel that intersects with the tool means that we can directly return that a collision will occur. Another reason is simplicity: assigning threads to orientations is an owner-computes strategy that avoids communication and synchronization. The overall algorithm, which each thread executes, appears in Algorithm 2.

3.4.2 Mitigating load imbalance

The choice of thread mapping affects load imbalance in the baseline. The execution time of a thread is determined by the number of checks at the line 3 of Algorithm 2, which varies with each orientation. To mitigate this load imbalance, we calculate ICAs of the voxels on upper S levels of octree in parallel as a precomputation stage before the parallel CD tests, rather than calculating ICA at runtime as shown in the left side of Figure 3.11. In practice, each thread's calculation of an ICA and comparisons alternate with checks, and the time spent in the two phases appears in Figure 3.11. We create, for each voxel in the target, a memoized table of ICA values. These are the values labeled ica_1 and ica_2 for the

two spheres, meaning one pair per voxel. This increased storage is a modest fraction of the total voxel storage and this precomputation is pleasingly parallel since there are no inter-voxel dependencies. Precomputation is feasible because ICA is independent of the tool's orientation.

This approach confers three overall benefits. First, it avoids redundant calculations as the ICA values in the table are calculated once but reused by all threads. Secondly, it mitigates load imbalance as calculating ICA in the precomputation stage is easily parallelizable, at the granularity of voxels. Lastly, it reduces the execution time of all threads and thus improves overall performance because of efficient SIMDization on GPU.

As S increases, the total cost of all CHECKICA tends to increase, whereas the amortized cost of CHECKBOX tends to decrease. Thus, there is a tradeoff. A heuristic is that S can be set to a relatively higher value on recent, more powerful GPUs (Section 3.5.4).

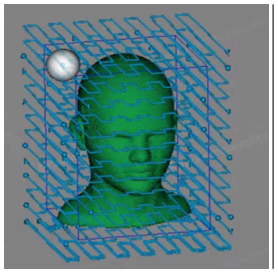
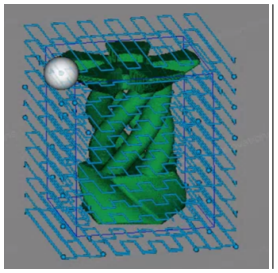
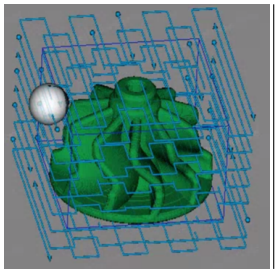
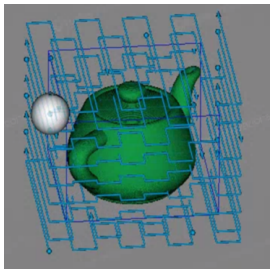
3.4.3 Optimization on the corner cases

For corner cases, we may still need to invoke the baseline CHECKBOX to verify the intersection. However, we can reduce the corner case cost by utilizing the hierarchical spatial structure as an optimization.

Suppose that the algorithm stops at a voxel facing the corner case as shown in Figure 3.12 (left). We have two choices. One is to directly invoke the baseline algorithm. The other is to expand the voxel into its children voxels, and then apply our CHECKICA algorithm recursively on each voxel; the recursion stops when no further expansion is allowed, in which case CHECKBOX is still used as the fallback. Our optimization approach is to choose the latter.

The cost of this approach is an increase of the number of checks resulting from the expansion. Nevertheless, the benefit is the reduction in cost of CD test by invoking CHECKICA. We believe that the benefit largely outweighs the cost. Note that the cost of a single CHECKICA is 3 here, rather than $N_c * 10 + 3$, since most voxels' ICA values have already

Table 3.1: Geometric statistics of sample CAD Benchmarks.

	Head				Candle Holder				Turbine				Teapot			
																
Number of Triangles	23028				38000				57792				57600			
Dimension XYZ(mm)	48.6*46.0*64.4				48.4*48.9*57.7				48.9*48.9*31.1				46*46*31			
Bounding Volume	51331				21275				7823				25619			
Effective Resolution	256^3	512^3	1024^3	2048^3	256^3	512^3	1024^3	2048^3	256^3	512^3	1024^3	2048^3	256^3	512^3	1024^3	2048^3
#layers	6	7	8	9	7	7	8	9	6	7	8	9	6	7	8	9
#voxels in octree(10^6)	0.44	1.06	4.26	17.56	0.57	1.59	5.92	26.94	0.62	1.37	6.44	26.06	0.74	1.53	6.14	23.89
#points on path(10^3)	61.14	101.3	203.7	409.3	58.32	97.32	196.9	360.6	29.43	41.46	83.48	168.2	30.60	44.57	89.37	179.1

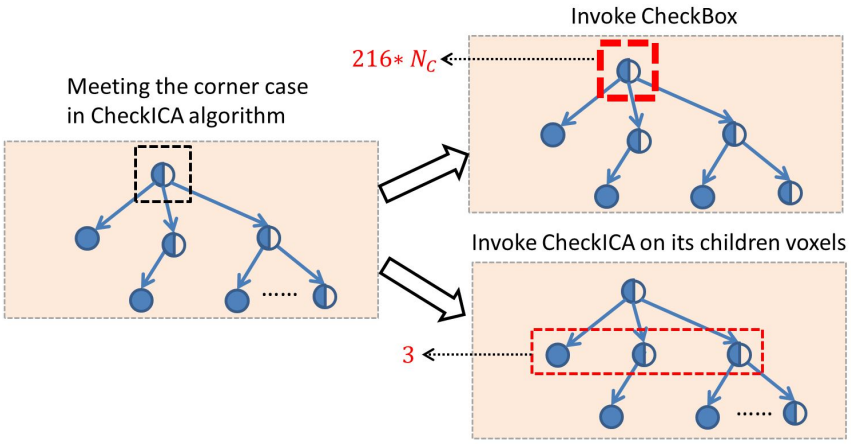


Figure 3.12: Optimization on the corner cases. When meeting a corner case on a voxel, AICA algorithm expands it into its children voxels and calls CHECKICA recursively.

been calculated in the precomputation stage. The corner case is also an important factor that causes the load imbalance, so we will benefit from optimizing it, too. This tradeoff will be evaluated in detail in Section 3.5.2.

3.5 Evaluation

We evaluated three aspects of our approach:

- (1) examining the impact of the parallelism method and verifying the efficacy of ICA efficiency;
- (2) assessing absolute performance with various object resolutions and various AM resolutions;
- (3) analyzing the cost of the parallel ICA calculation under various configurations.

Our comparison includes AICA and four other schemes:

- A parallel box (***PBox***), which is the baseline algorithm with parallel CD tests using CHECKBOX.
- ***An optimized*** **PBox** is still on the baseline algorithm but using axis-aligned bounding boxes (AABBs). The optimization is to apply AABBs on the voxel after each

rotation. If no intersection exists on the bounding box, we can directly return False.

- A parallel ICA (***PICA***), which is the algorithm with the parallel CD tests using CHECKICA.
- A memoized ICA (***MICA***), which is the algorithm that has the parallel CD tests using CHECKICA and has the parallel ICA calculation but without the optimization of corner cases.
- Our approach, ***AICA***, which has both.

The first two—*PBox* and *optimized PBox*—represent the state-of-the-art, and are both implemented in SculptPrint.

Table 3.2: Experimental test platforms.

Two Setups	GTX 1080 Ti	GTX 1080
CPU	Intel i7-2600K 3.40GHZ	Intel i7-7820HK 2.90GHZ
DRAM	16GB	32GB
OS	Windows 7	windows 10
CUDA runtime	9.1	10.0
GPU card	11GB, 1.68GHZ 3548 CUDA cores	8GB, 1.77GHZ 2560 CUDA cores

3.5.1 Experimental Setup

CAD Benchmarks. Our experiments use 4 CAD benchmarks for evaluation. A summary of the input meshes and their detailed geometrical characteristics are listed in Table 3.1. For each benchmark, we evaluate 4 target resolutions on the construction of octree, from

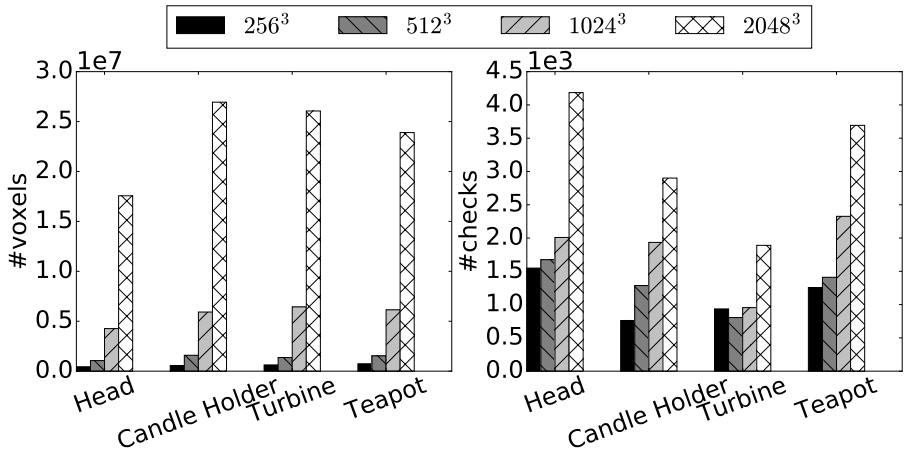


Figure 3.13: Comparison between the number of voxels in octree and the number of checks under various object resolutions. The actual number of checks on the critical thread is much smaller than the total number of voxels.

256^3 to 2048^3. The tool geometry has 4 cylinders, with varying radii (31.5, 20, 6.225, 6.35)mm and heights (22.1, 78, 76.2, 25.4)mm. The AM resolution starts from 32^2 to 256^2. To choose representative pivot points, we generate a path surrounding the CAD models, with each point on the path having a 1 mm distance from the surface of the model.

Configuration. We implement our algorithms in SculptPrint, a computer-aided manufacturing (CAM) application for producing CNC tool paths [69]. During the process of generating an AM, we assume that all of the information about the 3D object model has already been loaded onto the GPU, since this information is read-only and only need only be loaded once. There is no overhead of memory copy between CPU and GPU in the runtime. Thus, the cost of the transferring is excluded in our experimental results.

Table 3.2 presents two platforms for our experiments: GTX 1080 Ti and GTX 1080. The main difference is that GTX 1080 Ti has more CUDA cores (3548 than 2560), but with a lower frequency (1.68 GHZ than 1.77 GHZ).

In our experiments, 2000 random points are chosen from the path as the pivot points. The last row of Table 3.1 shows the total number of points on the path. Every experimental result in this section is the average value of the 2000 samples. We directly expand the top 5 levels of octree into one level, and report the final number of levels under various

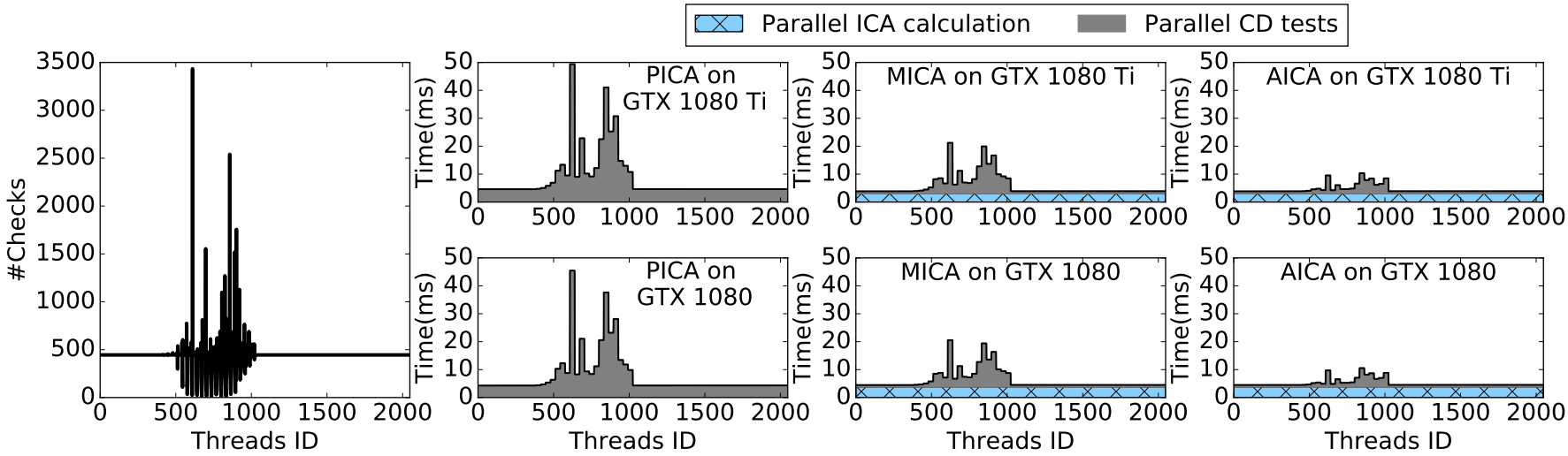


Table 3.3: The parallel ICA calculation mitigates the load imbalance and improves the performance, with the head model with 1024^3 resolution and 2048 orientations. The first column plot shows the actual number of checks on all threads.

