Clemson University
TigerPrints

All Dissertations | Dissertations

August 2020

Geometric-based Optimization Algorithms for Cable Routing and Branching in Cluttered Environments

Nafiseh Masoudi
Clemson University, masoudi88@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Recommended Citation

Masoudi, Nafiseh, "Geometric-based Optimization Algorithms for Cable Routing and Branching in Cluttered Environments" (2020). *All Dissertations*. 2702.
https://tigerprints.clemson.edu/all_dissertations/2702

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

GEOMETRIC BASED OPTIMIZATION ALGORITHMS FOR CABLE ROUTING AND BRANCHING IN CLUTTERED ENVIRONMENTS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Mechanical Engineering

---

by
Nafiseh Masoudi
August 2020

Accepted July 21, 2020 by:
Dr. Georges M. Fadel, Committee Chair
Dr. Margaret M. Wiecek
Dr. Joshua D. Summers
Dr. Cameron J. Turner

Distinguished External Reviewer: Dr. Jonathan Cagan, Carnegie Mellon University

ABSTRACT

The need for designing lighter and more compact systems often leaves limited space for planning routes for the connectors that enable interactions among the system's components. Finding optimal routes for these connectors in a densely populated environment left behind at the detail design stage has been a challenging problem for decades.

A variety of deterministic as well as heuristic methods has been developed to address different instances of this problem. While the focus of the deterministic methods is primarily on the optimality of the final solution, the heuristics offer acceptable solutions, especially for such problems, in a reasonable amount of time without guaranteeing to find optimal solutions. This study is an attempt to furthering the efforts in deterministic optimization methods to tackle the routing problem in two and three dimensions by focusing on the optimality of final solutions.

The objective of this research is twofold. First, a mathematical framework is proposed for the optimization of the layout of wiring connectors in planar cluttered environments. The problem looks at finding the optimal tree network that spans multiple components to be connected with the aim of minimizing the overall length of the connectors while maximizing their common length (for maintainability and traceability of connectors). The optimization problem is formulated as a bi-objective problem and two solution methods are proposed: (1) to solve for the optimal locations of a known number of breakouts (where the connectors branch out) using mixed-binary optimization and visibility notion and (2) to find the minimum length tree that spans multiple components

of the system and generates the optimal layout using the previously-developed convex hull based routing. The computational performance of these methods in solving a variety of problems is further evaluated.

Second, the problem of finding the shortest route connecting two given nodes in a 3D cluttered environment is considered and addressed through deterministically generating a graphical representation of the collision-free space and searching for the shortest path on the found graph. The method is tested on sample workspaces with scattered convex polyhedra and its computational performance is evaluated. The work demonstrates the NP-hardness aspect of the problem which becomes quickly intractable as added components or increase in facets are considered.

DEDICATION

To my best friend, life coach, motivator, and the love of my life, Yasha, for his relentless support and encouragement during my challenges as a doctoral student both inside and outside the lab. This dissertation is also dedicated to my beloved parents Akram and Mansoor for their unconditional love and being my primary teachers.

ACKNOWLEDGMENT

The present effort would have not been possible without the kind support and input of many. I would like to take this opportunity to acknowledge some of the many people who have contributed to the success of my work.

First and foremost, I would like to extend my deepest gratitude to my advisor, Professor Georges Fadel for his unwavering support and patience throughout the progress of my research. I truly appreciate the opportunity to work with him as his final Ph.D. student and the input he provided at different stages of my research and academic growth. His insight and advice always helped me find the right direction both in research and my professional development. Professor Fadel always held me to high academic standards and mentored me to take ownership of my work and therefore become a more effective leader and communicator for which I am truly grateful.

I gratefully acknowledge the efforts and guidance of my dissertation co-advisor, Professor Margaret Wiecek. Her insight from a mathematician's point of view was instrumental to the outcome of this research.

I would like to express my sincere appreciation to my dissertation committee members, Professors Summers and Turner, and the distinguished external reviewer, Professor Jonathan Cagan, for their invaluable feedback, practical suggestions, and review of my work.

My special gratitude goes out to Dr. Summers for all his support and mentorship in helping me become a more effective researcher, instructor, and leader. He was always

willing to take the extra mile in providing advice whenever I felt stuck with a problem in the classes I teach or in my research.

Ph.D. studies have enough challenges to discourage a person from keeping up with the path. I, on the other hand, was not left alone in this journey and was fortunate to be joined by my greatest source of encouragement, my husband. Yasha wholeheartedly stood by my side with all his love at every step I took over the course of my studies, never gave up on motivating me when even I was doubting myself and provided insightful suggestions on my research. His continuous support was the key to helping me overcome the hardship of the path.

I am deeply grateful for all the love and support I have received from my parents Akram and Mansoor and my sisters Fatima and Bahar during my doctoral studies. Though far away, I always felt them by my side, and I am very fortunate to be a member of this lovely family.

I had an amazing opportunity to be a part of a collaborative and vibrant research group, Clemson Engineering Design Applications and Research (CEDAR). To me, CEDAR is a school by itself and this is made possible by the contribution of all the principal investigators (Professors Fadel, Summers, Mocko, and Turner) as well as the graduate students. I take this opportunity to recognize the contribution of my research lab mates at CEDAR (Apurva Patel, Vijay Sarthy, Shubhamkar Kulkarni, Maria Vittoria Elena, Malena Agyemang, Elizabeth Gendreau, Nicole Zero, and Nick Spivey, to name a few) along with the lab alumni: Dr. James Righter, Dr. Meng Xu, Dr. Qing Mao, Dr. Anthony Garland (I appreciate his help with the recursive programming aspects of the

research), Dr. Ivan Mata (for providing programming tips), Dr. Jingyuan Yan, and Dr. Mo Chen. The scientific discussions we had in CEDAR played an important role in constructing a problem-solving and critical thinking mentality for me.

The Mechanical Engineering Department at Clemson University is highly appreciated for the generous financial support made through Teaching Assistantships over three years during my doctoral studies. The experiences I gained from laboratory and classroom teaching had a significant contribution in making me enthusiastic about teaching and deciding to take this route as my future career.

I am also grateful to Dr. Todd Schweisinger, my supervisor in undergraduate labs, for his mentorship and support during my teaching appointment and his contribution in offering me a Graduate Teaching Fellowship to teach a junior level mechanical engineering course, Machine Design, so that I could gain more teaching experience as a Ph.D. student.

My special thanks go to all the 115 Mechanical Engineering students I had the opportunity to be their instructor during my time at Clemson University who made teaching an enjoyable aspect of my graduate studies.

Last but not the least, I acknowledge the help and support of all the ME 2220 Teaching Assistants as well as my friends at Clemson. I am also thankful to the kind staff, Ms. Kathryn Poole and Ms. Trish Nigro, at the Mechanical Engineering Department for the unparalleled service they provide for the graduate students.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One
INTRODUCTION

Finding the shortest path between two given points in an environment has been one of the classical problems in geometry. Without any obstacles to block the line of sight between the two points, the shortest path is trivially the line segment that connects the two. The problem, however, becomes challenging when the two points are not visible to each other due to the presence of obstacles blocking the direct path.

Path planning emerges in a variety of real-world problems. For example, as new features are continuously added to complex electromechanical systems such as hybrid electric vehicles, the number of their wire or cable connectors is considerably increased and the wire harnesses are becoming heavier and more complex to be designed. According to studies (for example see [1]), cabling is the third heaviest and costliest component in a car after its engine and chassis. Traditionally, cables and hoses have been routed using a manual trial-and-error approach in CAD software. The routing might be tested on prototypes [2]; however, it mainly relies on the experience of skilled engineers [3]. This makes the process time-consuming, tedious, and error-prone [4]. In addition, it could result in non-optimal solutions. Therefore, an optimal cable routing method is required to reduce their length and therefore minimize the total weight of the system.

While cable/wire routing is an example of path planning in cluttered environments and the main focus of the present study, other examples include robot motion planning, VLSI (Very Large Scale Integration) design, transportations, pipe routing (in ships and process plants), and navigation problems (e.g. vehicle routing, and UAV path planning).

Tremendous effort has been made to address different instances of this problem in both two- and three-dimensional workspaces. All these studies have one element in common; they attempt to identify a finite number of waypoints between the path start and goal points and then connect the found waypoints in series to form piecewise linear and collision-free paths. While the waypoints can be located anywhere in the collision-free space, studies show that to minimize the length of the path, the waypoints should lie on the vertices (in 2D workspaces) or the edges of (in 3D workspaces) the intersecting obstacles or at an offset from their entities (if an object must not be touched; for example to avoid sharp edges or high-temperature surfaces). In cases where the path constructed by the waypoints is not unique, the points are concatenated in a graph, which is later searched using algorithms such as Dijkstra [5] or A* [6] to find the global shortest collision-free path on the graph.

In addition to the main objective, minimization of the path length, there might be other criteria such as minimizing the number of turns in the path. This could be critical in instances where turns in the path cause complications and should be avoided. The design of the layout of chemical process plants with thousands of pipes or motion planning of robots with arbitrary geometry in densely populated environments are examples of path planning problems where the number of turns should be minimized in addition to the path length.

Besides avoiding intersection with any of the obstacles in the workspace, there could also be constraints on the path turn angle, further limiting the feasible space of the optimization problem. Other factors may also increase the complexity of the problem

including complexities in the shapes of the obstacles (e.g. holes or nonconvexities), increase in the number of intersecting obstacles, and maintaining a set clearance between the obstacles at all times. These factors, though not fundamental to this study, are occasionally discussed throughout this manuscript.

The two-dimensional class of collision-free path planning has gained extensive attention by scholars and a variety of exact as well as non-exact methods have been developed to tackle its different instances.

In our previous work, an exact geometric-based algorithm for planar routing in cluttered environments is presented. The effort is made to overcome the computational limitations of the classical visibility graph (the only exact path planning method available) by constructing candidate partial visibility graphs. The algorithm makes use of the convex hulls of the intersecting objects to construct collision-free graphs. The advantage of using the convex hulls is that it enables handling any free form obstacle in the workspace. The developed algorithm has a proven time complexity of $O(nlog(\frac{n}{f}))$ for n vertices and f intersecting objects which is a significant improvement from the classical visibility and its variants. Further, the algorithm outperforms its competitors in constructing partial visibility graphs that are used to yield the globally optimum solution in addition to being faster.

Apart from finding the shortest collision-free path between two given points, there are problems wherein multiple points in the environment are to be connected with the shortest segments. This can be seen in cable/wire routing in electromechanical systems that connect different system components or piping systems in chemical process plants and ships.

Other non-intuitive applications of multipath planning problems include but may not be limited to facility location in the presence of obstacles, layout design of wind farms, and design of transportation networks. Facility location is the problem of locating one or more new facilities in the proximity of existing facilities to minimize (or maximize) the distances between all facilities while avoiding both the placement of the new facilities inside and the travel through forbidden areas. These applications are further discussed in Chapter 3 along with the existing methods to address them.

In multipath planning problems, instead of finding the shortest path between any pair of points separately, a more efficient approach is to create the main route which can then branch out to reach different nodes; this results in a shorter network of paths. The problem, therefore, can be deemed as finding a minimum length network in the presence of obstacles.