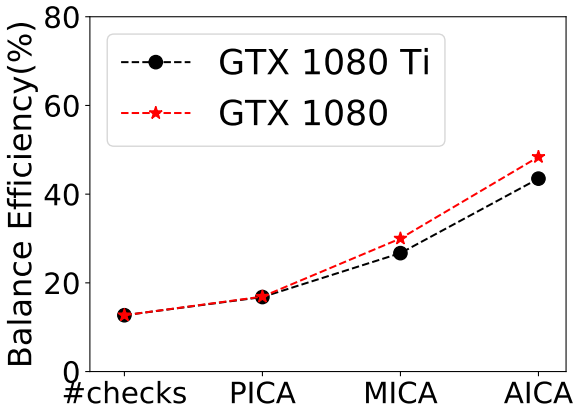


Figure 3.14: Our parallel algorithms can gradually increase the balance efficiency. Our proposed algorithm AICA is the most load-balanced.

resolutions in Table 3.1. This expansion aims at reducing the height of octree (see the load imbalance part in Section 3.5.2). The parameter S is set to 8, which means that the parallel ICA calculation stage computes the ICA values for the voxels on the upper 8 levels ($\sim$7 billion voxels).

3.5.2 Analysis of Parallelism Exploitation

Threads mapping. Figure 3.13 shows the total number of voxels under various resolutions for the 4 models. However, the total number of voxels does not reflect the actual workload on each thread. The right-hand plot presents the actual number of checks on the critical thread. Because of the spatial hierarchy of octree in Algorithm 2, each thread unnecessarily traverses all voxels. It is obvious that the number of the actual checks is much smaller than the number of voxel in octree, indicating that our approach of the threads mapping is very efficient.

Mitigating load imbalance. Table 3.3 shows how the parallel ICA calculation stage mitigates load imbalance and, thereby, improves performance. The leftmost plot shows the actual number of checks executed by each thread. The leftmost and rightmost threads run the same number of checks, because we expanded the top 5 levels into 1 level as mentioned before, and these threads have to check all voxels on the top level before returning. In the

two plots of the second column, we can see that the execution time is proportional to the number of checks, where CHECKICA is used for CD tests. Comparing the two GPU cards, we can see that the time on GTX 1080 is a little shorter than the other, because GTX 1080 has a higher clock rate 1.77 GHz than 1.68 GHz of GTX 1080 Ti.

In the two plots of the third column, the bottom area represents the time of the parallel ICA calculation, which mitigates the load imbalance by reducing the execution time of calculating ICA values. Note that the CD tests in the corner cases are not influenced by the parallel ICA calculation, which still takes a relatively long period of time. Comparing the two GPU cards, we can see that the time of the parallel ICA calculation on GTX 1080 ($\sim$ 3.8 ms) is longer than the other ($\sim$ 3.1 ms), because GTX 1080 has 2560 CUDA cores while GTX 1080 Ti has 3548 CUDA cores. The last column of Table 3.3 shows the cost of the corner cases and the effect of our optimization technique, which effectively improves performance further from the previous column.

We define the balance efficiency as the percentage of the actual executions to the maximum execution over all threads. 100% means completely balanced, where all threads have exactly the same execution. Figure 3.14 present the percentages for the parallel algorithms. Our AICA algorithm has the best efficiency of 43.5% and 48.4% on the two platforms.

Optimization of corner cases. Since the performance of generating the AM is determined by the thread that has the longest execution time, namely the critical thread, we only report the execution on the critical thread.

Figure 3.15 reports three types of percentages: box checks in MICA, box checks in AICA, and the increased percentage of the total check from MICA to AICA. Recall that our optimization of corner cases reduces box checks at the expense of increasing the total checks, where the cost of a single ICA check is much smaller than the cost of a single box check. We can see that the percentage decreases from 14.4% to 0.9% on average, comparing AICA with MICA, resulting in a 34.1% increase on the total number of checks. Suppose that we need to reduce S number of box checks; then, the number of required ICA

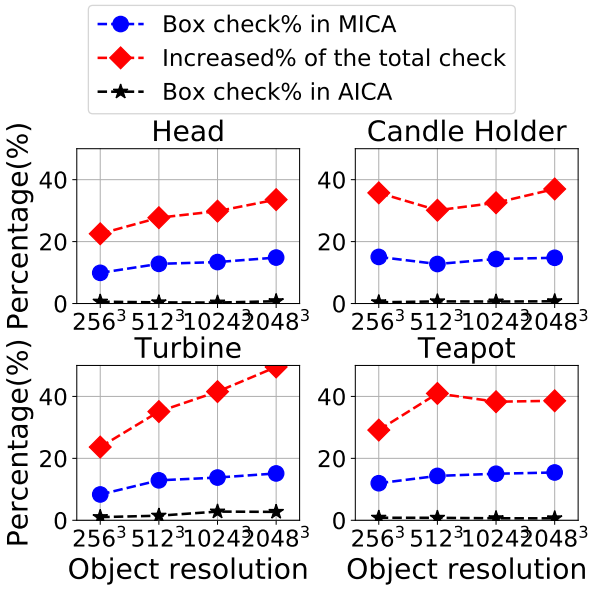


Figure 3.15: Optimization of corner cases. An intentional increase of the total checks is made to reduce the number of Box checks from MICA to AICA, where ICA efficiency (=1-Box checks%)

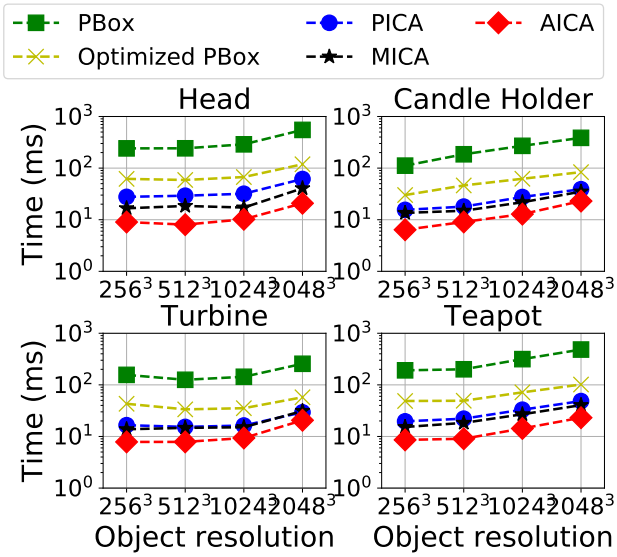


Figure 3.16: Averaged execution time of 5 approaches with various object resolutions. Our approach AICA performs $23.9\times$ faster than the approach of CHECKBOX, and $4.8\times$ faster than our best optimized version of CHECKBOX.

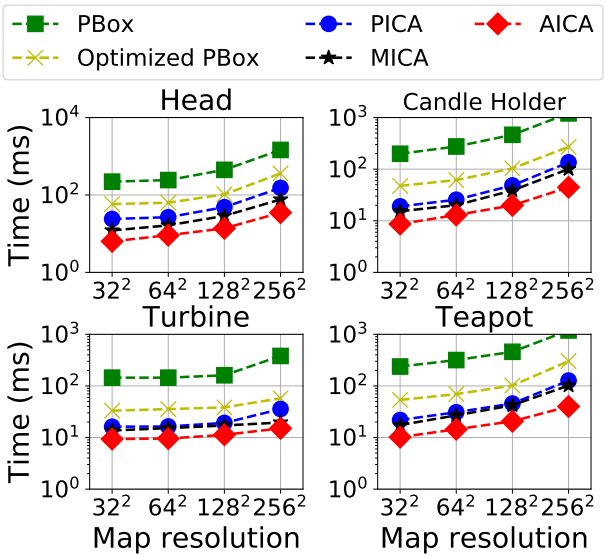


Figure 3.17: Averaged execution time of 5 approaches with various AM resolutions. Our approach AICA performs $20.2\times$ faster than the approach of CHECKBOX, and $4.1\times$ faster than our best optimized version of CHECKBOX.

checks should be larger than the number S as the cost, because one Box check demands a substitute of multiple ICA checks on the expanded children voxels.

However, increasing the number of ICA checks is worthwhile because doing so reduces the number of box checks and improves performance, given the inherent difference in costs between the two types of checks. The remaining percentage of box checks is the actual ICA efficiency, which is 99% on average, indicating that 99% of CD tests benefit from the ICA abstraction.

3.5.3 Overall Performance Results

Varying Object resolution. Figure 3.16 shows the average execution time for all 4 models. For PICA, it is $23.9\times$ faster than PBox, and $4.8\times$ faster than the optimized version on average of the 4 models. That is because CHECKICA only needs 2D computations with the ICA and avoids most of the three computation-intensive operations that exist in CHECKBOX. MICA improves the speed by 28.3% on average compared to PICA. The simple parallelization of the ICA precomputation is faster than the on-the-fly ICA compu-

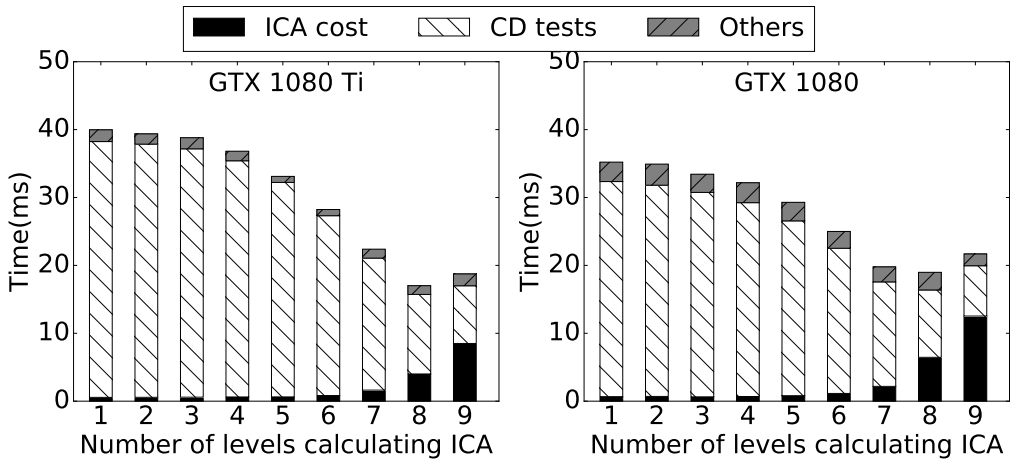


Figure 3.18: Time breakdowns under various numbers of layers in octree, using the head input model in 2048^3 resolution with AICA approach. Though the ICA cost increases as the growth of the layers, the overall performance is improving.

tation, even though MICA memoizes ICA values for all voxels, while PICA only applies the calculations on the voxels in the current test. Note that the cost of the precomputation increases as the number of the voxels grows. On the 256^3 resolution, the improvement is 32.5% while for size 2048^3, the improvement becomes only 19.3%. A detailed discussion of the cost on varying the number of voxels in octree is given in Section 3.5.4. For AICA, it is 81.1% faster than MICA on average, indicating that increasing the total number of the checks can still yield a higher ICA efficiency. A detailed analysis of this tradeoff is given in Section 3.5.2.

Varying the resolution of the AM. Increasing the resolution of the AM leads to a large number of orientations to check. The object model resolution is fixed to 1024^3. The 32^2 resolution requires 1024 threads and 64^2 needs 4096 threads. The experiments run on GTX 1080 Ti that has 3548 cores. This number of cores explains why the increasing ratio of the execution time from 32^2 to 64^2 is smaller than the others. Figure 3.17 presents the average execution time of all 4 models. For PICA, it is $20.2\times$ faster than PBox on the average of the 4 models, and $4.1\times$ better than our optimized CHECKBOX. For MICA, it is improved by 39.5% compared to PICA. AICA achieves 84.8% improvement than MICA due to a high ICA efficiency, which it includes the cost of creating the memoized table.

3.5.4 Cost Analysis

Calculating ICA affects execution time in the manner shown in Figure 3.18. There are three types of costs: ICA cost, CD test and others. Others is mainly the time of launching the GPU threads, not including the memory copy between CPU and GPU. The x-axis represents the number of the upper levels in octree that we calculate the ICA values. If the ICA value is not precomputed, it will be calculated in runtime. In both subplots, for the upper 3-4 levels in octree, there is not much time reduction of CD tests. This is because the execution time is always bounded by the critical thread. Starting from the 5 upper level, an obvious reduction of the time appears, and meanwhile, the cost of calculating ICA increases as the number of voxels grows exponentially. Taking the cost into account, it is still worth calculating ICA for the voxels on the uppper 8 levels to speed up the CD tests.

We also explore variations in the object resolution from 256^3 to 2048^3. Figure 3.19 presents the performance results. The execution time increases gradually, as the cost of ICA calculation increases with an exponential growth of the number of voxels. Therefore, we believe that using a more powerful GPU card can reduce the cost further, and thus achieve better performance. If we do not have a new GPU card, we still can gain a decent performance by tuning the parameter S to control the cost of the parallel ICA calculation.

3.6 Apply ICA to bounding box

ICA abstraction is also applicable to bounding box. Bounding box is a general data structure used to simplify computation by enclosing a series of arbitrary objects. Similar to the way that a voxel is approximated by 2 spheres described in Figure 3.8, a bounding box can also be approximated into 2 cylinders. Both cylinders are able to use ICA for the checking. There should be certain corner cases that are not covered by ICA, but the percentage should be very small, similar to our ICA efficiency analysis. Therefore, ICA is a general geometric abstraction that substantially simplifies CD tests. We believe that our ICA abstraction

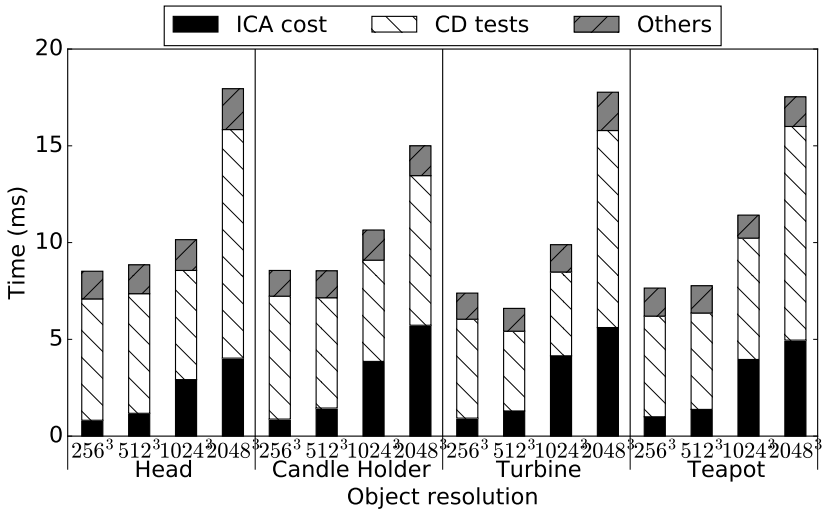


Figure 3.19: Time breakdowns under various resolutions of object models with AICA. As the object resolution rises, most of the increasing portion comes form the ICA cost.

will be implemented into a new standard elementary procedure in many other tests, such as cylinder-sphere, sphere-box, and box-box CD tests.

3.7 Related work

The problem of efficient CD has been well studied and excellent surveys are available [70, 71, 72]. In this section, the approaches of improving the performance of CD are classified into three categories: acceleration using graphics representations, acceleration using novel parallelization schemes, and lastly others.

Many graphics techniques including spatial data structure [65, 68, 37, 9], culling [62], ray casting [73], visibility query and collision map, are commonly used for approximation of CD [10, 11, 74]. Using ray-casting algorithm is proposed in the paper [73] with hardware frame buffer operations to optimize the performance of CD between solid polyhedral bodies. By using depth maps store distance values to represent outer shape of objects, Kolb [16] handles collision detection and reaction of particles with objects for arbitrary shape in massively parallel simulations. Sucan [18] describes a collision map data structure, which uses axis aligned cubes to model the point cloud and to perform collisions. Bounding sphere is commonly used for CD acceleration as an encapsulation of the target

object [75, 76, 77, 78] with an approximation. Our proposed method is different from these approaches. because our tool is composed of multiple cylinders with arbitrary sizes, which can not be approximated into a straight line, and more importantly we need to preserves accuracy.

Many algorithms [79, 65, 80, 64, 67, 17] have been proposed to exploit computational capabilities of a multi-core platform. Lauterbach [67] presents novel GPU-based algorithms to efficiently perform collision and separation distance queries using tight-fitting bounding volumes. With the goal to compute collision-free paths for robots in complex environments, Pan [17] presents a novel GPU-based parallel algorithm to perform collision queries for sample-based motion planning. A novel hybrid parallel continuous collision detection is proposed by Kim [66], which utilizes both CPUs and GPUs to achieve the interactive performance of CD. Instead of focusing a shared-memory test bed, Du [63] targets on high-performance cluster with a parallel continuous collision detection(CCD) algorithm aiming to accelerate CCD culling by efficiently distributing workload. Our parallel algorithm can be integrated with these various parallel schemes to explore efficient parallelisms.

Pan [81] formulates collision checking as a two-class classification problem, applying machine learning to compute the collision probability for acceleration. Ding [58] conducts the interference detection between the tool oriented bounding boxes and the gray octants of the surface octree in order to simplify the computation process of updating tool positions and orientations in 5-axis machining. Zhiwei [82] proposes an efficient algorithm of CD to generate tool posture collision-free area for the whole free-form surface by sampling and cubic B-surface interpolation. Since these techniques still need to process the elementary CD tests, our proposed abstraction can be embedded to further improve the performance.

3.8 Conclusions and Future Work

The key ideas of our proposed methods are the ICA concept, which is a new geometric abstraction for the CD problem, and its parallel algorithm AICA, including the mitigation of load-imbalance and the optimization on corner cases. We have prototyped our AICA algorithm within a real CNC milling tool, SculptPrint [69]. Experimental results on 4 CAD benchmarks demonstrate that AICA is up to $23\times$ faster than the approach of the traditional approach.

While our results show ICA can be effective, our experimental analysis also identifies several new opportunities. For instance, neighboring pivot points, which were outside the scope of this paper, are likely to have AM with overlapping values. Therefore, future work should develop methods to reuse the AM values among nearby pivots. Another idea is to construct an algorithm that can intelligently tune the parameter S to adjust the cost of the parallel ICA calculation. Lastly, to broaden its use in computer graphics, our AICA should be extended and tested against other spatial volume structures common in that domain, such as BVH (Section 3.1) and kd-trees, among others.

CHAPTER 4
MAX ORIENTATION COVERAGE BY AVOIDING COLLISIONS

This chapter moves to the CPP problem of 3D objects using the information of the accessibility map output from the proposed method last chapter. We present our proposed approach called max orientation coverage, which employs a two-step optimization scheme. It can improve path efficiency with respect to both the path length cost and the cost of dealing with the constraints of avoiding collisions.

4.1 Introduction

Coverage path planning has many applications, including robotic vacuum cleaners, aerial robotic inspection [83, 84], 3D printing [11, 37, 9], and autonomous underwater vehicles (AUV) [20, 21, 22], among others. To facilitate autonomous path planning for complex environments or objects, a robot must be equipped with algorithms capable of computing efficient paths that achieve full coverage while respecting any robot limitations or motion constraints that may apply.

We are specifically motivated by problems in CNC milling, and Figure 4.1 gives an example from that domain. The problem is to cover the object (e.g., the head) where the "robot" is a milling tool composed of multiple cylinders. The constraint is that the robot (tool) and the object ought not collide *except* at the pivot point, which is where the end of the tool touches the surface. Any other collision would be detrimental to the milling process, damaging the robot or the target. To represent possible collisions, an AM is constructed to determine, at each pivot point, which tool orientations are inaccessible (e.g., the right side of Figure 4.1). When generating a valid path, the algorithm has to ensure that no collision exists by selecting only accessible orientations at all points.

Conventional coverage path planning methods either assume that the robot is under no

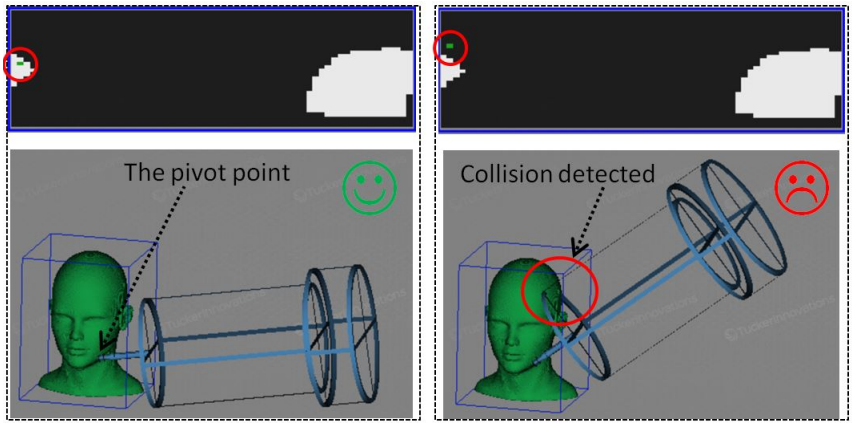


Figure 4.1: An example of AM shows two orientations: accessible orientation (on the left) and inaccessible orientation (on the right), corresponding to the white point and the black point in the top AM respectively. An AM at $(r * c)$-resolution is discretized uniformly into $r * c$ points, with each point denoting a (θ, φ) orientation in a spherical coordinate system.

constraint or focus on the optimization of the path length, and ignore this type of constraint. Most times for safety reasons, satisfying the constraint may require conservative operations, like retracting the tool and resetting its position or orientation, which can be very expensive in practice. Focusing on the path length only would underestimate such costs, leading to an inefficient path.

Tool path planning problem is different from conventional CPP as it is specifically designed for the milling applications to generate collision-free paths. Most prior approaches assume the tool is a single cylinder and only a small piece of the object surface (e.g., the place that the tool touches the surface) can cause collisions. However, in reality a tool has multiple cylinders with various sizes and anywhere on the object might cause a collision. Checking collision under this situation demands massive computational resources, which is the reason why the tool path planning problem has the simplification described above. Thus, these approaches can not be extended for a generic coverage.

We propose a novel path planning algorithm, called *max orientation coverage*, that constructs an efficient path generally covering an arbitrary object. We consider both the cost of path length and the cost of operations handling the constraints. It has two components.

The first is a segmentation algorithm, which we call *max segmentation* (Section 4.4.1).

Prior segmentation algorithms rely on random sampling to achieve a full coverage. Doing so is usually unstable and leads to a large number of segments, resulting in a higher likelihood of redundant coverage. By contrast, our segmentation algorithm can obtain a stable result and reduce the number of segments. Our approach is to convert the coverage optimization problem into a minimal vertex cover (MVC) problem.

The other component is a new coverage algorithm based on orientation (Section 4.4.2). Most earlier approaches focus on optimizations of the path length and do not consider the effects of dealing with the constraints on the robot. By contrast, our approach does so, trading off small increases in path length to significantly lower the cost of enforcing the tool constraints and thus yield better overall paths.

Additionally, we conduct several simulations to help validate and evaluate our approach. In particular, we test it on four CAD benchmark objects against a previously proposed random sampling-based coverage algorithm [85, 8, 84, 86, 22]. We observe that our max orientation coverage improves the path efficiency by 29.7% on average and the improvement goes up to 46.5% for one of the geometrically complex objects, the dragon (Figure 4.6).

4.2 Related work

Many contributions have been made to address the CPP problem. We divide them into three types: the approaches of covering general environments, the approaches of covering 3D objects and the approaches of milling applications.

Several literatures [23, 87] provides in-depth and comprehensive surveys on coverage path planning. Gabriely [88] presents an online approach that constructs a systematic spiral path with a spanning tree algorithm. Luo [89] presents an algorithm that utilizes neural network to generate a real-time path. Choset [24] proposes a coverage named boustrophedon cellular decomposition for a known environment. Acar [90] applies cellular decomposition considering sensor-based detector. Atkar [91, 92] extended the ideas of cellular decom-

position into 3D objects. Some other approaches proposes some techniques specific to applications such as under-water 3D coverage [20, 21]. Most these approaches focus on the path lengths as the optimization target, while our method concerns an additional cost of handling the restraints.

For the approaches that cover 3D objects, random sampling-based coverage approaches [85, 8, 86, 84, 93, 83, 22, 94, 95] are commonly known as the state-of-the-art algorithms to handle versatile 3D coverage problems, which employs a two-step optimization. The first step is a segmentation problem to compute a minimal set of viewpoints using random sampling to achieve a complete coverage of the target structure. This step is equivalent to solving a variant of art gallery problem (AGP), which is a well-known NP-hard problem. In the second step, a minimum cost tour over all the viewpoints is searched to optimize the tour length. This involves solving a variant of TSP. Although the two problems are NP-hard, there are fast algorithms that approximately solve the two problems, for example [96, 97] for AGP, and [98, 25] for TSP. Our proposed method follows the structure of the two-step optimization to reduce the computational cost. Instead of doing random sampling, our method collects viewpoints by solving a MVC problem without loss of generality.