Steiner Minimal Tree (SMT) and Minimum Spanning Tree (MST) are the most popular methods to find the minimum length tree (or network) that connects multiple points in a known network. These methods, however, do not deal with the collision avoidance constraint that is common in cluttered environments.

Building on and extending our previous work on the 2D convex hull based path planning method, the present study addresses the multipath planning problem in two-dimensional spaces by answering the question: what is the optimal layout of a network of points in the presence of obstacles?

This is a cable harness design problem; the harness trunk includes the majority of the cables together, and then they are distributed to connect to their respective nodes. The

problem is formulated as a bi-objective optimization problem whose objectives are (1) to minimize the overall length of the network and (2) to maximize the common length of the paths.

Most real-world path planning problems, however, are in 3D space. Nonetheless, moving to 3D space introduces more complexities that make most of the existing methods for 2D problems inefficient for three dimensions. Due to these complexities, most methods developed for 3D problems attempt to address the simplified version of the generic path planning problem in the presence of obstacles. The methods mainly make use of heuristic or stochastic techniques that may not be able to guarantee the attainment of the globally optimum solution. These methods are discussed in detail in Chapter Four.

Based on this background, the second part of this study investigates the possibility of developing a deterministic optimization method to find the shortest collision-free path in cluttered 3D environments. As stated by Canny [7], the shortest collision-free path in 3D cluttered environments passes through the edges of some of the obstacles if there is no direct path from start to end. This research, therefore, answers the questions of where and how the waypoints should be located on obstacle edges in a known 3D cluttered environment.

Dissertation Organization

The following provides an overview of the organization and content of each chapter in this dissertation, following Chapter One.

Chapter Two begins with a description of the 2D path planning problem and the existing methods to address it. The chapter discusses the contributions and limitations of

the related work on 2D problems along with an explanation of our previously developed convex hull based approach for 2D planar problems in a cluttered environment. The extension of these studies to 3D problems is also briefly discussed.

Chapter Three is allocated to the discussion of multipath planning problems with a particular focus on 2D problems. The review of the related work is provided and the specific application of this problem in cable harness design is explored. Finally, an optimization paradigm for cable harness layout in 2D cluttered environments is proposed and results are further presented.

Chapter Four serves as the second part of the literature review on path planning problems as it focuses on 3D cluttered environments. The discussion of the existing methods to tackle this class of path planning problems leads to the identification of gaps in methods to address 3D path planning problems which is the basis of the hypothesis in this research.

Chapter Five sets the stage for the development of the deterministic 3D path planning algorithm by stating the definitions used and assumptions made throughout this study. In addition, the workspace representation and data types/structures used to organize the geometric data of the workspace are also explained. Finally, the formulations of the graph construction and the shortest path problems are given.

Chapter Six details the geometric algorithms used to detect two types of intersections in 3D space: (1) between a line and a 3D object and (2) between two triangles. The former is used to detect intersecting objects while the latter is used as a step in constructing the collision-free graph using the information of the intersecting objects on

the way to the path goal point. Intersection detection is at the core of the 3D graph construction algorithm proposed in this research.

Chapter Seven portrays the steps in constructing the collision-free graph in a given workspace after the identification of the intersecting objects. It also presents the results of applying the developed 3D path planning algorithms on a variety of test cases. Further, a discussion of the contribution of this research in routing applications as well as the limitations of the method is provided.

Chapter Eight concludes this dissertation by summarizing the major research findings and the limitations of the proposed methods to design cable harnesses and route 3D connectors. It, additionally, presents research questions that can be explored in the future as potential extensions of the present study and research avenues that could be further explored.

Chapter Two
AN OVERVIEW OF 2D PATH PLANNING PROBLEMS AND SOLUTION METHODS

The Path Planning problem has been widely studied in the literature. Whether one is interested in solving a problem modeled on a network graph or a more real-world planning problem in 3D, the solution methods developed so far can be summarized and classified into four main categories that are explained in this chapter, though not all of them address the problem in full generality [8].

This chapter is allocated to an overview of the related work on path planning problems in 2D environments in particular. First, different approaches to address path-planning problems in a variety of environments are explained along with their assumptions and constraints. Next, the different approaches are compared based on the optimality of the solution they provide and their computational efficiency. Finally, the gaps in the literature are identified and our solution to bridge one gap is presented. The developed method is the basis of the 2D multipath planning problem that is explained in Chapter 3. The path planning problem in 3D environments is further discussed in Chapters 4 through 8.

2.1 Roadmap techniques

Roadmap techniques are among the first methods to address path planning problems. These methods map the free space to a connectivity graph that is later searched to find the shortest path. Roadmaps are geometric based methods, meaning they take into account the geometric properties of the obstacles when constructing the graph. According to Tran et al. [9], geometric-based path planning methods are more accurate than their

heuristic and stochastic competitors. The visibility graph and retraction method (using Voronoi diagrams) are examples of roadmap techniques.

2.1.1 Visibility graphs

Constructing the "visibility graph" to model the free space is deemed the first method in computational geometry to address the shortest path problem [10]. The method is introduced by Nilsson [11] to plan a safe path for a mobile robot.

The visibility graph of a set of polygonal objects in 2D consists of visible vertices of the objects that are connected by non-intersecting line segments. Any two nodes that can be connected by a line segment not intersecting any obstacles in the workspace are visible. Figure 2.1 shows an example of a visibility graph created for an environment with two objects.

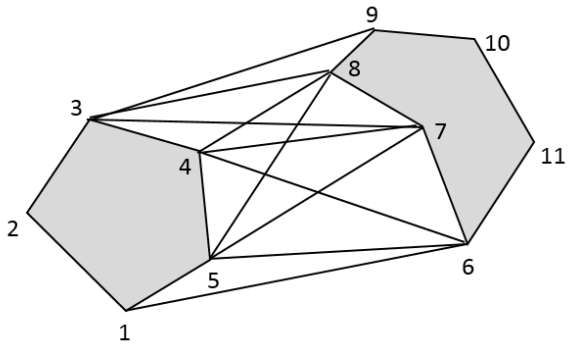


Figure 2.1 Sample visibility graph for two objects

The method is widely applied to different planning problems to reduce the routing problem to that of searching a graph of the feasible solutions, for example, see [12–16]. It is noteworthy that for robot motion planning, a configuration space introduced by Udupa [17] is first obtained that dilates the obstacles using the Minkowski sum of the robot's

geometry and the obstacle space. Consequently, the polygonal robot is treated as a moving point instead of a moving object. For example, Lozano and Wesley [13] tackled the problem of planning a collision-free path for a moving object of a known geometry among polyhedral obstacles using visibility graphs. To find the configuration space of the problem, they considered the position as well as the orientation of the robot. After determining the configuration space, a visibility graph needs to be constructed and finally searched for the shortest path.

Despite its simplicity and completeness, the brute-force algorithm to generate the visibility graph is computationally expensive since it explores all the obstacle vertices [18]. In fact, the classical algorithm to develop the visibility graph of an environment in 2D takes $O(n^3)$ time where n is the total number of vertices [19]. Hence, researchers have attempted to improve the efficiency of the visibility algorithm in one of these two ways: (1) developing more efficient algorithms to create the complete visibility graph and (2) downsizing and creating the partial visibility graph by eliminating the unnecessary edges.

2.1.1.1 Efficient algorithms for visibility graph construction

The early efforts in generating the complete visibility graph through developing more efficient algorithms date back to the studies by Lee [20], Welzl [21], and Asano et al. [22,23]. Lee's algorithm improves the complexity of classical visibility up to $O(n^2 log\ n)$ while Welzl and Asano, in their separate research works, have further improved it to $O(n^2)$.

Sharir and Schorr [24] investigated the shortest paths in 2D spaces with polyhedral obstacles. They developed an algorithm that constructs the visibility graph of the environment with n total number of vertices in $O(n^2 log\ n)$ time although they presented

some special cases for which the time complexity of the construction could be improved up to $O(n\ log\ n)$.

Additionally, Ghosh and Mount [25] achieved an output-sensitive algorithm with $O(E\ +\ n\ log\ n)$ time that computes the visibility graph for *E* edges of the visibility graph and *n* obstacle vertices. Readers are referred to [15,24,26–28] for further examples of efficient algorithms for the visibility graph generation.

2.1.1.2 Downsizing and partial visibility graph algorithms

Since the classical visibility graph is comprised of all non-intersecting segments, some of them might not be useful to find the shortest path. As claimed by Wein [29], for example, the edges created by nonconvex vertices are never used in the shortest path, therefore, they can be simply removed. In this section, some of the developed algorithms to construct partial visibility graphs are presented.

In a study by Clarkson [30], a method is proposed to improve the time complexity of the visibility-based shortest path algorithm. His developed technique works based on eliminating some of the unnecessary edges of the visibility graph. To construct the reduced visibility graph, he creates a family of cones per each vertex of the obstacles. The apex of the cones is the corresponding vertex of the obstacle. The edges of the reduced graph are segments between the apexes and the closest visible points in each cone. The reduced graph is a subset of the original visibility graph that needs to be augmented by the start and goal points of the path. The edges connected to the start and goal are added analogous to the other edges in the graph using the conical regions. A sample conical region is shown in Figure 2.2. He then applies the algorithm developed by Fredman and Tarjan [31] to find

the ε-shortest path. The ε-short path is the path that has a length no longer than $(1 + \varepsilon)$ times the shortest path. Clarkson's algorithm is capable of constructing the data structure in $O(n\ log\ n)$ and finding the ε-shortest path in 2D cases in $O(n\ log\ n + n/\varepsilon)$ time, with n vertices and ε, $0 \leq \varepsilon \leq \pi$.

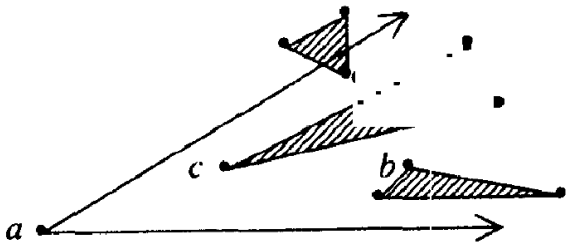


Figure 2.2 Sample conical region in Clarkson's method [30]

Hershberger and Guibas [32] considered downsizing the visibility graph. They developed a pruning heuristics decision making that removes the unnecessary edges based on mathematical rules and the triangle inequality. Their algorithm finds the shortest path for a moving convex body in $O(n^2)$ time.

Rohnert [33] in another study developed an algorithm that computes the shortest path in a Euclidean plane in the presence of disjoint convex polygonal obstacles in $O(f^2 + n\ log\ n)$ time for f number of obstacles. His algorithm generates the local visibility graph that works efficiently for planes with convex obstacles. He introduced the supporting segments between polygons and claimed that the shortest collision-free path between two given points passes through both the polygon edges and supporting segments. A supporting segment as defined by Rohnert is the common tangent line segment between two polygons. The supporting segments between two convex polygons are depicted in Figure 2.3. He then suggested that the partial visibility graph needs to be constructed using only the supporting

segments and the polygon edges. However, the supporting segments that intersect with other polygons of the environment are removed from the visibility graph. The elimination of these supporting segments may, however, restrict the feasible region and cause difficulty in reaching the globally optimal path.