Tool path planning problem is another type of problem that concerns avoiding collisions for an efficient milling process. Cho [99] propose to use a potential energy method to avoid collision and improve the machining efficiency, which represents Cartesian space by an artificial energy field. When the cutter and part surfaces are virtually charged with static electricity, a potential energy field is formed for collision detection. Jun [100] presents a methodology of optimizing and smoothing the tool orientation control 5-axis sculptured surface machining. A searching method in the machining configuration space is proposed to find optimal orientation by considering various types of collisions. Chu [101] presents a novel approach that automatically generates a collision-free tool path 5-axis milling of a ruled surface. In the method, the surface needs to be divided into curve segment. Each segment works as a guide curve for developable Bezier surface with available degrees of

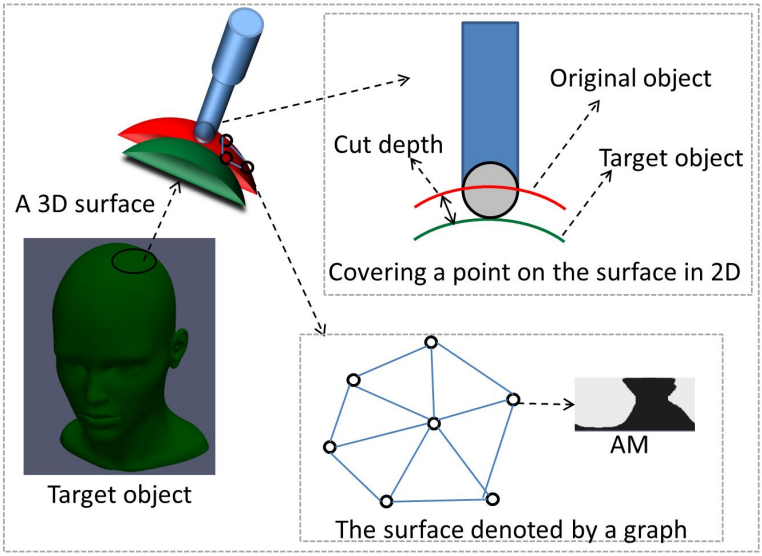


Figure 4.2: A solution to the CPP problem is one step in the milling application.

freedom. Hsueh [102] proposed a two-step method for preventing tool collisions in 5-axis machining. Each step adjusts the tilt angle and the yaw angle respectively. Wang and Tang [103] present a method that concerns both collision avoidance in 5-axis machining and angular-velocity that avoid drastic orientation change. Chu [104] presents a tool path planning framework for 5-axis machining of centrifugal impeller with split blades. A planning template was proposed for specifying four operations: a method of machining control, selection of tool path pattern, determination of tool orientations, and calculation of tool paths.

Our proposed method is different from them in the following two ways. Firstly, most prior approaches assume that the tool is a single cylinder, while ours allows the tool to be multiple cylinders that have various sizes. Secondly, most of them only consider the collision analysis in a "local" coordinate system, assuming that only a small piece of the object surface can cause collisions and other parts can be ignored, while our approach allows the object to be arbitrary shape that results in collision anywhere on the object.

4.3 Background and candidate approaches

This section starts with how CPP problem can be used for our CNC milling application. Then we describe the input to our problem, the constraint and the problem statement. Lastly, we define the cost as our optimization target.

4.3.1 CPP problem in CNC milling application

A CNC milling application is to convert a given object (e.g., the stock in Figure 4.1) to a target object (e.g., the head) through a milling process. We assume that the object always stays inside of the stock. A CPP problem is to cover the surface of a 3D object. To obtain the target object for the milling application, multiple steps of solving a CPP problem are required.

Figure 4.2 illustrates one step of covering the surface of the head for the milling process. The original object is the object before applying the milling and the target object is the result after it. The space difference represented by the cut depth in Figure 4.2 denotes the materials that are machined off by the path generated by the CPP problem. Usually the cut depth equates to the size of the robot. For these steps, the milling process uses various robot sizes in a decreasing order. Initially, a large robot size is preferred to get a coarse shape. Finally, a small robot size is used to obtain a high-quality surface.

The surface of the object is represented by a dense mesh as shown in the right bottom of Figure 4.2. The length of the edge equals to the size of the robot, so that traversing all vertices in the mesh can achieve the covering of the surface of the object. To ensure a safe movement from one vertex to another, each vertex has a corresponding AM, indicating that only accessible orientation can be specified. Note that at a vertex, we only allow the movement to its neighboring vertex following the AM. This paper studies the CPP problem for a single step that covers an arbitrary object.

4.3.2 AM as an input to our problem

AM is given as an input, since our focus is on how to construct an efficient collision-free path, rather than how to check possible collisions, which is proposed in our previous work [105]. As a preliminary task, the AM of all vertices on the surface of the object are calculated. The optimization is on how to specify an orientation for each point to construct a valid path. It has a high cost to calculate an accurate AM for each point, because the object keeps changing in the process of the milling. Instead, we calculate the AM of all points only once in a parallel way at beginning of each step of the CPP problem, under an assumption that the object remains the same. This is conservative on avoiding collisions. If an orientation has no collision with the original object, we can guarantee that the orientation is safe with the current object undergoing the milling, because it is a shrink version of its original object.

4.3.3 Constraint of avoiding collision

This paper targets at the optimization of the cost of dealing with the constraint that the path has to be collision-free, where the robot needs to smoothly move from one point to another. Even if specifying an accessible orientation, we can not assume that the orientation can be changed abruptly. Instead there is a time-cost when changing from an orientation to another. Note that we do not concern any joint limits [106] [107] or other kinematic constraints [108][109][110]. The purpose of handling the constraint is to gain a smooth movement from one point to another. There are three possible situations describe in the following, and we have two types of operations accordingly, called reorientation and retraction, as shown in the bottom of Figure 4.3.

- 1) Given that the current orientation is accessible at the next point, we directly conduct the movement without worrying the orientation.

- 2) Given that the current orientation becomes inaccessible at the next point but there

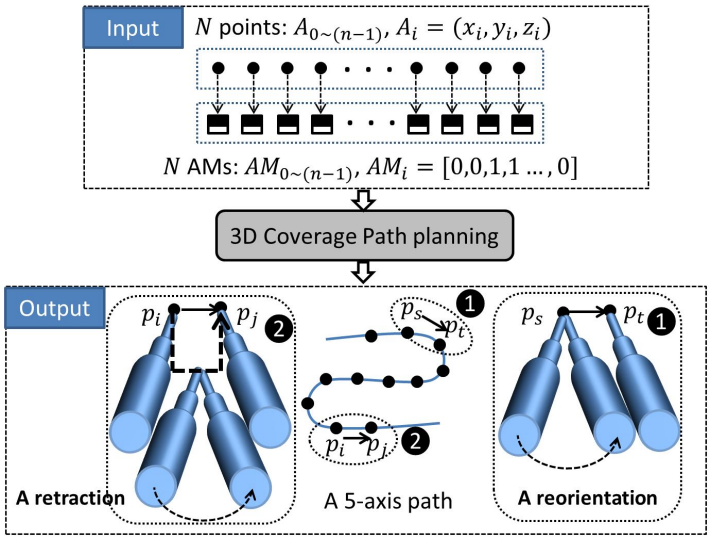


Figure 4.3: The input of our problem is the N points on the surface of the target 3D object and the corresponding N AM. The output is a 5-axis $(x, y, z, \theta, \varphi)$ path, where (x, y, z) is the pivot point on the end of the robot and (θ, φ) is an orientation the robot is placed.

exists another orientation accessible on both the next and the current point, we demand a *reorientation*, which is a process that reorients the robot to another accessible orientation and then move, shown in the bottom right of Figure 4.3.

- 3) Given that the current orientation becomes inaccessible at the next point and there is no accessible orientation shared between the two points, we demand a *retraction*, which is a process that pulls back the robot to a place far away from the object, reorients to an accessible orientation, and resumes, shown in the bottom left of Figure 4.3. Note that the point needs to stay far away from the object so that any orientation is collision-free. This operation has three steps for a smooth transition: pull back, reorient and then push in.

4.3.4 Problem statement

Our problem is to construct an efficient path for a robot that can physically touch all points on the surface of a given 3D object (e.g., the head) as a complete coverage, and guarantee no collisions. The robot is a 5-axis CNC milling tool. Figure 4.3 shows the input and the

output. The focus of this paper is on formulating and solving the optimization problem; the task of determining which orientations are safe at a given pivot point (i.e., calculating the AM) appears in our previous work [105].

4.3.5 Define the cost function

We define the cost as the machining time, so that an efficient path can cover an object fast. The machining time is assumed to be proportional to the path length. The time of applying the two operations varies significantly, which depends on specifics of the robot, such as how to move and how to reorient. We specify a range as shown in Section 4.5 for their cost by normalizing it with the path length. For example, if the cost $(c1, c2)$ is (10, 50)mm, it means that each reorientation consumes the time of the robot moving 10mm and each retraction consumes the time of the robot moving 50mm. In the equation below, num_1, num_2 denote the number of reorientation and the number of retraction respectively. Therefore, the optimization goal is to minimize the cost to achieve a fast coverage.

$$Machine\_Time = \alpha * [Path\_Length + Cost(constraints)]$$

$$Machine\_Time = \alpha * (Path\_Length + c_1 * num_1 + c_2 * num_2)$$

We do not consider the approaches of the tool path planning problem as the candidate methods because of its limitation of determining generic collisions. Since we already have the information about the accessibility of the orientation as the input, we consider coverage path planning as our candidate ways as illustrated in Figure 4.4.

Most approaches have two stages: running a segmentation algorithm and a coverage algorithm. For the segmentation algorithm, the K-guard sampling method randomly samples K guards and chooses the one that covers the most points [85]. Iterative sampling chooses points that shorten the path in an iterative way [84]. Greedy sampling finds the points that cover the most points until all points are covered [22]. For the coverage algorithm, LKH

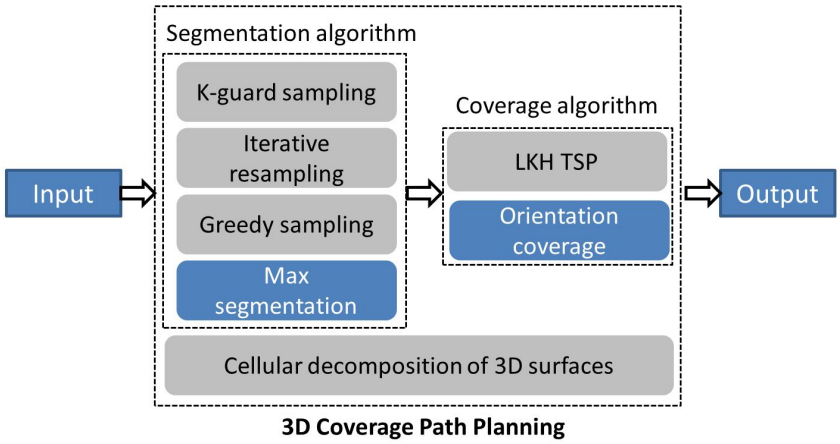


Figure 4.4: Candidate ways to solve the 3D path planning problem. Our proposed methods are highlighted in blue color.

TSP constructs a tour covering all the point for the purpose of the path length reduction [85, 84, 8, 86, 93, 22]. Three-dimensional cellular decomposition intersects a slice plane with the object surface to form a loop around the target object and then traces the loop to the next slice plane, until a full coverage is done [91, 92].

The baseline algorithm used in this paper is the random sampling-based coverage approaches that has the two stages. More specifically, we compare our approach with two algorithms: one is K-guard sampling with LKH TSP, and the other is greedy sampling with LKH TSP.

4.4 Proposed Approach

Our approach has two components: max segmentation used to reduce the number of segmentation and orientation coverage used to lower the cost of the retraction and the reorientation by sacrificing the path length.

4.4.1 Max segmentation

Motivated by the high cost of the two operations, our segmentation algorithm aims to minimize the total number of viewpoints. We define a set as a series of connected points that can

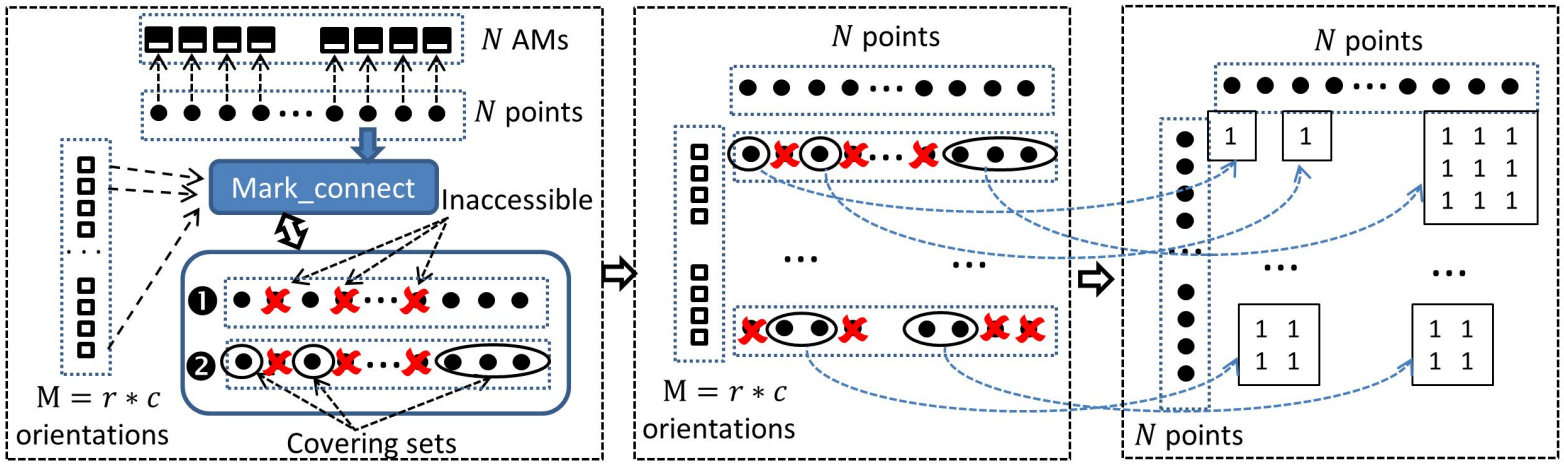


Figure 4.5: Construct a graph $N * N$ matrix embedding all candidate covering sets to cover all N points on the surface of the 3D object, where M orientations correspond to the $r * c$ orientation points in AM.

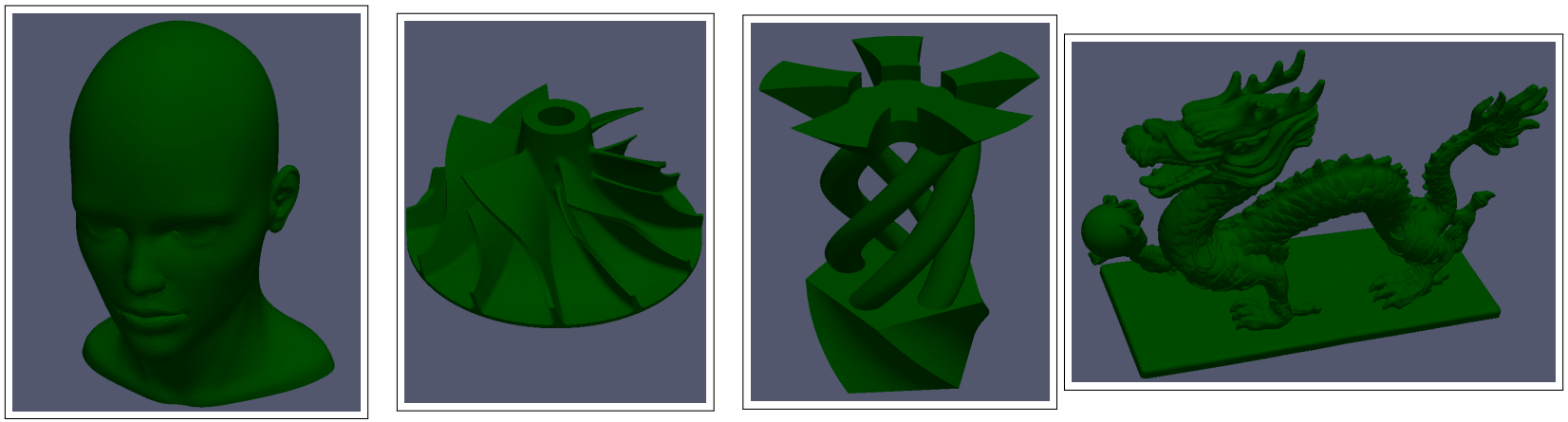


Figure 4.6: Our four CAD object models from left to right are: Head, Turbine, Candle holder, Dragon.

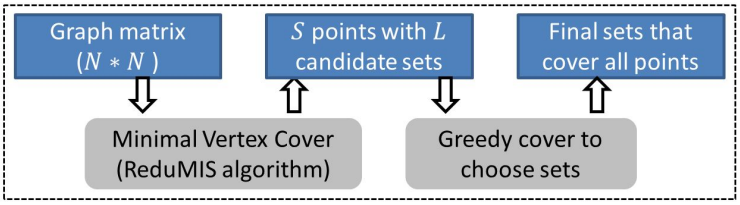


Figure 4.7: Our max segmentation approach to choose a small number of covering sets to achieve a complete coverage.

be covered by a single orientation. An orientation on a point is associated with a set, where the robot can cover all the points in this set under this orientation. Our max segmentation algorithm divides all points into sets.

In the process of choosing the sets, we apply the MVC algorithm [111] to reduce the number of sets while achieving a complete coverage, instead of sampling the point or sampling the orientation. Figure 4.5 illustrates how to construct a graph matrix, where the vertices are the N points and the edges represent the candidate sets. Firstly, a function called "Mark_connect" is used to calculate all candidate sets. Given an orientation, all the points and AM, it has two steps: mark the accessible points under the specified orientation and form a set if the points are connected as neighbors. The outputs are the candidate sets as shown in the middle of Figure 4.5. Each orientation has a series of sets that can be covered under the orientation, and each point has a series of sets that can cover the point. Lastly, a graph matrix is created, where we let the points in the candidate sets all-connected. Note that each set has a single orientation that covers the points within the set.

Now that we have all the candidate sets embedded in the graph matrix, we need to choose the sets to cover all the points. Figure 4.7 shows the steps of our max segmentation algorithm. The first step is to apply ReduMIS algorithm [111] to choose S points with a complete coverage. Note that each point is associated with multiple sets. We are sure that the S points' L candidates sets can achieve a complete coverage. The next is to do a greedy coverage that at every iteration, a set that covers the most points is chosen from the L sets until all points are covered.

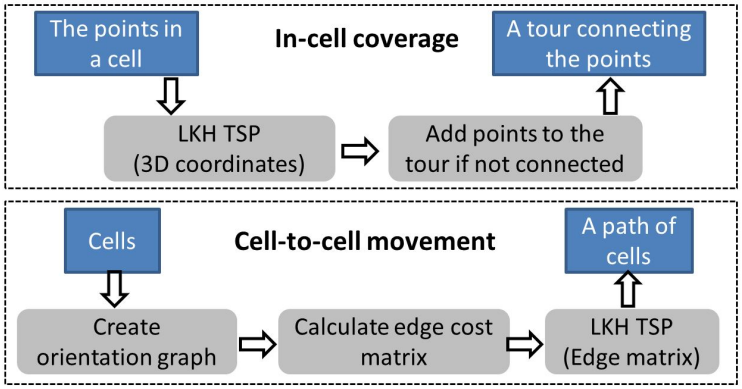


Figure 4.8: Steps of in-cell coverage and cell-to-cell movement.

Compared with the random sampling segmentation algorithms, the benefits of our max segmentation come from two points: S is much smaller than N and L is much smaller than the total number of the candidate sets. Note that if the robot does not need to set the orientation in some applications, the step of doing the greedy coverage can be removed.

4.4.2 Orientation-based coverage

With the covering sets generated from the segmentation algorithm, our coverage algorithm needs to construct a path to cover the points in sets. The algorithm needs to answer two questions: how to cover the points in sets and how to move between sets. We create a graph for the cost reduction, called orientation graph, where each set corresponds to a vertex. For simplicity, the set is called cell from now on.

To answer the two questions, our coverage algorithm has two parts respectively: in-cell coverage and cell-to-cell movement. Figure 4.8 shows the steps of the two parts. On the first part, we use LKH TSP to construct a tour to reduce the path length of covering each cell. To avoid redundant coverage, each point on the object surface is assigned to one cell, which might lead to the points disconnected in a cell. Then the next step is to add points between two neighboring points in the tour if they are not connected. On the other part, we create orientation graph and use the cost of an edge to represent the cost of a movement from one cell to another. After all the costs of all possible edges are calculated, we apply

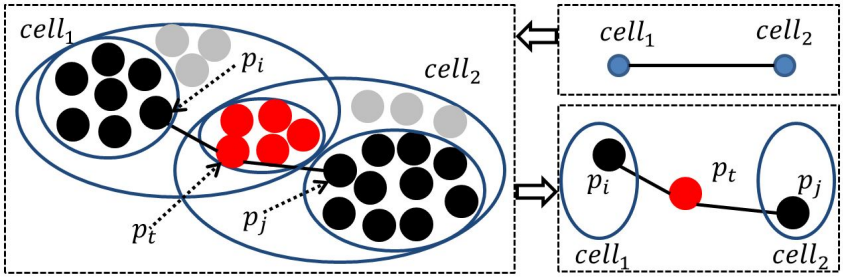


Figure 4.9: Calculate the cost of an edge by mapping the edge to three points. Red points are the intersected points. P_i and P_j are the mapped two ends of the edge. P_t is middle point where the robot does reorientation.

LKH TSP to obtain an order of cells.

Figure 4.9 illustrates how to calculate the cost of an edge by mapping the edge to three points. We add an edge between two cells to the orientation graph as a possible movement, only when the two cells have intersected points. Among all the intersected points, the one that has the shortest distance to exit one cell and to enter the other cell is selected as the middle point between them. The middle point is the place the robot does the reorientation and move to another cell.

Figure 4.10 shows a real example of an edge connecting two cells in the head object. Due to the requirement of avoiding redundant coverage, each cell has two types of points: the actual points to cover in the current cell and the points already assigned to other cells. The middle point is chosen from the intersection of both types of points while the two ends points are from the actual points. The cost of the edge includes two parts: the sum of the two geodesic distances and the cost of a reorientation.

To further reduce the cost of moving between cells, we apply Floyd-warshall algorithm to calculate the edge cost matrix. Different from the normal calculation of the shortest distance in a graph, when trying to insert a new cell between two cells, the edge cost needs to add the cost of a path passing through the new cell. Lastly LKH TSP is employed to produce an order of cells that has a small cost of moving between cells.

As the order of the cells is determined, we know which edges will be used for the

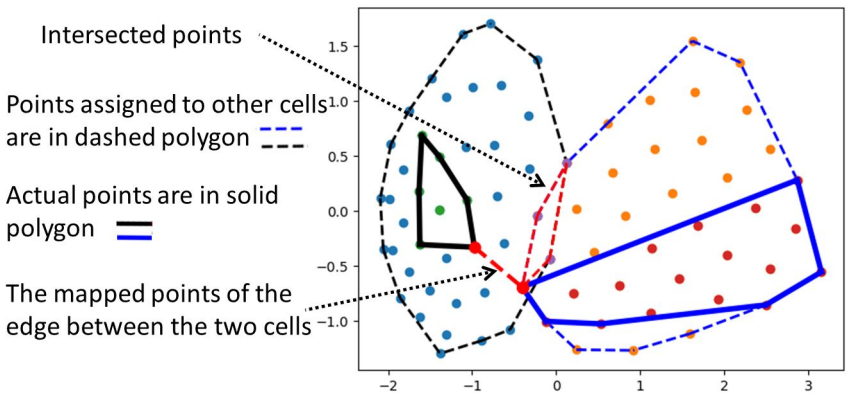


Figure 4.10: An actual example of mapping an edge to points in the head CAD benchmark. All the points are projected to 2D using principal component analysis (PCA). The middle point and one end happen to be located at one point.

coverage. Through the mapping from the edge to the points, the two end points become the exit and the entrance points. Combining with the path of covering each cell, we can form a 5-axis path.

4.5 Experimental Evaluation

The experimental validation of our approach mainly answers the following three questions:

- (1) how our max segmentation algorithm performs on the number of sets, compared with other sampling-based segmentation algorithms?
- (2) how our orientation coverage algorithm behaves on the cost of the two operations? on the three types of costs: the path length, the reorientation and the retraction?
- (3) how our proposed method performs on the total cost of 5-axis paths? under different configurations, compared with other candidate algorithms?

Implementation of candidate algorithms. We evaluate the candidate algorithms shown in Figure 4.4. For the segmentation algorithm, K-guard sampling and greedy sampling are implemented for comparison. Because the key idea of the iterative resampling [84] is to reduce the distance among all sets with no need to cover each point in the sets, it does

not work for our scenario. Thus the iterative resampling is not presented. In our problem setting, the sampling metric has two dimensions as shown in the middle part of Figure 4.5: the N points and the M orientations. We implemented K-guard on the orientations and greedy sampling on the points. K-guard algorithm samples K sets as candidate sets at each iteration, and then choose the one that covers the most points, until all points are covered. Greedy algorithm samples a point from the uncovered points and choose the orientation that cover the maximal number of points.