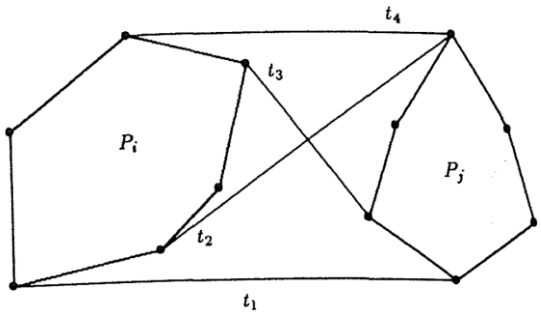


Figure 2.3 Supporting segments defined by Rohnert [33]

Following Rohnert's approach to reducing the visibility graph, Priya and Sridharan [34] developed a faster algorithm to create the supporting segments. Their algorithm benefits from a coding paradigm that identifies the tangent segments. They assign binary codes to different inner and outer regions created by a polygon and identify tangents by operating on these codes. Similar to Rohnert's technique, they remove the intersecting supporting segments from the visibility graph. Their algorithm can generate a reduced visibility graph in $O(f^2 + log((n/f)^2))$ which is an improvement from Rohnert's.

More recently, Jan et al. [35] have proposed an algorithm based on the Delaunay triangulation to reduce the size of the visibility graph and find the near-shortest path.

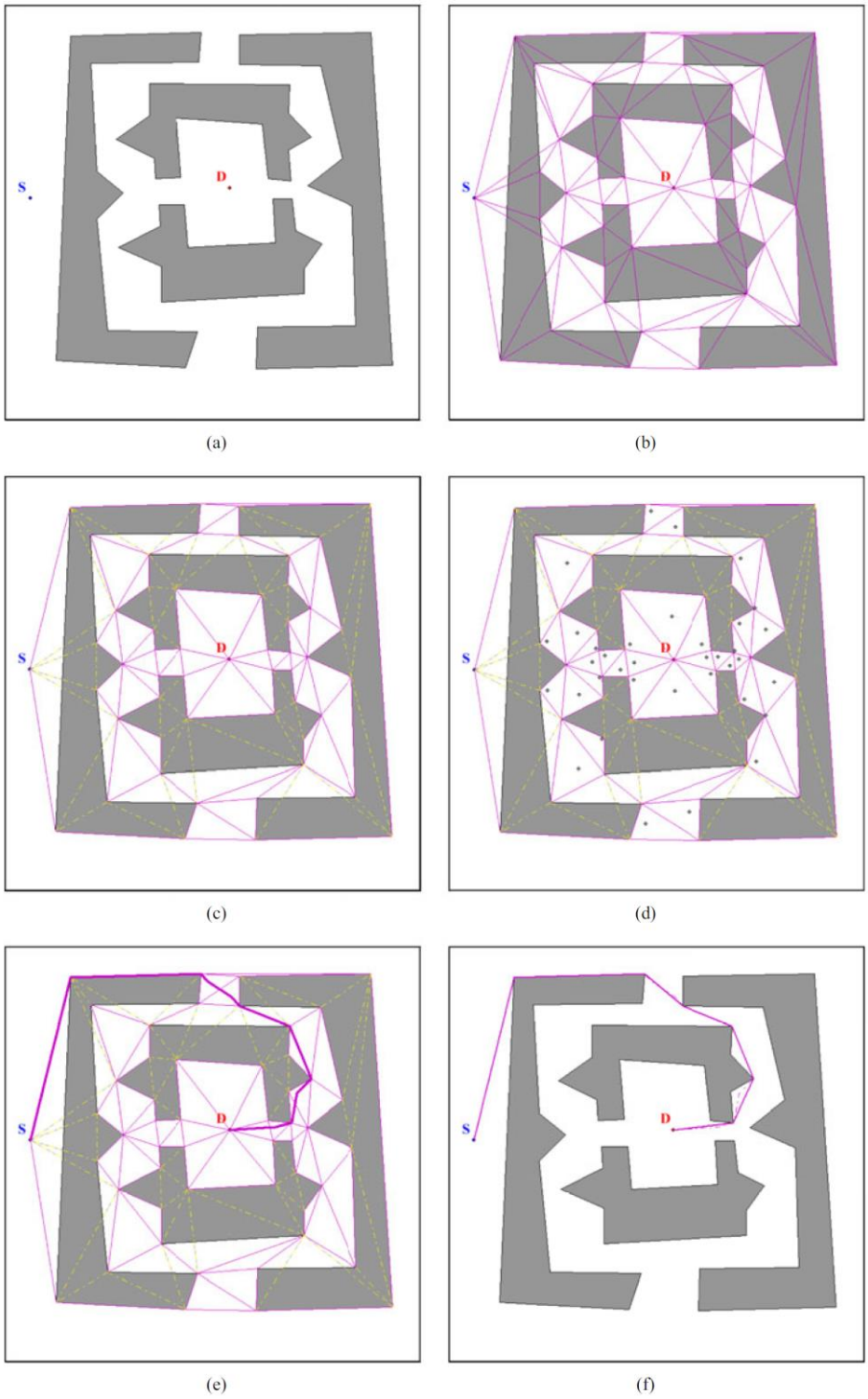


Figure 2.4 Steps in Jan's method [35]

Although the algorithm's dominant time complexity is found $O(n\ log\ n)$, for n obstacles, it requires multiple post-processing refinements and it only generates the near-shortest path [36]. The method also requires complete knowledge of the environment a priori to be able to perform the triangulation, which makes it inefficient for real-time planning problems. Moreover, as noted by Qureshi and Ayaz [37], Jan's method is limited to low dimensional spaces as it works based on workspace discretization which makes it inefficient for higher dimensions. The general steps in Jan's method are illustrated in Figure 2.4.

In another planar path-planning study, Jafarzadeh et al. [38] developed an exact method that applies to static environments with convex as well as nonconvex shapes. Their algorithm reduces the size of the graph by identifying the effective polygons and eliminating their unnecessary vertices from the graph. This algorithm finds the shortest path in $O(nn'^2)$, where n is the total number of vertices and n' is the number of graph's nodes.

In addition to reducing the graph size, some scholars proposed methods to create the graph for a limited region in the workspace, i.e. restricting the feasible region to that of the shorter paths. For example, one algorithm computes a region by the extreme vertices of the intersecting obstacles [39]. In another study [40], a grouping technique is proposed to merge multiple smaller neighboring obstacles into a bigger one (see Figure 2.5). Although this method reduces the number of obstacles, it compromises finding the global solution as it excludes the regions in between the smaller obstacles. This situation may worsen in tighter workspaces wherein the path should go through narrow passages.

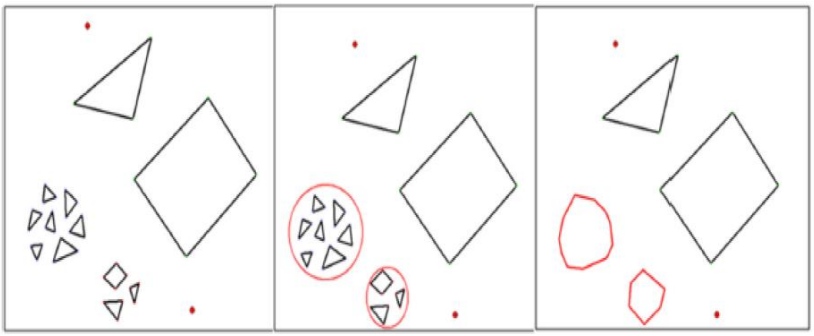


Figure 2.5 Steps in grouping obstacles in Toan's method [40]

Along with these efforts, Gasilov et al. [41] presented a technique that reduces the feasible region to an ellipse created by the start and goal points of the path as the two ellipse foci. They claim that based on the definition of an ellipse it is the right representative of the feasible region. Hence, paths found in the ellipsoidal region would be optimal. An example of such an ellipsoidal region with an optimal path is shown in Figure 2.6.

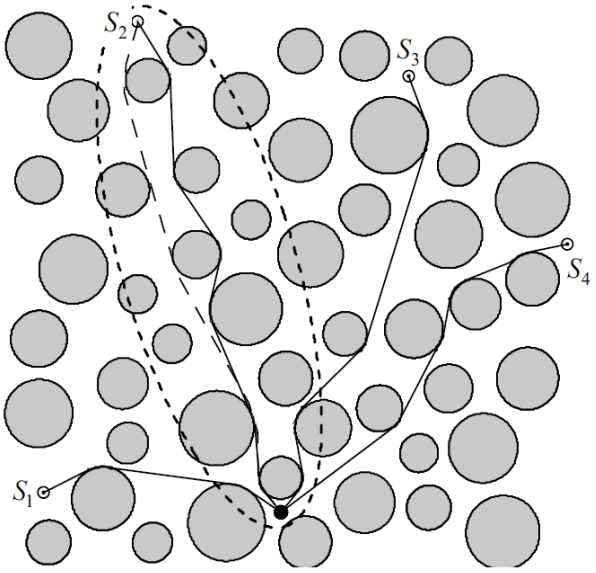


Figure 2.6 Ellipsoidal bounding region for downsizing of visibility graph [41]

In a rather similar manner, Badariyah et al. [42] proposed a method that limits the graph generation to an equilateral area. They create the visibility graph on a rhombus-shaped region, the diagonals of which are the baseline (the line connecting the start and goal points of the path), and the line perpendicular to the baseline at its midpoint. Despite the efficiency of the proposed algorithm in yielding the globally optimal path, it still generates unnecessary edges in the visibility graph of the limited region, and not all the obstacles that lie inside the region may be useful in finding the shortest path.

In the aforementioned studies, if efficient, the proposed methods are only valid for special cases that involve convex objects and if the objects are non-convex, a convexification is performed a priori. In some applications [32,43], the complete visibility graph has to still be developed and then downsized via removing some of the edges. In addition, the free space graph is created using all obstacles in the workspace or by limiting its feasible region [40–42] regardless of their contribution to the shortest path which could result in near-optimal solutions. In other studies [35], graph construction needs pre or post-processing steps to refine the final solution and obtain results closer to the global optimum.

2.1.2 Voronoi diagrams

Researchers have also utilized Voronoi diagrams in solving path-planning problems over the past decades [44]. Ó'Dúnlaing and Yap [45] are pioneers of using Voronoi diagrams for solving planning problems by introducing the “Retraction Method”. Simultaneously, though independently, Brooks [46], introduced the *freeway* technique, which is a more empirical version of the retraction by the notion of Voronoi diagrams.

As defined by O'Rourke [10], the Voronoi region of a point p, on a plane, is the set of all points that are closer to p than any other specified points or sites. By this definition, the Voronoi diagram of a set of n disjoint planar polygons divides the plane into n maximal clearance connected cells [47]. An edge of a Voronoi diagram is equidistant to two vertices or polygon edges while any Voronoi vertex is equidistant to vertices or edges of at least three polygons. Figure 2.7 illustrates a Voronoi diagram of four obstacles in a plane.

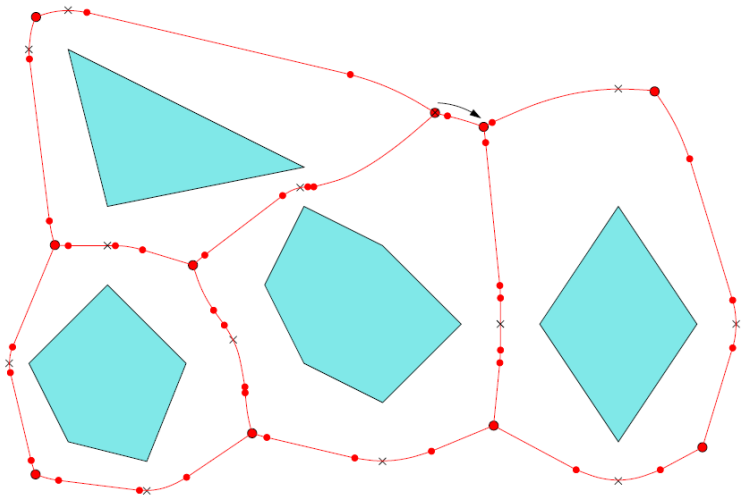


Figure 2.7 Example of a Voronoi diagram with four obstacles [47]

Bhattacharya and Gavrilova [48], undertook the problem of 2D path planning using Voronoi diagrams and developed a shortest-path algorithm that works in $O(n\ log\ n)$ time, n being the total number of vertices. They create the Voronoi diagram of the workspace by approximating the obstacles by their boundary points. They then dynamically add the start and goal points into the diagram and connect them to all Voronoi vertices to avoid intersections. Next, they define the minimum clearance, c, from the obstacles, and remove all the edges of the Voronoi diagram that result in a clearance less than c. Finally, they apply Dijkstra's search algorithm to find the shortest path on the graph.

A downside to this method is that the solution found might require some smoothing and refinement since the shortest path includes redundant vertices and unnecessary turns.

The Voronoi diagram is effective for cases where the maximal clearance or the safest path is of particular interest, for example, see [49–51]. Additionally, a generalization of this method is presented in [52]. Since the edges of the Voronoi diagram are created by the points equidistant from pairs of vertices and/or edges of the two closest obstacles, it results in the maximal clearance path, which is not necessarily the shortest.

To achieve the shortest path and at the same time maintain a certain clearance from the obstacles, Wein et al. [47] proposed an algorithm applicable to small-sized workspaces. They improved the efficiency of their algorithm up to $O(n^2 log\ n)$, over the time-expensive visibility graph construction. The algorithm evolves from a visibility graph to a Voronoi diagram as c grows from 0 to ∞. In the preprocessing phase, they grow the polygonal obstacles by c using the Minkowski sum of the polygon and a disk of radius c. They then, construct the visibility graph of the grown obstacles. In case a narrow passage is blocked by two or more of the grown obstacles, they find the intersection of the union of the grown obstacles and the Voronoi diagram, hence replacing the blocked portion by a Voronoi edge passing through the narrow passage. Although the clearance of the Voronoi edge from the blocking obstacles is less than c and it may yield sharp turns, to ensure the shortest path is achieved, this passage is allowed. The graph is later searched using Dijkstra's algorithm to find the shortest path. Despite the proven efficiency of this algorithm, it may not be practical to apply it to larger-scale problems [48].

2.2 Cell decomposition

Cell Decomposition method [53–55] aims at partitioning the collision-free space into a finite number of non-overlapping cells. The decomposition could be conducted either exactly or approximately [8]. The exact method decomposes the free space into cells of triangular and/or trapezoidal shapes [8]. Alternatively, the approximate decomposition starts with discretizing the workspace to a known number of cells of prespecified shape. The dividing of the cells is continued recursively until each cell is located completely inside or outside the obstacle space or a termination criterion is achieved.

Quadtree and octree techniques [8] are examples of approximate cell decomposition where the decompositions are in four (for 2D) and eight (for 3D) cells respectively. After the decomposition is complete, the neighboring cells are connected in the form of a graph capturing their adjacency information to search for the shortest path. More applications of cell decomposition in routing problems can be found in [56,57].

Neither exact nor approximate cell decomposition is efficient in finding the shortest path since the exact algorithm cannot provide the global solution and the approximate algorithm is not computationally efficient [58].

2.3 Potential Fields (PF)

In the PF method, first developed by Khatib [59], scalar functions similar to electrostatic potentials are assigned to all nodes of the search graph. The potentials assigned to the obstacles are the highest. The objective is then to find the path with the minimum electrostatic potential, thus avoiding collisions.

Unlike roadmaps and cell decomposition, PF is a local optimization method for which the development of a graph from the free space is not needed[8]. Despite its efficiency in dealing with collisions in real-time, as Overmars [60] states, PF's main drawback is the possibility of it getting stuck at local minima other than the goal, preventing the attainment of the true optimum. In addition to its main drawback, PF performs poorly in planning a path through narrow passages [61].

However, in recent years some heuristic techniques are introduced to reduce the risk of being trapped at a local optimum, (for example see [62–66]) though the techniques are predominantly applicable to special cases or otherwise increase the computational time drastically [60].

2.4 Stochastic and Sampling-Based methods

The probabilistic RoadMap (PRM) method, first presented by Overmars [60], generates a random graph in the free space. The probabilistic algorithm first adds the start and goal nodes to the graph; then introduces random nodes in the free space to be added to the graph until a complete path through the randomly generated nodes connects the start and goal. For more details and implementation examples, readers are referred to [67–71]. Also, other variants of PRM known as Rapidly-exploring Random Trees (RRTs) are presented in [37,72].

Intermittent Diffusion is another stochastic method for solving shortest path problems [73]. Lu et al. developed a stochastic algorithm based on intermittent diffusion that solves the path-planning problem in cluttered dynamic environments. They used a mathematical approach and modeled the path as a curve, the length of which is to be

minimized. To achieve the global solution, they use Intermittent Diffusion that finds a good approximation of the global minimizer of a scalar function [74]. Since the method is stochastic, the probability of achieving the global solution increases by increasing the run time. According to Chow et al., the method is proven efficient in solving 3D path-planning in static as well as dynamic environments scattered by obstacles with C^2 continuous boundaries.

PRM and other stochastic methods may be effective in dealing with dynamic or on-line path-planning problems. However, they may have difficulty meeting the optimization criteria of the path-planning due to the probabilistic nature of such algorithms in constructing the graph [75].

2.5 Heuristic methods

In addition to the four widely used algorithms for path planning problems, researchers have started integrating mathematical and heuristic methods to solve larger scale and real-time routing problems [76–82]. Most popular heuristic methods to solve path planning problems include the Genetic Algorithm, Simulated Annealing, and Ant Colony, although Tabu Search and Hill Climbing have also been used in the past. Despite their efficiency in solving NP-hard problems, they are not exact mathematical optimization methods; hence, there is no guarantee they can find the global solution.

In the next section, a comparison of the path planning methods is provided followed by a discussion of the limitations of the previous work and a proposed solution to address such limitations.

2.6 Comparison of path planning methods

The review of the related methods for 2D planning shows among all developed and practiced path planning methods, only visibility roadmaps have both properties of being able to guarantee the attainment of the globally optimal solution and being exact [60] as shown in Table 2.1. However, they could be computationally expensive. Although efforts have been made to improve the efficiency of the visibility method, they still fall short of yielding the global solution to the problem. A resolution to this challenge is suggested in the next section which is proven efficient for planar path planning problems.

The C-hull based approach described in [83] reduces the complete visibility graph to a local graph without loss of generality and therefore improves the efficiency of the graph construction algorithm. Unlike the previous algorithms [34,39], this algorithm does not require any pre-processing such as the convexification of any of the obstacles, neither does it need post-processing steps [33,35] to prune the graph. Therefore, it applies to the generalized path-planning problems on a plane including routing through narrow passages between obstacles that may be non-convex and is proven to generate the globally optimal solution.

Table 2.1 Comparison of path planning methods

Comparison Criteria	Method						
	Roadmap		Cell Decomposition		Potential Fields	Heuristics	Stochastic
	Visibility	Voronoi	Exact	Approximate			
Exact or approximate	Exact	Exact	Approximate	Approximate	Approximate	Approximate	Approximate
Global or local optimal solution	Global	Non-optimal	Local	Local	May trap at local optima	No guarantee for global	Local
Best-known complexity of computation	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	Depends on the size and shape of the cell	Depends on the potential function	Variable depending on the heuristic type	Not available
Real-time routing	No	No	No	No	Yes	Yes	Yes

2.7 Planar convex hull based approach for 2D routing

To overcome the above-mentioned challenges in roadmap development, Masoudi et al. [83] proposed an algorithm to construct the free space graph in a 2D environment scattered with arbitrarily shaped objects with the time complexity of $O(n\,log(\frac{n}{f}))$ for n vertices and f intersecting objects. The algorithm benefits from the convex hulls of intersecting objects which contain the candidate nodes and edges of the roadmap.

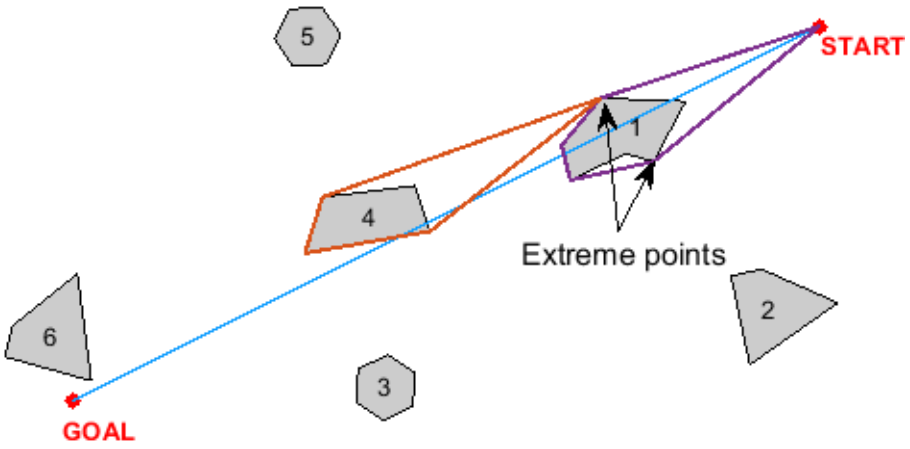


Figure 2.8 Generation of the convex hulls of the intersecting objects

After the intersecting objects are determined, they are ordered from the closest to the start point to the farthest. The algorithm then starts with creating the convex hull with the start point and the first intersecting object as shown in Figure 2.8. On this convex hull, two extreme points are identified. The extreme points are the points on the convex hull with maximum distances from the line connecting the start and goal points (see Figure 2.8). by this definition, two extreme points are identifiable on each convex hull, one per each side of the line. After the extreme points are found, each is set as the new start point of the path. At this step the algorithm checks for a direct collision-free path between the new start point and the goal point. If one exists, the algorithm is terminated; otherwise, the new

intersecting objects are identified, and another convex hull is created using the new start point and the first intersecting object. This process is iteratively continued until a complete collision-free graph is formed. The nodes of this graph are vertices of the objects on the series of convex hulls while the graph edges are extracted from the edges of the convex hulls.