For the coverage algorithm, LKH TSP is our baseline. We also use LKH TSP for the in-cell coverage. The main comparison is on how to construct a path for the cell-to-cell movement. LKH TSP is employed to calculate a order of cells to reduce the path length. Firstly we calculate the mean coordinates of the points within a cell as the center to represent the cell. Secondly, LKH TSP is employed to generate a short tour passing through all the cells as the cell order. Lastly when combining the in-cell coverage with the cell order, the point in the next cell that is closest to the exit point in the current cell is considered as the next entry point.

For our coverage baseline algorithm, we focus on reducing the path length representing the 3D cellular decomposition approach [91, 92] as well. Regarding lowering the cost of the reorientation and retraction, we always choose an orientation that covers the most points for a given path.

Benchmarks and configurations. We use four CAD benchmark models for our evaluation as presented in Figure 4.6. The AM is generated form SculptPrint, a computer-aided manufacturing (CAM) application for producing CNC tool paths [69, 105], with a tool composed of 4 cylinders, with varying radii (0.79, 1.59, 25, 31.5)mm and heights(2.28, 5.08, 78, 22)mm. For the K-guard sampling algorithm, the K value is ranged from 10 to 40. Both sampling algorithms are run 20 times and report their mean and their std. Depending on the specific application and the robot, the reorientation and retraction cost may vary significantly. Thus we vary the cost of a reorientation and a retraction by normalizing them

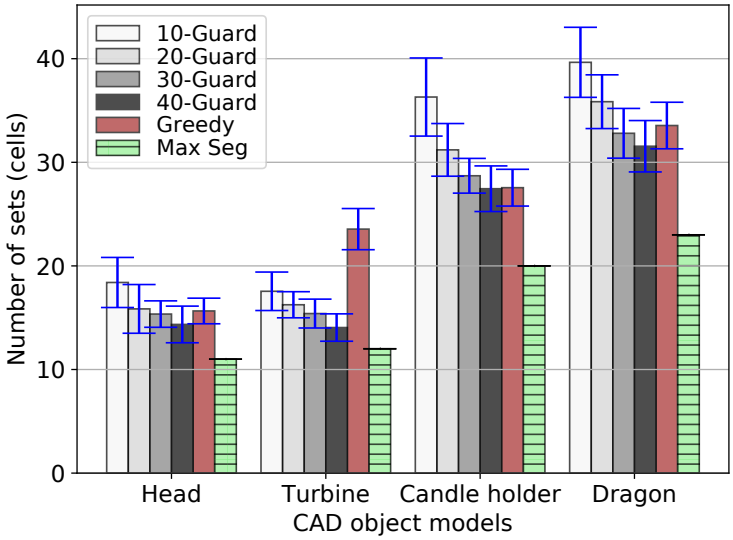


Figure 4.11: Using various object models to evaluate segmentation algorithms.

into the path length as a range from (10, 50)mm to (50, 250)mm. For simplicity, "Ret" and "Ort" are short for the cost of retractions and reorientations. "GreTSP", "GuaTSP", "MaxTSP" and "MaxOrt" are the four candidate algorithms, representing (greedy sampling + TSP), (K-guard sampling + TSP), (max segmentation + TSP) and (max segmentation + orientation coverage).

4.5.1 Number of sets in segmentation algorithms

We report the total number of sets (cells) that cover the surface of the four objects. Figure 4.11 shows the results. We can see that the number of cells required for the K-guard sampling decreases as the K value increases. Comparing the 40-guard sampling with the greedy sampling, they have a roughly same number of cells on the candle holder object, but on the turbine object, the greedy sampling behaves even worse than 10-guard sampling. It is uncertain which one is better because the random sampling is unstable. Our max segmentation gains a smaller number of cells on all four objects. Overall we can reduce the number of cells by 24.5% and 34.2% on average compared with 40-guard sampling and greedy-sampling respectively.

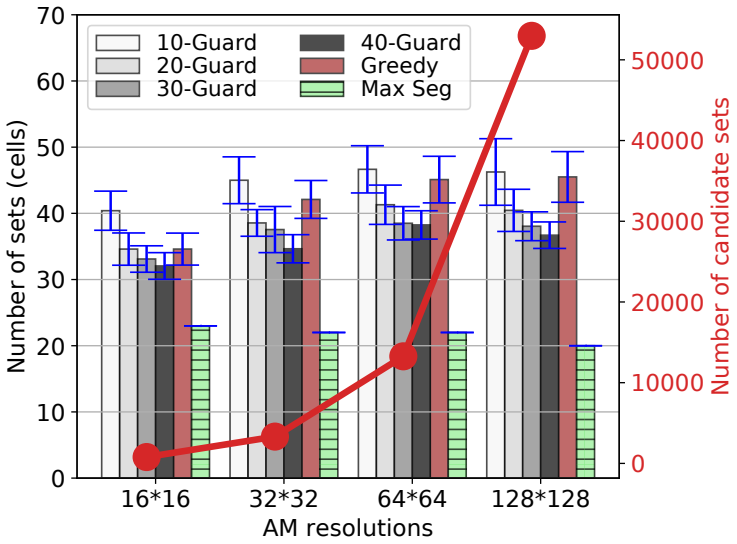


Figure 4.12: Segmentation algorithm calculates the number of sets under various AM resolutions with the dragon object model.

We increase AM resolutions leading to the growth of the number of accessible orientations on each point. Figure 4.12 shows the results on the number of sets and the number of candidate sets. The number of candidate sets does not grow exponentially as the AM resolution. This is because as the number of orientations raises, it is more likely that multiple orientations correspond to a set as a connected component. From the resolution 16^2 to 64^2, both the K-guard sampling and the greedy sampling gain an increasing number of sets, because it becomes less likely that the algorithm chooses the "correct" set as the total number of sets grows. On the resolution 128^2, the number of cells does not increase because more "correct" sets are added for sampling. In contrast, our max segmentation is not negatively influenced by the AM resolution, but the number of sets decreases slightly. On average, it reduces the number of sets by 38.6% and 48.0% than 40-guard sampling and greedy-sampling respectively.

4.5.2 Coverage algorithms

To evaluate how much our orientation coverage can contribute, we fix the max segmentation on the first step, but compare it again LKH TSP algorithm. Figure 4.15 shows the results

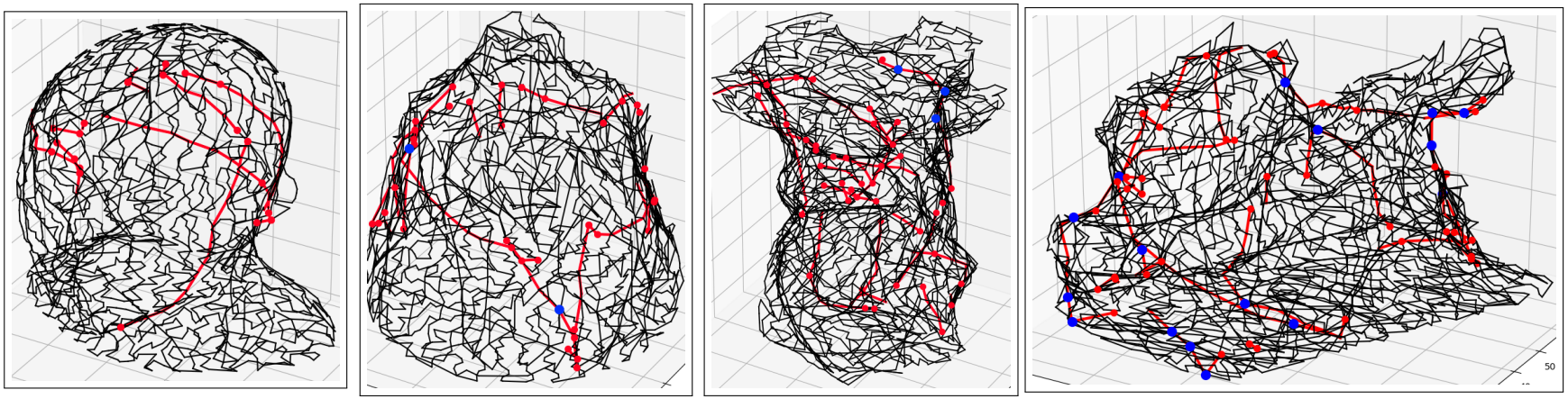


Figure 4.13: Path generated by "GreTSP". The black lines represent the path of covering cells. The red lines represent the path of cell-to-cell movement. The red point denotes the point that requires a reorientation. The blue point means a retraction.

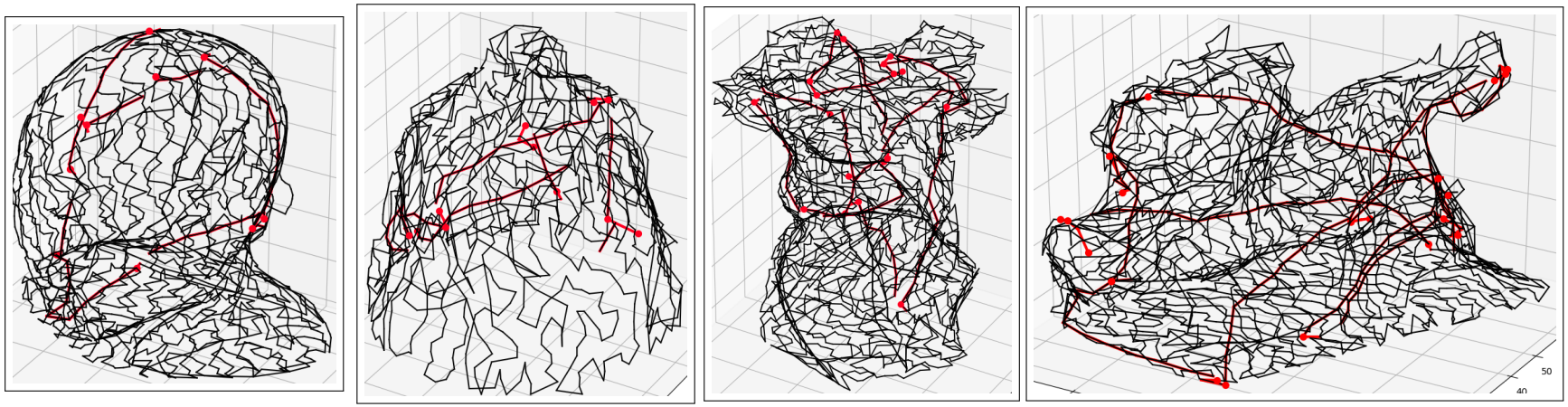


Figure 4.14: Path generated by our "MaxOrt", where black lines represent the path of covering cells, the red lines represent the path of cell-to-cell movement. There is no retraction and only reorientation.

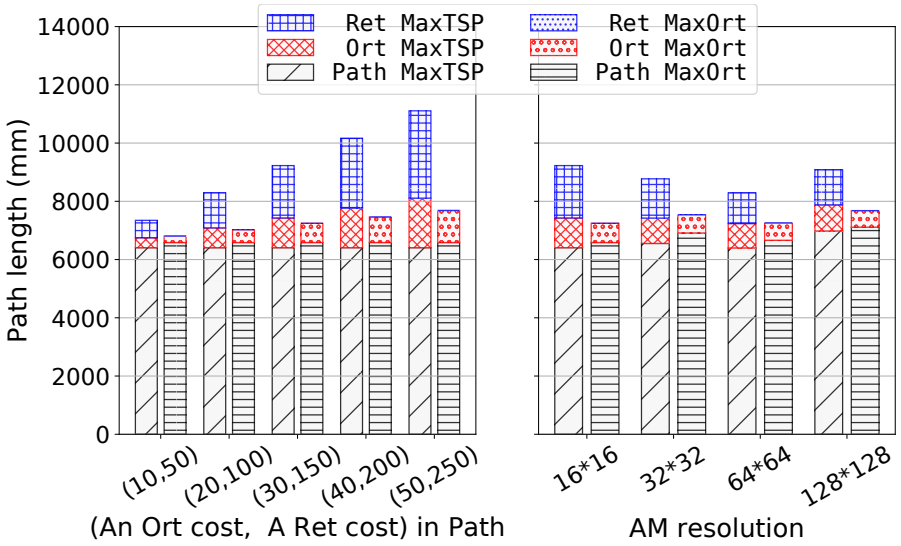


Figure 4.15: Orientation coverage against LKH TSP with the dragon object model. On the left side, the cost of reorientation and retraction is converted into the path length. On the right side, the two costs are fixed to (30, 150).