Using convex hulls of only the intersecting objects leads to the construction of a smaller graph as well as handling both convex and non-convex obstacles by a more computationally efficient means, outperforming the previous methods. After the graph is completed, Dijkstra's search algorithm is applied to output the shortest route from the start to the goal. A graph with the shortest path between the start and goal points for a workspace with four arbitrary objects is shown in Figure 2.9.

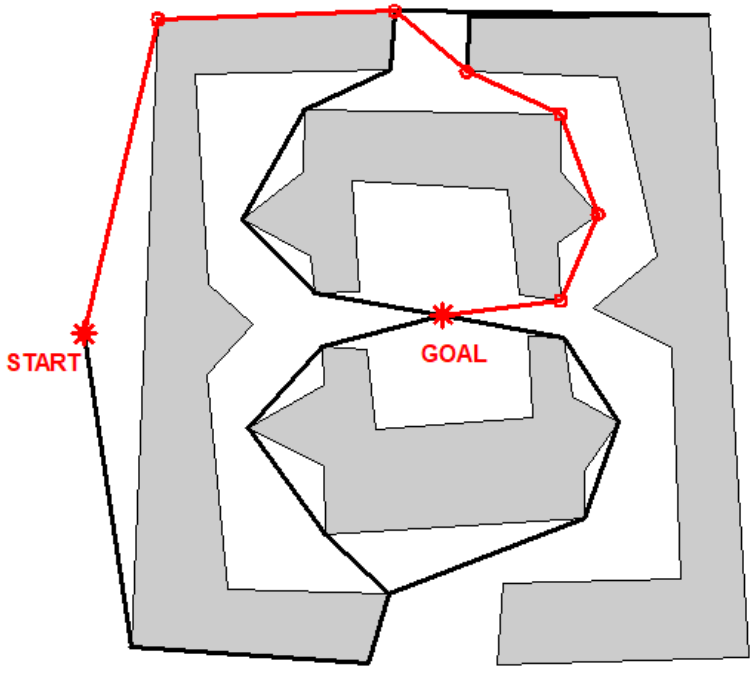


Figure 2.9 The solution of C-hull based planning

Table 2.2 Comparison of methods to improve visibility algorithm

	Reference	Approach	Time complexity
Efficient algorithms for complete visibility	Welzl [21], Asano[23], Hershberger [32]	Efficient Visibility algorithm	$O(n^2)$
	Lee [20], Wein[47], Sharir and Schorr [84]	Visibility	$O(n^2 log\ n)$
	Ghosh and Mount[25]	Output-sensitive visibility	$O(E + n\ log\ n)$
Reduced size visibility	Rohnert [85]	Partial visibility graph	$O(n + f^2 log\ n)$
	Priya et al. [34]	Reduced Visibility	$O(f^2 + \log((n/f)^2))$
	Jan [35]	Delaunay Triangulation	$O(n\ log\ n)$
	Jafarzadeh et al. [38]	Geometry-based	$O(nn'^2)$
	C-hull Based Roadmap [83]	Convex hulls	$O(n\ log(n/f))$

Table 2.2 summarizes and compares the attempts to downsize the visibility graph or improve the algorithm's performance based on the time complexity and their ability to obtain the globally optimum solution. As seen in Table 2.2, the C-hull based roadmap outperforms the other efficient algorithms developed to date. The presented time complexities only include the graph construction step.

Even though this algorithm is efficient in dealing with any planar routing problems among scattered obstacles, the exact algorithm may not be generalized to the 3D routing problems. Irrespective of the computational time, the idea of employing convex hulls to identify the next set of traveling points, while working efficiently for 2D problems, may not apply to 3D problems. 3D convex hulls contain more than two extreme points which make the identification of extreme points not as evident as in 2D. Hence, if an approach based on the convex hull notion is to be employed for 3D routing problems, a new method to identify the waypoints on the obstacle edges must be developed. This is further discussed in Chapter 7. For more details on the applicability of the convex hull based method to 3D problems, please refer to [83].

The next chapter discusses multipath planning problems in 2D cluttered environments using the convex hull based approach.

Chapter Three
2D MULTIPATH PLANNING AND THE CABLE HARNESS DESIGN PROBLEM

Even though planning the shortest path between two points in a cluttered environment is essential in applications such as robot motion planning, real-world routing problems often require planning of multiple routes, e.g. pipe routing or cable harness design. In complex interconnected systems like automobiles and aircraft, hundreds to thousands of wires and cables are required to connect various components of the system. The routing of these wires, therefore, becomes the multipath planning problem where multiple wires or cables are to be routed while collision with any of the objects (or even other wires) is prohibited. Often, these wires are bundled to form a cable harness assembly and start to branch out at a breakout point to connect to a system component. The number and location of these breakouts can then determine the final layout of the cable harness assembly.

Building on the results of Chapter 2, this chapter addresses the multipath planning problem in a 2D cluttered environment with a focus on cable harness design problems. The objective is to develop an optimization framework to determine the optimal locations of the breakouts in cable harness assemblies.

The chapter starts with a review of the related work and the identification of gaps in the literature. Then, a solution is proposed and tested to address some of the limitations in the previous studies.

3.1 Review of the related work

The design and routing of cable harness assemblies have been a challenging problem for decades. Yan et al. in a survey of design of cable harness assemblies [86] and Ng et al. [4] independently pointed out several challenges in the design process of cable harnesses such as being costly, time-consuming, complex, tedious, and often done by a trial-and-error approach in the late stage of detailed design that leaves limited space for harnesses. They claim that even though attempts have been made to fully automate the process, they were not completely successful and human input is still demanded at different stages of design.

By this introduction, efforts to address cable branching and layout design are channelized in two main directions, each of which has their sub-branches: (1) Design process for cabling in different products/systems and (2) optimization of the cable layouts to satisfy various objectives including but not limited to the minimization of wire lengths, minimization of the number of branches, and minimization of the number of breakouts or junctions.

It is also noteworthy that these categories are not mutually exclusive; for example, in some of the design-based studies that are discussed in the next section, optimization of the wire/cable routes are also considered as a step in the design process. The optimization class of approaches, on the other hand, mainly focuses on developing or deploying a mathematical framework to achieve the set objectives while satisfying the optimization constraints. Therefore, these efforts often overlook the actual design process that leads to

the development of the final product or system. A classified summary of the studies reviewed in this research is shown in Figure 3.1.

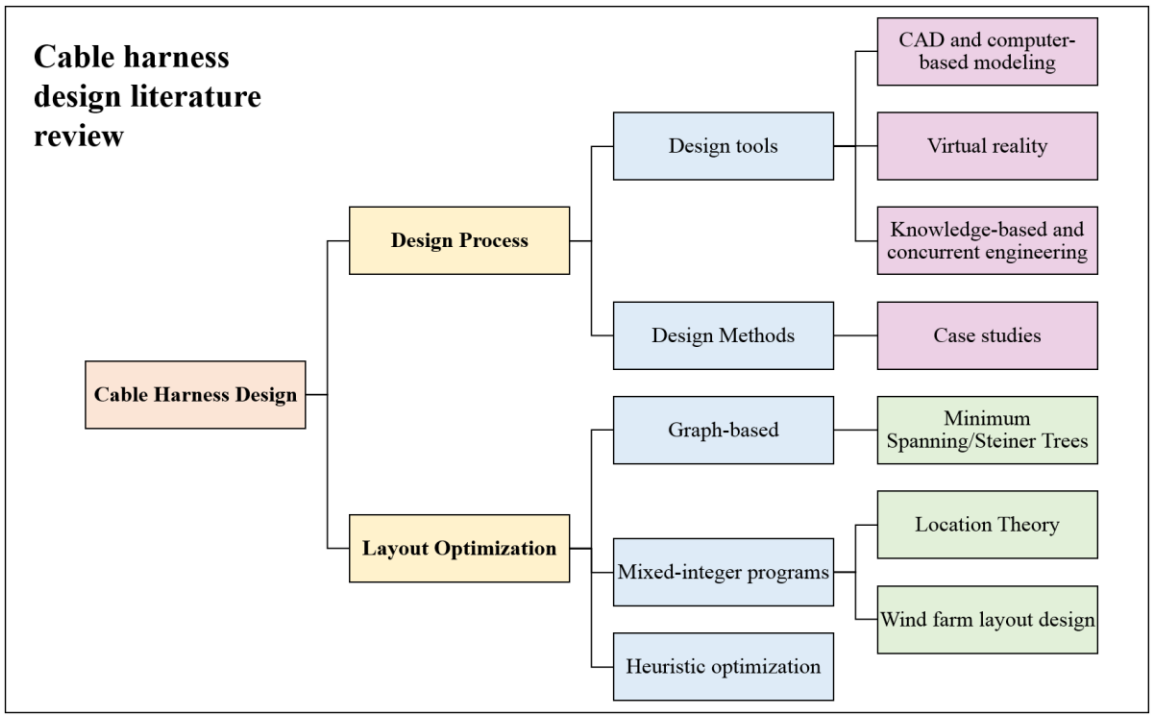


Figure 3.1 The literature review of cable harness design

3.1.1 Design of cable harnesses and piping systems

Design of one-dimensional connectors such as wires or pipes is a complex problem as the designer is often left with a limited free space to squeeze a large number of components, the requirements are varied across the multiple disciplines involved, and the sizes of the components can also be different [3]. For example, there could be requirements on the bending stiffness and mass distribution in a cable, or some requirements may even change depending on a region in the environment such as a high-temperature zone that requires thicker cables or insulation [3]. Therefore, different tools have been developed over the years to assist the designer from modeling the design environments and connectors

to choosing the route and size of wires or pipes. In addition, design methods such as case studies have been followed in designing cabling and piping systems.

While pipe routing and cable/wire routing may seem to be similar to the problems of planning paths for one-dimensional connectors in a system, the two have fundamental differences. For example, it is evident that the piping system is most often comprised of orthogonal routes for which the Manhattan distance metric is used, whereas, cables or wires have more flexibility in their shapes and therefore Euclidean metric is a more appropriate measure of distance for them.

The remainder of this section is allocated to the review of the related work in the design and optimization of cable harness layouts and similar problems.

3.1.1.1 Design tools

CAD and computer-based models in the design process

CAD and other computer-based models are helpful tools in the design process of pipes and wire harnesses. In this section, a review of the research efforts in this area is provided starting with the piping systems in ships or power plants.

An integrated computer-aided piping design system for the design, planning, and fabrication of piping systems in ships is introduced in [87] as one of the early works in the computer-based design of piping systems.

Another geometric modeling kernel is introduced in [88] based on documented design regulations and human designer's knowledge, which aids designers in modeling ship pipes by providing a user interface.

Roh et al. [89] pointed out a shortcoming in the then-available CAD support systems for ship pipe designs. Even though CAD packages were available at the time to support the designers, there was a lack of relationship between the CAD model of the pipes and the hull structure, i.e., any changes in the hull structure would have not affected the pipe model and vice versa [89]. Hence, if any changes were made to the hull structure, the designer had to manually modify the piping model to reflect such changes. To overcome this limitation, Roh et al. proposed a method that generates the piping model considering its dependence upon the hull structure. The method, however, does not consider the effects of the changes in the piping model on the hull structure.