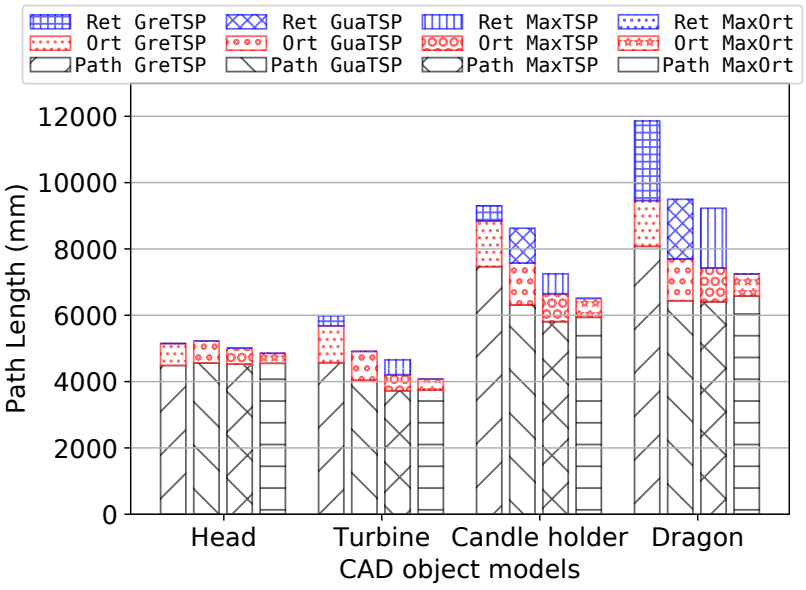


Figure 4.16: 5-axis path cost of the four candidate algorithms on the four objects. The cost of a reorientation and a retraction is fixed to (30, 150)mm in path length.

of the two algorithms. Because our orientation coverage does not have any retractions, its total cost increases slightly, while the "MaxTSP" algorithm has a sharp rise as the cost of the two operations increases. On average, our orientation coverage reduces the total cost by 21.5%.

When varying the AM resolutions, our algorithm always has a smaller number of reorientation. while "MaxTSP" has an unstable cost of both operations, because a shorter path may require a high cost of reorientation and retraction. On average, our algorithm only increase the path cost by 3.5%, and achieve a 15.9% cost reduction in total.

4.5.3 Cost of 5-axis path

We report the total cost of the 5-axis paths generated by the four algorithms. "MaxTSP" algorithm gains a cost reduction of 16.79% and 4.14% on the path length than "GreTSP" and "GuaTSP", because our max segmentation obtains a smaller number of cells, where a larger number of cells leads to a more-likely redundant coverage. Also because of the smaller number of cells, "MaxTSP" decreases the cost of reorientation by 37.8% and 30.4% respectively than the two sampling algorithms. Compared with "MaxTSP", our "MaxOrt" achieves a cost reduction of 67.2% on the cost of the two operations. Figure 4.13 and Figure 4.14 present the 5-axis path generated by "GreTSP" and our "MaxOrt" algorithms. For the dragon object, "MaxOrt" performs 38.9% better. The improvement goes up to 46.6% when the two costs are (50, 250)mm. Note that the generated path does not consider the influences of the cut depth. In practice, if the cut depth is large, the tool needs to move slower and vice versa.

Our method produces more efficient paths due to the lower cost of reorientation and retractions. As shown in Figure 4.14, the path does not have retraction denoted by the blue point. At this point, we can not guarantee that there would be no retraction on an arbitrary given object, which depends on the input to the problem. We discuss the extension on this in our future work. Overall, the path generated by our "MaxOrt" is 29.7% more efficient

than “GreTSP” on average.

4.6 Conclusion and Discussion

Focusing on how to avoid collisions when constructing an efficient path for a coverage, our proposed max orientation coverage has two key ideas: a segmentation algorithm called max segmentation Section 4.4.1 and a coverage algorithm called orientation coverage Section 4.4.2. Our experimental results on 4 CAD benchmarks demonstrate that our method can generate an efficient 5-axis path on both the cost of the path length and the cost of avoiding collisions, and it can improve the path efficiency by up to 46.6% Compared with a state-of-the-art baseline. For future work, we plan to extend our method to online algorithms that require producing fast and efficient paths, because our current work only targets on offline applications that allow several minutes of execution time. We will explore the value of incorporating more special operations, such as considering how to enter and exit a cell and the effects of speed-up and slow-down.

CHAPTER 5
CONCLUSION AND FUTURE DIRECTIONS

This thesis studies the problem called coverage path planning problem. Based on the types of the environment, we firstly analyze how to cover a 2D plane by considering various robot sizes, motion strategies, and multiple cost metrics. We propose an adaptive framework, called adaptive deep path (AD Path). It has two components: a corner model that represents how to enter and exit a region and a framework using deep reinforcement learning. The idea is to create an abstraction that reflects the cost of a path and can flexibly incorporate the characteristics of these configurations, and then use deep learning as a general black-box optimization tool.

The next problem we considered is path planning to cover a 3D object. It starts from a simple question, namely, how to check for a collision between a tool and an arbitrary object at high resolutions? To address the challenge of high computational costs to check collisions, we start with the idea of converting the 3D operation to 2D, using a geometric abstraction called the ICA, without losing any accuracy. A further improvement is the idea of applying efficient parallelism through threads mapping and load balancing. Although this method studies the collision between boxes and cylinders, the idea is still applicable to other collision tests, such as box and box, sphere and box.

Armed with an efficient collision detector, we need to select an accessible orientation for each point on the path to achieve a full coverage. We present an algorithm called *max orientation coverage*, which produces a collision-free path. It has two steps: a segmentation algorithm and a new coverage algorithm. The idea behind it is to minimize the number of segments (cells) using minimal vertex cover algorithm, for the purpose of reducing the cost of changing orientations. On the coverage algorithm, we guarantee that there is a smooth transition from one orientation to another.

We envision several ways to extend our three methods.

5.1 Reducing execution time of AD Path for 2D path planning

In Chapter 2, our model of AD Path targets offline scenarios, where users submit the application and can wait minutes or hours to return the path results. This allows our approach to go through the training step that lasts about several minutes. One way is to reduce the time of the training process by introducing some heuristics to guide the search in the training step for an optimal path.

For now, every time given a new environment, we need to go through the training process, which might become unacceptable for on-line approaches. Another extension is to train a generic model from a large number of environment so that the trained model can produce efficient paths for various types of environment. To do so, we need to collect representative environments as samples for the training step.

5.2 Collision detection for the continuous points on the path

In Chapter 3, we made some progress by simplifying the operation of checking collision and designing an efficient parallel algorithm. However, the result of the algorithm is the accessibility of the orientations on a single pivot point. One natural step is to compute the accessibility for a series of the pivot points. Since most objects in the environment are static, for any two neighboring points on the path, most computation used to detect collisions on the two points will be redundant. There should be some potential to be explored to reuse the computation from previous points, and thus accelerate the computations.

5.3 3D path planning

In Chapter 4, because we need to use MVC problem to minimize the number of segments, which takes about several minutes when the input has several thousands of points, a

straightforward question is how to reduce its execution time, especially when the application requires real-time response.

Another direction is to consider other constraints in CNC milling application, such as the cut depth, the cost of tool turning and the effects of speed-up, slow-down. In our current problem statement, we only consider the constraint of avoiding collision by specifying an accessible orientation as our first step. However, those constraints may also influence the machining time significantly in practical cases. For example, when the tool makes a turn, the tool needs to slow down, and the cut depth is another important factor that determines the time.

Another problem is on the cost reduction of the retractions. Although for all the data sets we have seen, our approach does not produce a retraction, we can not guarantee no retraction for any inputs. The exist of retractions depends on two data structures: the connectivity of the input graph and the accessibility between neighboring points. For example, if the graph had two disconnected component, there would be a retraction required when moving from one component to another. However, we can study how to sample the points on the surface to minimize the number of disconnected components as the input to our path planning problem.

One direction is to optimize the path of multiple steps in a milling process. In current problem setting, we concern the path efficiency in one step that covers the 3D object once. However, multiple steps are required (e.g.., machining a cylinder into a king kong target shown in Figure 1.1). For the operations that stay at close areas in various steps, they should share valid operations, so there would be some redundant computations if the paths are considered independently between steps. This direction can study the path planning of multiple steps that produce a full path for a milling process.

REFERENCES

[1] F. Yasutomi, M. Yamada, and K. Tsukamoto, "Cleaning robot control," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, IEEE, 1988, pp. 1839–1841.

[2] J. Hess, M. Beinhofer, and W. Burgard, "A probabilistic approach to high-confidence cleaning guarantees for low-cost cleaning robots," in *2014 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2014, pp. 5600–5605.

[3] P. N. Atkar, A. Greenfield, D. C. Conner, H. Choset, and A. A. Rizzi, "Uniform coverage of automotive surface patches," *The International Journal of Robotics Research*, vol. 24, no. 11, pp. 883–898, 2005.

[4] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain-covering algorithm for an auv," in *Underwater Robots*, Springer, 1996, pp. 17–45.

[5] H Najjaran and N Kircanski, "Path planning for a terrain scanner robot," in *INTERNATIONAL SYMPOSIUM ON ROBOTICS*, unknown, vol. 31, 2000, pp. 132–137.

[6] E. U. Acar, H. Choset, Y. Zhang, and M. Schervish, "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods," *The International journal of robotics research*, vol. 22, no. 7-8, pp. 441–466, 2003.

[7] M. Bosse, N. Nourani-Vatani, and J. Roberts, "Coverage algorithms for an under-actuated car-like vehicle in an uncertain environment," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, 2007, pp. 698–703.

[8] B. J. Englot and F. S. Hover, "Sampling-based coverage path planning for inspection of complex structures," in *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[9] D. Konobrytskyi, M. M. Hossain, T. M. Tucker, J. A. Tarbutton, and T. R. Kurfess, "5-axis tool path planning based on highly parallel discrete volumetric geometry representation: Part i contact point generation," *Computer-Aided Design and Applications*, vol. 15, no. 1, pp. 76–89, 2018.

[10] J. A. Carter, T. M. Tucker, and T. R. Kurfess, "3-axis cnc path planning using depth buffer and fragment shader," *Computer-Aided Design and Applications*, vol. 5, no. 5, pp. 612–621, 2008.

[11] J. A. Tarbutton, T. R. Kurfess, and T. M. Tucker, "Graphics based path planning for multi-axis machine tools," *Computer-Aided Design and Applications*, vol. 7, no. 6, pp. 835–845, 2010.

[12] I. A. Hameed, D. Bochtis, and C. A. Sørensen, "An optimized field coverage planning approach for navigation of agricultural robots in fields involving obstacle areas," *International journal of advanced robotic systems*, vol. 10, no. 5, p. 231, 2013.

[13] L. Lin and M. A. Goodrich, "Uav intelligent path planning for wilderness search and rescue," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 709–714.

[14] A. Xu, C. Viriyasuthee, and I. Rekleitis, "Efficient complete coverage of a known arbitrary environment with applications to aerial operations," *Autonomous Robots*, vol. 36, no. 4, pp. 365–381, 2014.

[15] A. Xu, C. Viriyasuthee, and I. Rekleitis, "Optimal complete terrain coverage using an unmanned aerial vehicle," in *2011 IEEE International conference on robotics and automation*, IEEE, 2011, pp. 2513–2519.

[16] A. Kolb, L. Latta, and C. Rezk-Salama, "Hardware-based simulation and collision detection for large particle systems," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM, 2004, pp. 123–131.

[17] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[18] I. A. Şucan, M. Kalakrishnan, and S. Chitta, "Combining planning techniques for manipulation using realtime perception," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 2895–2901.

[19] P. Du, J.-Y. Zhao, W.-B. Pan, and Y.-G. Wang, "Gpu accelerated real-time collision handling in virtual disassembly," *Journal of Computer Science and Technology*, vol. 30, no. 3, pp. 511–518, 2015.

[20] E. Galceran and M. Carreras, "Efficient seabed coverage path planning for asvs and auvs," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 88–93.

[21] T.-S. Lee, J.-S. Choi, J.-H. Lee, and B.-H. Lee, "3-d terrain covering and map building algorithm for an auv," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 4420–4425.

[22] N. Palomeras, N. Hurtós, M. Carreras, and P. Ridao, "Autonomous mapping of underwater 3-d structures: From view planning to execution," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1965–1971, 2018.

[23] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[24] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, no. 3, pp. 247–253, 2000.

[25] R. Tinós, K. Helsgaun, and D. Whitley, "Efficient recombination in the lin-kernighan-helsgaun traveling salesman heuristic," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2018, pp. 95–107.

[26] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 1994.

[27] H. A. Eiselt, M. Gendreau, and G. Laporte, "Arc routing problems, part i: The chinese postman problem," *Operations Research*, vol. 43, no. 2, pp. 231–242, 1995.

[28] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1-2, pp. 25–50, 2000.

[29] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[30] B. Donald, K. Lynch, and D. Rus, *Algorithmic and Computational Robotics: New Directions 2000 WAFR*. CRC Press, 2001.

[31] T. C. Shermer, "Recent results in art galleries (geometry)," *Proceedings of the IEEE*, vol. 80, no. 9, pp. 1384–1399, 1992.