Other studies explored CAD support systems specifically designed for wire/cable harness design, some of which are briefly explained here.

For instance, Billsdon and Wallington developed a CAD package that assists human designers with selecting the parts to be connected in an attempt to address the lack of a standard CAD software package for wiring harness design [90]. At the outset, a harness assembly drawing, which can be created using Microsoft Visio, is inputted to the system. The software then outputs a design sequence to connect the chosen parts. It also provides guidance on selecting wire sizes and materials based on the imposed design constraints. Although the package is made to work interactively, the final path for wires may not necessarily be optimal.

In another study, Lindfors et al. compared the cabling design done on physical prototypes with that of CAD software [2]. Although in their view, CAD packages save

designers' time by removing the need to test the design on physical prototypes, hardly could these systems yield an optimal solution for the configuration design.

Additionally, a flexible geometric model of cables with B-splines for virtual maintenance using VRML (Virtual Reality Modeling Language) is presented in [91].

A review of the CAD packages capable of routing cable harnesses is provided by Han and Guo [92] followed by a new cable harness modeling approach using design rules in Pro/E software.

Design guidelines

The focus on the design of cable harnesses mainly leads to developing a set of design guidelines rather than identifying ways to find the optimal configuration for the cable harness. For example, Lin et al. [93] developed a set of instructions for cost minimization of wire routing and wire sizing in electrical circuits while also considering the shortest routes for wires found using the depth-first method.

Virtual Reality

In recent years researchers have extensively studied the incorporation of Virtual Reality in the design and planning of cable harnesses [86,94]. This tool allows the designer to apply his/her knowledge and expertise in the design process especially where human input is required and makes a difference to the outcome of the design.

Ng et al. in a series of research studies [4,95–98] proposed a possible implementation of a virtual reality environment to model the design process of cable harnesses with the use of a designer's expertise. They claim that their approach enhances

the current automation degree in the process and enables the designer to develop the design schematic faster than using the previous methods [4].

Park et al. [3,99] discussed other levels of human interaction and collaboration in the concurrent and often multidisciplinary design of cable harnesses from modeling the workspace to assessing the final design solution.

Knowledge-based and Concurrent Engineering

In line with virtual reality tools for the design process to benefit from the human designer's input, some researchers also focused on the design and simulation of cables using human knowledge and Artificial Intelligence (AI) [100]. As in the past, cable layouts were carried out on prototypes or physical mockups based on human knowledge and experience, these researchers argue that this experience should not be overlooked. For example, in [101,102] different knowledge-based routing techniques are employed for the cable design problem. In this view, two approaches are generally taken: (1) the human is considered in the loop and can interact with the design environment, thus, the routing process is not fully automated or (2) the system captures human knowledge and imitates human behavior in design, hence automating the design process. However, in either case, there is no guarantee that the final solution is optimal since it only relies on human experience[93].

Advocates of knowledge-based design of cable harnesses believe that the full automation of the process is infeasible and human knowledge must be the base of this dynamic and iterative activity [103]. A survey of the AI and nature-inspired algorithms to

tackle cable harness design problems can be found in [104]. For more studies using AI and knowledge-based or concurrent engineering see [103].

3.1.1.2 Design methods

In addition to tools that aid designers in their decision making on cable harness or pipe route design, research methods are deployed to further study and improve the design process for these components. Case study is among the common methods used for the cable harness design process.

Case study has been highlighted as a design research method by Teegavarapu et al. [105]. As defined in [105], case study is "*an empirical method that investigates a contemporary phenomenon, focusing on the dynamics of the case, within its real-life context."* Research that benefits most from this method usually answers "why" and "how" questions. By this definition, it appears that the method can be suitably applied to cable harness design research.

As an example, Ng et al. [106] used a case study method to observe and investigate how cable harness design is practiced across five British advanced manufacturing companies. With these case studies, they confirmed that the industrial design process is sequential and reliant on the designer's expertise which involves a lot of trial-and-error. Additionally, they found out that the process is time-consuming, late in the design stage (which could even lead to the expensive re-design of the entire machine chassis to provide sufficient space for the cable routes) and still requires costly physical prototypes for validation. Figure 3.2 shows a generic model of the design and planning for cable harnesses developed by Ng et al. based on their case studies.

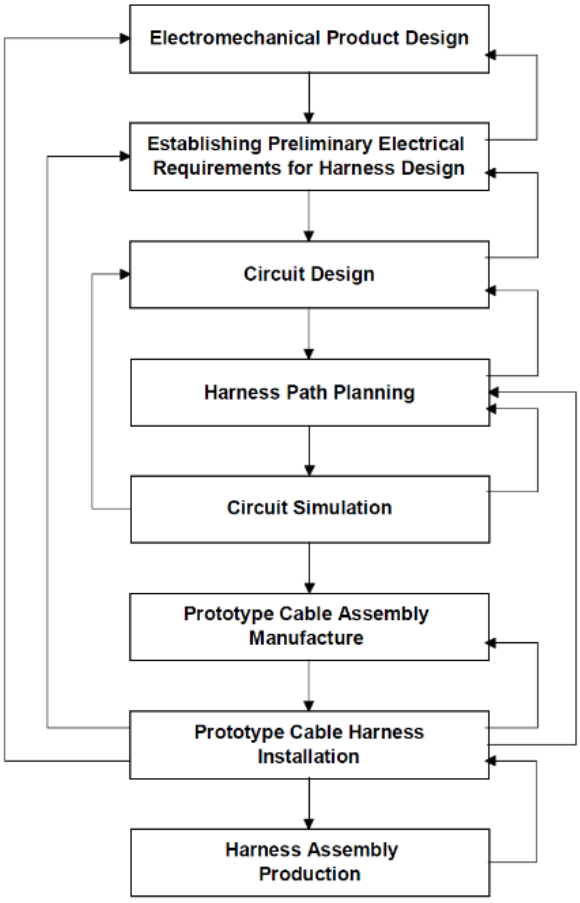


Figure 3.2 Generic cable harness design process [106]

The harness path planning in the five studied companies was performed either manually or through CAD packages such as CATIA; either of which lacks optimality in the provided solution for the cable routes. They also mentioned that it is a common practice in industries to determine the cable lengths, paths, and locations of breakouts manually using the physical prototype.

Based on the findings of this case study research, the authors finally made recommendations on the incorporation of a concurrent rather than sequential design

approach. They also pointed out the effectiveness of a virtual reality environment where cable harnesses can be designed using the expertise of a human designer.

3.1.2 Optimization of harness layouts

While the focus of design-based studies is mainly on the design process of the cable harness assemblies which requires some levels of human intervention, other studies focus primarily on generating optimal routes for cables or pipes and optimal locations for the breakouts in cluttered environments. The goal, herein, is to overcome the limitations in the overall design process of these components, namely the manual determination of paths and locations of breakouts for cable harnesses which often lacks optimality.

The research efforts in this area have led to the introduction of several deterministic as well as heuristic optimization methods or algorithms. Benefiting from the analytical properties of an optimization problem, deterministic methods generate a series of solutions in the feasible domain that eventually converge to the global solution [107]. Heuristics, on the other hand, are often used when applying deterministic methods is not efficient. This mainly occurs in large-scale or non-convex optimization problems [107]. Heuristics, however, cannot guarantee to converge to the global solution.

A brief review of the fundamental studies on the optimization of cable harness/piping assemblies is provided in this section starting with tree-based methods.

3.1.2.1 Steiner and spanning trees

Steiner Minimal Tree (SMT) and Minimum Spanning Tree (MST) are two popular methods for network optimization problems. Given a set of nodes, MST is the minimum

length tree that interconnects and spans all the nodes, which makes it an immediate solution to the wire routing problem [10]. Prim [108] and Kruskal [109] have independently developed methods to construct the MST for a given set of nodes.

Steiner Minimal Tree also pertains to minimizing the overall length of a network. Steiner trees, however, introduce external nodes (often called Steiner vertices) into the tree in order to further minimize the length of the tree [110].

Ever since the introduction of Spanning and Steiner trees, researchers have developed a variety of deterministic as well as heuristic algorithms to construct these trees for a given set of nodes, for example, see [111–123]. Due to the intrinsic advantages of minimizing the length of a network while spanning all or specified nodes, Steiner and Spanning trees are suitable candidates for cable harness layout optimization. As a result, a multitude of studies has looked into furthering the use of these methods in cable and pipe routing design, some of which are discussed in detail next.

In one example, Lin et al. have formulated the wire routing problem as a Steiner Tree problem with capacity constraints on the breakouts (where more than two wires are connected)[124]. After constructing the Steiner tree, they reformulate the problem as an Integer Linear Program (ILP) to relocate the breakouts and satisfy the capacity constraint. Next, they relax the ILP to a linear program since there exist more solution methods to solve this type of optimization problem. Due to this relaxation, the final solution becomes suboptimal.

Sommer et al. in another study [125] developed a method that optimizes the topology of Ethernet networks by finding the optimal locations of junctions (breakouts) in

the network. They used Simulated Annealing (SA) on the initial solution generated by placing the junctions randomly on the network and applying the minimum Steiner tree to connect all the random nodes in a minimum length tree. Following this approach, a near-optimal solution for the location and number of junctions on the network is achievable.

Looking at the minimum spanning tree and the shortest path problem simultaneously, scholars in the field of operations research have defined the "cable trench problem" [126] and proposed various solution methods to address its instances [127–132]. The problem is defined as: let a connected graph with its known sets of vertices and edges be given. The objective is to minimize the weighted sum of two functions: the total length of the spanning tree and the total length of all paths from a specified vertex, v_0, to all other vertices in the graph. The name has originated from the application of this problem in connecting the buildings on a university campus to the building that houses the main computer [126]. Since only the edges from the edge set are allowed and all the vertices must be connected to v_0, a Steiner tree cannot be the solution [126].

For real-world problems of cable and/or pipe routing, the obstacle avoidance constraint must be satisfied when a spanning or Steiner tree is to be formed. It is noteworthy that for cabling, Euclidean Steiner or spanning trees are of interest while for piping the rectilinear (or Manhattan) trees are normally generated that reflect the orthogonality of pipes.

Multiple solutions for the construction of obstacle-avoiding trees have been proposed by scholars over the past three decades. For instance, a Steiner visibility graph is introduced by Winter that produces suboptimal solutions to the Euclidean Steiner tree

problem with polygonal obstacles [133]. As mentioned by Winter, the Euclidean Steiner tree problem is NP-hard, even in the absence of any obstacles. Therefore, good heuristics ought to be sought to solve even small size Steiner trees with obstacles [133].

Winter's idea was to break down the Euclidean Steiner Tree Problem with Obstacle (ESTPO) into subproblems of finding obstacle-avoiding Steiner trees for subsets of two, three, and four terminals. After these smaller subtrees are found, they are concatenated in the form of a spanning tree which is the final obstacle-avoiding Euclidean Steiner minimal tree. The problem of finding these subtrees becomes challenging when the number of terminals exceeds three or there remains more than one obstacle (or the only remaining obstacle is non-convex) after pruning the irrelevant obstacles. Therefore, Winter introduced heuristics to address such cases assuming the obstacles are convex, for simplicity. In addition, to avoid intersections with obstacles, his algorithm benefits from a geometric construct called *Steiner visibility graph*. An example of two Steiner visible points is shown in Figure 3.3 as described by Winter.

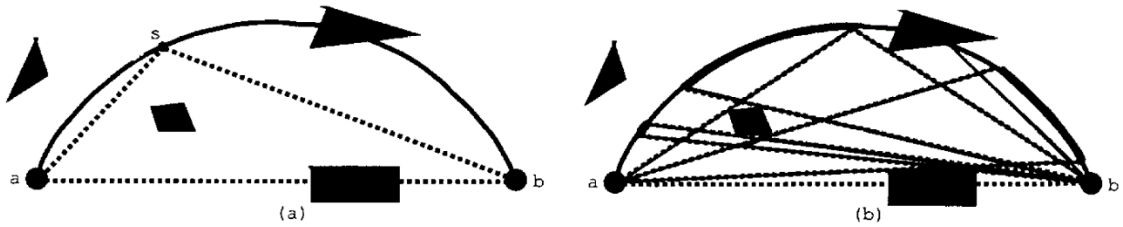


Figure 3.3 Notion of Steiner visibility [133]

By Winter's definition, the point b is Steiner visible from a, iff a point s can be placed on the arc ab, such that a and b are both visible from s. For further details of constructing the graph and determining the arc ab, readers are referred to [133].

Building on the success of their obstacle-free Steiner tree construction [120], Zachariasen and Winter proposed an exact algorithm for the obstacle-avoiding Steiner minimal tree construction [134]. Similar to Winter's [133], their algorithm also takes advantage of the generation and concatenation paradigms. In further detail, they generate full Steiner trees (FST), which are Euclidean Steiner trees where all the terminals are of degree one, for subsets of terminals and store the shortest obstacle-avoiding FST. The union of these FSTs involving the terminals and some of the obstacles' vertices will form the final obstacle-avoiding Steiner tree. Contrary to Winter's heuristic approach, they used a visibility graph that enables the computation of the shortest obstacle-avoiding distances between any two points in the plane. Though the use of visibility graphs may increase the overall computation time, it allows having non-convex obstacles while generating the FSTs and finally results in a more optimal solution. Another visibility-based obstacle-avoiding Steiner tree construction method is proposed in [135] which benefits from approximations to improve the time complexity up to nearly linear time.

Parque and Miyashita while looking at constructing an obstacle-avoiding Euclidean Steiner tree, also considered preserving a known topology (or *n-star* topology for *n* terminals) in the tree [136]. As they claimed, this particular case is of importance in layout design when clutter-free visualization of networks is of interest (e.g. in VLSI design).

In addition to obstacle-avoiding Euclidean Steiner trees, some scholars explored the construction of rectilinear Steiner trees in the presence of obstacles which has specific application in pipe routing optimization and circuit design. For example, Chiang et al. [137] introduced a weighted minimal Steiner tree to address the routing of wires in the presence

of obstacles and obtain the globally optimum solution. They assign infinity weights to obstacles as an indicator of a high-cost region and to avoid a path going through them. The weighted minimal Steiner tree then minimizes the weighted sum of the lengths. For more studies on the use and generation of obstacle-avoiding rectilinear Steiner trees, please see [138–142].

3.1.2.2 Mathematical programming for wind farm layouts

The problem of cable layout design is also vital in applications like wind farm layout design and planning. The objective of wind farm layout design is to find the location of wind turbines to meet the problem requirements. A combination of different mathematical programming, as well as heuristic methods, have been presented to address the layout design of wind farms. Wędzik [143], for example, looked at the problem of designing a new wind farm from the perspective of locating the wind turbines. He compared the efficacy of a graph-based optimization using the minimum spanning tree against a Mixed Integer Program (MIP) when the problem is formulated as a cable trench problem. He concluded that while the difference between the length of cables produced using either method was negligible, the MIP method provided more flexibility in the selection of different components for the wind farm (e.g. the number of wind turbines in each section of the farm) which could be of high importance for designers.

Wind turbine allocation and their optimal connection using cables for both onshore and offshore wind farm designs are investigated in [144]. The authors made use of Mixed Integer Linear Programming (MILP) to address the two problems while also considering physical constraints (including the wake models that affect downstream turbines) that, due

to their nonlinearity, are modeled using stochastic programs. Their proposed MILP model determines a feasible allocation of turbines while maximizing power production. The constraints pertinent to the layout optimization include the minimum and the maximum number of turbines that can be built, clearance between any two consecutive turbines (to ensure the blades do not interfere), and the foundation cost (for the offshore case). It is noteworthy that their optimization model benefits from a grid that comes with possible locations for the turbines and thus the decision variables are binary variables indicating whether or not a specific grid point is selected for a turbine location.

After the layout is optimally determined, the next problem is to find an optimal cable connection between all turbines and the substations. The constraints imposed on this problem include the capacity constraints for cables, a no-crossing constraint between any two cables, and the constraint on the maximum number of strings that can be connected to a substation. This problem is also modeled and solved as a MILP. A sample layout generated by this method is shown in Figure 3.4 for 72 turbines and one substation.

In their next study [145] Fischetti and Pisinger added the real-world constraint of avoiding obstacles and the objective of minimizing power losses to the wind farm layout optimization problem. To model the forbidden area imposed by an obstacle, they introduced “dummy” nodes at the vertices of the polygonal obstacle. To indicate the borders of the obstacle, they forced zero-cost cables in the problem formulation by setting the corresponding binary variables equal to 1. Thanks to the no-crossing constraint on the cables, the actual cables are not allowed to cross the obstacles. An example of the optimal layout for a wind farm in the presence of polygonal obstacles is illustrated in Figure 3.5.

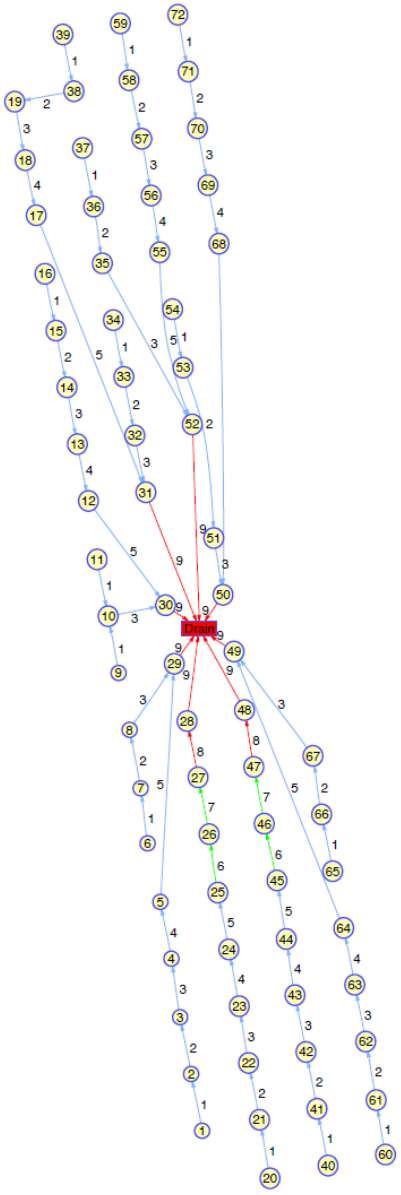


Figure 3.4 Final optimal layout for a wind farm[144]

Another MILP-based solution for the wind farm layout design is presented in [146] that considers the cost of energy losses and technical parameters of cables and turbines (e.g. number of feeders, cables' cross-sections, and the number of turbines connected to one feeder) in the optimization model.

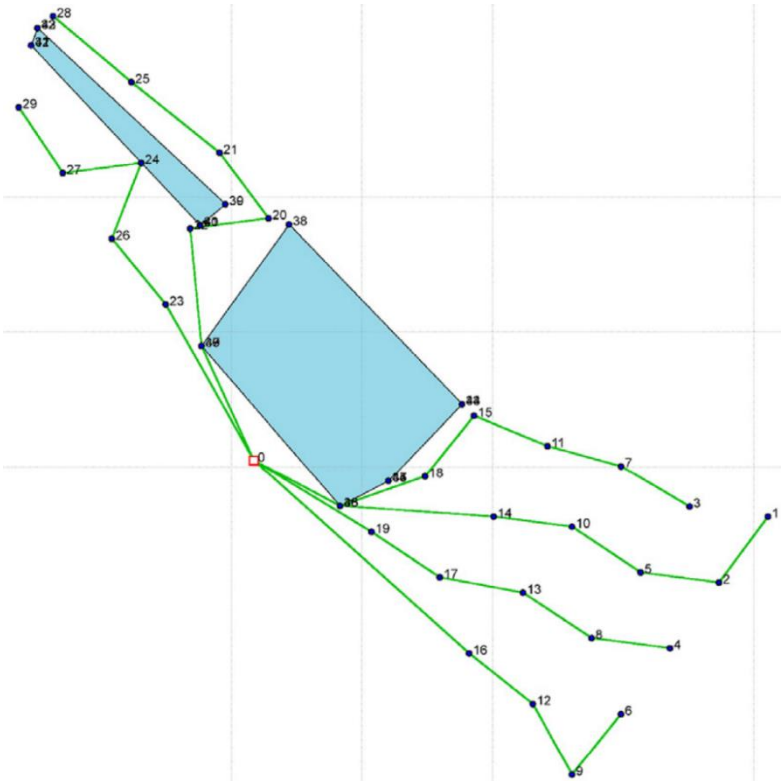


Figure 3.5 Obstacle-avoiding optimal wind farm layout[145]

3.1.2.3 Location Theory

There are instances where the primary focus in a cable harness layout optimization problem is on finding the optimal location of the path breakouts. Location theory (aka facility-location)in operations research (OR) deals with problems of this kind.

Alfred Weber's well-known location problem [147] aims at placing a new facility in the vicinity of a number of existing facilities to minimize the sum of its transportation costs to all facilities. Different versions of this problem are addressed in business and OR fields [147]. A classification of location problems is presented by Hamacher and Nickel [148]. For each location problem, they defined five attributes to classify the problems in the form of *Pos1/Pos2/Pos3/Pos4/Pos5*. The 5 properties are attributed to the number and

type of facilities, type of location topology, model specifics, the relation between facilities, and type of objective function, respectively. Continuous location, network location, and discrete location are three classes of location models based on Hamacher's definition. They also provided a general approach to multicriteria planar location problems in the absence of obstacles for the single facility case though they have addressed the planar multicriteria multi-facility location problems in their other work [149].

The location problem in the presence of obstacles can be modeled analogously to the cable harness layout optimization when the decision variables are the locations of the breakouts. Therefore, it is worthwhile to review the related work in location problems in the presence of obstacles.

Katz and Cooper are among the first researchers who considered the location problem in the presence of obstacles. They addressed the problem of locating a new facility in the presence of one circular forbidden region using the Euclidean distance metric [150]. They modified the distance function to geodesic distance using the calculus of variation to be able to find the shortest non-intersecting path between any two points and solved the nonlinear location problem using sequential unconstrained minimization technique.