[32] W.-P. Chin and S. Ntafos, "Shortest watchman routes in simple polygons," *Discrete & Computational Geometry*, vol. 6, no. 1, pp. 9–31, 1991.

[33] F. Li and R. Klette, "An approximate algorithm for solving the watchman route problem," in *International Workshop on Robot Vision*, Springer, 2008, pp. 189–206.

[34] D. S. Johnson and C. H. Papadimitriou, *Computational complexity and the traveling salesman problem*. Mass. Inst. of Technology, Laboratory for Computer Science, 1981.

[35] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 1846–1852.

[36] S. Bochkarev and S. L. Smith, "On minimizing turns in robot coverage path planning," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, 2016, pp. 1237–1242.

[37] R. Lynn, D. Contis, M. Hossain, N. Huang, T. Tucker, and T. Kurfess, "Voxel model surface offsetting for computer-aided manufacturing using virtualized high-performance computing," *Journal of Manufacturing Systems*, vol. 43, pp. 296–304, 2017.

[38] E. U. Acar and H. Choset, "Sensor-based coverage of unknown environments: Incremental construction of morse decompositions," *The International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002.

[39] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *The international journal of robotics research*, vol. 21, no. 4, pp. 331–344, 2002.

[40] D. Meger, I. Rekleitis, and G. Dudek, "Heuristic search planning to reduce exploration uncertainty," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2008, pp. 3392–3399.

[41] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot collaboration for robust exploration," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 7–40, 2001.

[42] H. Choset and J. Burdick, "Sensor based planning. ii. incremental construction of the generalized voronoi graph," in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, IEEE, vol. 2, 1995, pp. 1643–1648.

[43] J. Edmonds and E. L. Johnson, "Matching, euler tours and the chinese postman," *Mathematical programming*, vol. 5, no. 1, pp. 88–124, 1973.

[44] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[45] G. Reinelt, *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, 1994.

[46] L. Wei, Z. Zhang, D. Zhang, and S. C. Leung, "A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints," *European Journal of Operational Research*, vol. 265, no. 3, pp. 843–859, 2018.

[47] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[48] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in neural information processing systems*, 2016, pp. 3981–3989.

[49] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.

[50] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, p. 471, 2016.

[51] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2692–2700.

[52] R. Mannadiar and I. Rekleitis, "Optimal coverage of a known arbitrary environment," in *2010 IEEE International conference on robotics and automation*, IEEE, 2010, pp. 5525–5530.

[53] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*, Springer, 2005, pp. 317–328.

[54] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[55] H. Dai, *Graph combination optimization*, `https://github.com/Hanjun-Dai/graph_comb_opt`, 2017.

[56] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," *arXiv preprint arXiv:1603.05629*, 2016.

[57] D. Konobrytskyi, "Automated CNC tool path planning and machining simulation on highly parallel computing architectures," PhD thesis, Clemson University, 2013.

[58] S Ding, M. Mannan, and A. N. Poo, "Oriented bounding box and octree based global interference detection in 5-axis machining of free-form surfaces," *Computer-Aided Design*, vol. 36, no. 13, pp. 1281–1294, 2004.

[59] J. Tarbutton, T. R. Kurfess, T. Tucker, and D. Konobrytskyi, "Gouge-free voxel-based machining for parallel processors," *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 9-12, pp. 1941–1953, 2013.

[60] M. Tang, D. Manocha, S.-E. Yoon, P. Du, J.-P. Heo, and R.-F. Tong, "Volccd: Fast continuous collision culling between deforming volume meshes," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 5, p. 111, 2011.

[61] M. Tang, D. Manocha, J. Lin, and R. Tong, "Collision-streams: Fast gpu-based collision detection for deformable models," in *Symposium on interactive 3D graphics and games*, ACM, 2011, pp. 63–70.

[62] N. K. Govindaraju, S. Redon, M. C. Lin, and D. Manocha, "Cullide: Interactive collision detection between complex models in large environments using graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, 2003, pp. 25–32.

[63] P. Du, E. S. Liu, and T. Suzumura, "Parallel continuous collision detection for high-performance gpu cluster," in *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, 2017, p. 4.

[64] X. Zhang and Y. J. Kim, "Scalable collision detection using p-partition fronts on many-core processors," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 3, pp. 447–456, 2014.

[65] O. S. Lawlor and L. V. Kalée, "A voxel-based parallel collision detection algorithm," in *Proceedings of the 16th international conference on Supercomputing*, ACM, 2002, pp. 285–293.

[66] D. Kim, J.-P. Heo, J. Huh, J. Kim, and S.-e. Yoon, "Hpccd: Hybrid parallel continuous collision detection using cpus and gpus," in *Computer Graphics Forum*, Wiley Online Library, vol. 28, 2009, pp. 1791–1800.

[67] C. Lauterbach, Q. Mo, and D. Manocha, "Gproximity: Hierarchical gpu-based operations for collision and distance queries," in *Computer Graphics Forum*, Wiley Online Library, vol. 29, 2010, pp. 419–428.

[68] R. Lynn, M. Dinar, N. Huang, J. Collins, J. Yu, C. Greer, T. Tucker, and T. Kurfess, "Direct digital subtractive manufacturing of a functional assembly using voxel-based models," *Journal of Manufacturing Science and Engineering*, vol. 140, no. 2, p. 021 006, 2018.

[69] T. I. Inc., *Sculptprint*, `http://www.sculptprint.com`, Accessed: 2018-09-30.

[70] S. Kockara, T. Halic, K Iqbal, C. Bayrak, and R. Rowe, "Collision detection: A survey," in *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, IEEE, 2007, pp. 4046–4051.

[71] C. D. Toth, J. O'Rourke, and J. E. Goodman, *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.

[72] C. Ericson, *Real-time collision detection*. CRC Press, 2004.

[73] D. Knott, "Cinder: Collision and interference detection in real time using graphics hardware," PhD thesis, University of British Columbia, 2003.

[74] S.-K. Wong, W.-C. Lin, C.-H. Hung, Y.-J. Huang, and S.-Y. Lii, "Radial view based culling for continuous self-collision detection of skeletal models," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 114, 2013.

[75] I. J. Palmer and R. L. Grimsdale, "Collision detection for animation using sphere-trees," in *Computer Graphics Forum*, Wiley Online Library, vol. 14, 1995, pp. 105–116.

[76] P. M. Hubbard, "Approximating polyhedra with spheres for time-critical collision detection," *ACM Transactions on Graphics (TOG)*, vol. 15, no. 3, pp. 179–210, 1996.

[77] J.-W. Chang, W. Wang, and M.-S. Kim, "Efficient collision detection using a dual obb-sphere bounding volume hierarchy," *Computer-Aided Design*, vol. 42, no. 1, pp. 50–57, 2010.

[78] D.-J. Kim, L. J. Guibas, and S.-Y. Shin, "Fast collision detection among multiple moving spheres," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 3, pp. 230–242, 1998.

[79] A. Sud, N. Govindaraju, R. Gayle, I. Kabul, and D. Manocha, "Fast proximity computation among deformable models using discrete voronoi diagrams," *ACM Transactions on Graphics (TOG)*, vol. 25, no. 3, pp. 1144–1153, 2006.

[80] M. Tang, D. Manocha, and R. Tong, "Multi-core collision detection between deformable models," in *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, ACM, 2009, pp. 355–360.

[81] J. Pan, S. Chitta, and D. Manocha, "Probabilistic collision detection between noisy point clouds using robust classification," in *Robotics Research*, Springer, 2017, pp. 77–94.

[82] L. Zhiwei, S. Hongyao, G. Wenfeng, and F. Jianzhong, "Approximate tool posture collision-free area generation for five-axis cnc finishing process using admissible area interpolation," *The International Journal of Advanced Manufacturing Technology*, vol. 62, no. 9-12, pp. 1191–1203, 2012.

[83] K. Alexis, G. Darivianakis, M. Burri, and R. Siegwart, "Aerial robotic contact-based inspection: Planning and control," *Autonomous Robots*, vol. 40, no. 4, pp. 631–655, 2016.

[84] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 6423–6430.

[85] T. Danner and L. E. Kavraki, "Randomized planning for short inspection paths," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, IEEE, vol. 2, 2000, pp. 971–976.

[86] B. Englot and F. Hover, "Planning complex inspection tasks using redundant roadmaps," in *Robotics Research*, Springer, 2017, pp. 327–343.

[87] R. Almadhoun, T. Taha, L. Seneviratne, J. Dias, and G. Cai, "A survey on inspecting structures using robotic systems," *International Journal of Advanced Robotic Systems*, vol. 13, no. 6, p. 1 729 881 416 663 664, 2016.

[88] Y. Gabriely and E. Rimon, "Spiral-stc: An on-line coverage algorithm of grid environments by a mobile robot," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 1, 2002, pp. 954–960.

[89] C. Luo, S. X. Yang, D. A. Stacey, and J. C. Jofriet, "A solution to vicinity problem of obstacles in complete coverage path planning," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 1, 2002, pp. 612–617.

[90] E. U. Acar, H. Choset, and J. Y. Lee, "Sensor-based coverage with extended range detectors," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 189–198, 2006.

[91] P. N. Atkar, H. Choset, A. A. Rizzi, and E. U. Acar, "Exact cellular decomposition of closed orientable surfaces embedded in/spl rfr//sup 3," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, IEEE, vol. 1, 2001, pp. 699–704.

[92] P. N. Atkar, A. Greenfield, D. C. Conner, H. Choset, and A. A. Rizzi, "Hierarchical segmentation of surfaces embedded in r3 for auto-body painting," in *Proceedings of the 2005 IEEE international conference on robotics and automation*, IEEE, 2005, pp. 572–577.

[93] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon" next-best-view" planner for 3d exploration," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 1462–1468.

[94] G. Papadopoulos, H. Kurniawati, and N. M. Patrikalakis, "Asymptotically optimal inspection planning using systems with differential constraints," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4126–4133.

[95] E. Galceran and M. Carreras, "Planning coverage paths on bathymetric maps for in-detail inspection of the ocean floor," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4159–4164.

[96] J. O'rourke, *Art gallery theorems and algorithms*. Oxford University Press Oxford, 1987, vol. 57.

[97] H. González-Banos, "A randomized art-gallery algorithm for sensor placement," in *Proceedings of the seventeenth annual symposium on Computational geometry*, ACM, 2001, pp. 232–240.

[98] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[99] I. Cho, K. Lee, and J. Kim, "Generation of collision-free cutter location data in five-axis milling using the potential energy method," *The International Journal of Advanced Manufacturing Technology*, vol. 13, no. 8, pp. 523–529, 1997.

[100] C.-S. Jun, K. Cha, and Y.-S. Lee, "Optimizing tool orientations for 5-axis machining by configuration-space search method," *Computer-Aided Design*, vol. 35, no. 6, pp. 549–566, 2003.

[101] C.-H. Chu and J.-T. Chen, "Tool path planning for five-axis flank milling with developable surface approximation," *The International Journal of Advanced Manufacturing Technology*, vol. 29, no. 7-8, p. 707, 2006.

[102] Y.-W. Hsueh, M.-H. Hsueh, and H.-C. Lien, "Automatic selection of cutter orientation for preventing the collision problem on a five-axis machining," *The International Journal of Advanced Manufacturing Technology*, vol. 32, no. 1-2, pp. 66–77, 2007.

[103] N. Wang and K. Tang, "Automatic generation of gouge-free and angular-velocity-compliant five-axis toolpath," *Computer-Aided Design*, vol. 39, no. 10, pp. 841–852, 2007.

[104] C.-H. Chu, W.-N. Huang, and Y.-W. Li, "An integrated framework of tool path planning in 5-axis machining of centrifugal impeller with split blades," *Journal of Intelligent Manufacturing*, vol. 23, no. 3, pp. 687–698, 2012.

[105] X. Chen, D. Konobrytskyi, T. M. Tucker, T. R. Kurfess, and R. W. Vuduc, "Faster parallel collision detection at high resolution for cnc milling applications," in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[106] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[107] K. Jolly, R. S. Kumar, and R Vijayakumar, "A bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits," *Robotics and Autonomous Systems*, vol. 57, no. 1, pp. 23–33, 2009.

[108] H. Liu, X. Lai, and W. Wu, "Time-optimal and jerk-continuous trajectory planning for robot manipulators with kinematic constraints," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 2, pp. 309–317, 2013.

[109] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Transactions on Robotics*, vol. 29, no. 1, pp. 105–117, 2012.

[110] G. Yang and V. Kapila, "Optimal path planning for unmanned air vehicles with kinematic and tactical constraints," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, IEEE, vol. 2, 2002, pp. 1301–1306.

[111] D. Hespe, C. Schulz, and D. Strash, "Scalable kernelization for maximum independent sets," *Journal of Experimental Algorithmics (JEA)*, vol. 24, no. 1, pp. 1–22, 2019.