As an example of the continuous location problem, Aneja and Parlar looked into Weber's location problem for the single facility in the presence of forbidden regions [151]. They deployed a visibility graph to create a network with the existing facilities and the barriers' corners and applied Dijkstra's algorithm with the source node being the location of the new facility to find the shortest routes to all existing facilities. Lastly, they used Simulated Annealing to find an approximate optimal location for the new facility.

Since the distance between any two points may no longer be calculated as simple Euclidean or Manhattan (or any other l_p norms) when their direct path is blocked by a barrier, Hamacher and Klamroth redefined the distance metric for such cases by introducing polyhedral gauges [152]. The new distance function makes use of the piecewise continuous parametrization of the permitted path connecting the two points, i.e. a curve that does not intersect the interior of any objects. The length of this curve is equivalent to the shortest non-intersecting distance between any two points.

To further simplify the computation of the non-intersecting distance, they used the polyhedral gauges instead of the parametrized curve between the two points. As described in [152], a polyhedral gauge is given by a convex symmetric polyhedron in the plane, containing the origin in its interior. They used the extreme points of this polyhedron to define the fundamental directions (see Figure 3.6). For any point X inside the cone spanned by two consecutive fundamental directions d_i and d_{i+1}, only these two fundamental directions need to be used to determine the norm of X (e.g. d_1 and d_2 can define ||X|| in Figure 3.6).

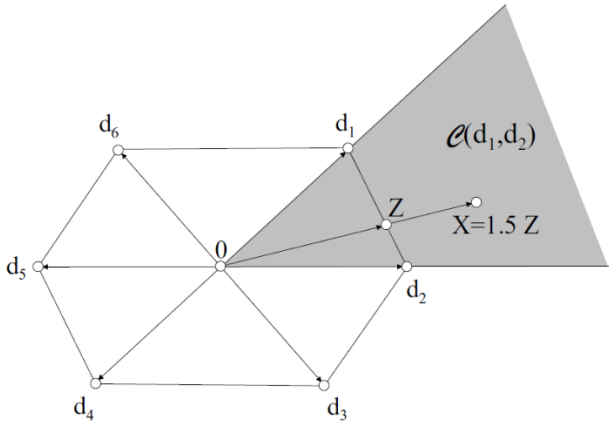


Figure 3.6 Sample polyhedral gauge with 6 fundamental directions [152]

In addition, Hamacher and Klamroth noted that the presence of obstacles in the location problem destroys the convexity of the objective function. As a result, they proposed a discretization of the plane using the fundamental directions at the existing facilities and barriers' extreme points. They proved that one of the grid points is optimal for the location problem in the presence of convex barriers using polyhedral gauges. A sample constructed grid, based on their algorithm, is depicted in Figure 3.7.

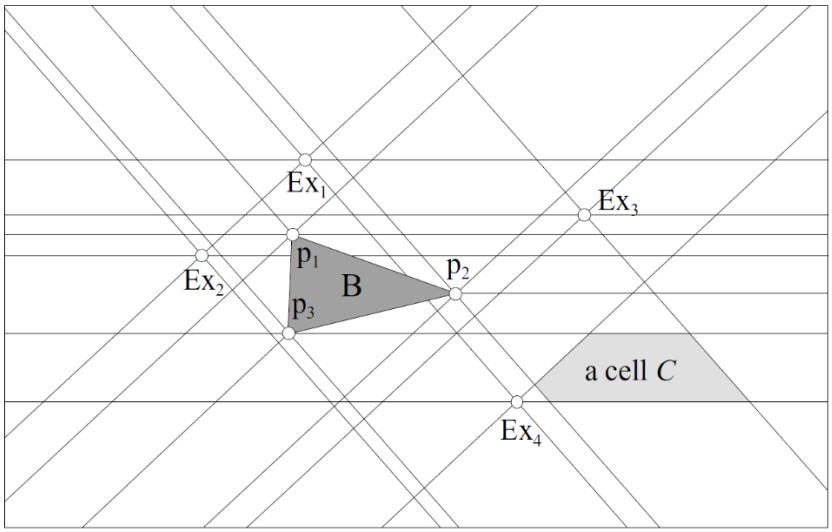


Figure 3.7 Sample grid for location problem with barriers[152]

In Figure 3.7, *Exi* denotes the i^{th} existing facility ($i = 1,..,4$) and *B* is the shaded region occupied by the triangular barrier whose vertices are p_1, p_2, and p_3. Also, in this figure an example of a cell created by the discretization of the plane is shown in the shaded area denoted by "a cell *C*."

Klamroth in another study [153], proposed a reduction of the nonconvex barrier problem to a set of convex location problems without barriers using a novel subdivision of the feasible region which led to an exact algorithm. The subdivision makes up a grid denoted by the boundaries of the shadows of all existing facilities and all extreme points

of the convex polygonal barrier. Given a distance function *d*, the set of all points in the region that are not visible from a point *X* in that region form the shadow of *X* with respect to *d* [153]. An example of shadow shown in [153] is provided in Figure 3.8 followed by a final subdivision of the feasible region to decompose the non-convex location problem to a finite set of convex problems pictured in Figure 3.9.

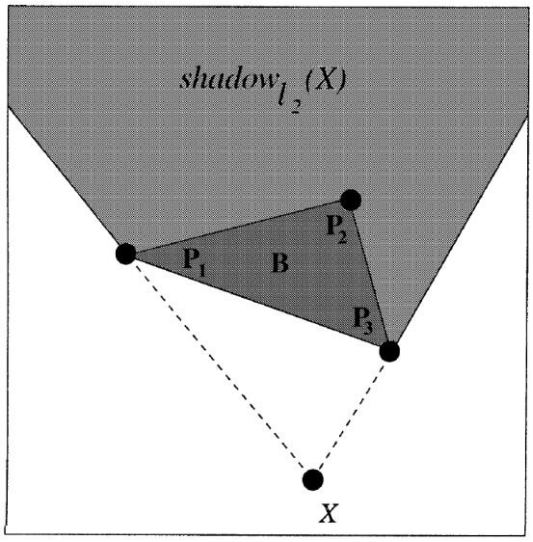


Figure 3.8 Illustration of the shadow of a point [153]

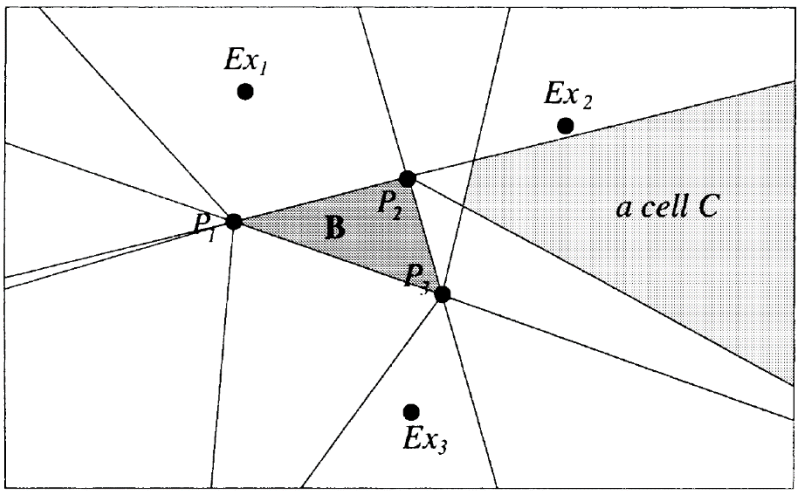


Figure 3.9 Sample subdivision of the feasible space by Klamroth's method [153]

In Figure 3.9, the discretization of the plane is performed using the shadow of each existing facility, Ex_i, and the extension of the borders of the barrier ***B***. The construction of the grid lines is therefore based on the notion of shadow described in Figure 3.8. The formation of the grid lines is further justified in Figure 3.10.

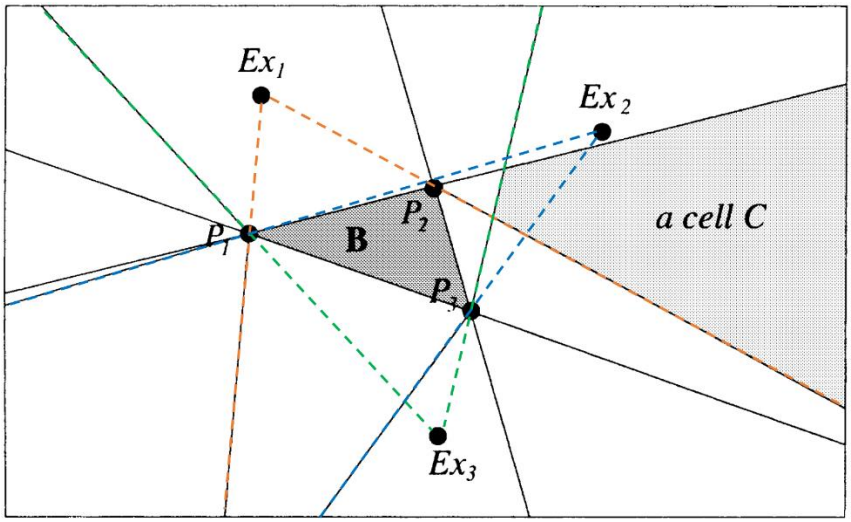


Figure 3.10 Construction of the grid in the feasible domain

To further improve the computational performance of the location algorithm in the subdivided region proposed in [153], Bischoff and Klamroth [154] found applying a heuristic (genetic algorithm) beneficial to solve a finite series of convex subproblems though the final solution is an approximation to the globally optimum.

A global optimal approach to locating a facility in presence of convex forbidden region(s) is presented by Mcgarvey and Cavalier [155] using a version of the branch-and-bound algorithm known as Big Square Small Square (BSSS) developed by Hansen [156]. BSSS divides the plane into discrete squared regions and provides global or near-global optimal solutions.

Kuhn proved two results from Weber's location problem[157]. First, if the facilities are not collinear, the objective function is convex meaning any local optimum is also a unique global optimum [155]. And second, the location of the new facility is inside the convex hull of the existing facilities. This helps to limit the search region and to improve the computation time. In the case of location problems with barriers, it is not sufficient to only include the existing facilities when creating the convex hull as the boundary of the convex hull might intersect with an object. Thus, Klamroth [158] suggested an iterative convex hull approach that extends the boundary of the convex hull to include the intersecting objects. The boundary of the convex hull expands iteratively until all the edges of the convex hull are found non-intersecting.

In the case of non-convex forbidden regions, Butt [159] has shown that the location of the new facility will never be within the convex hull of a non-convex forbidden region unless an existing facility locates inside this convex hull.

Finally, a multi-facility location problem with polyhedral barriers is considered in [160]. They proposed two decomposition approaches to tackle the problem. The first approach reduces the multi-facility location problem for N new facilities to N single-facility location problems of the same type by fixing the assignment variables in the problem formulation to 1. The second approach, on the other hand, keeps the location variables constant and benefits from the set partitioning of the feasible domain based on visibility properties. In the latter case, they restrict each new facility to one of the candidate domains of the feasible space which could be deemed as the extension of the reduction results of [153] to multiple new facilities. These decompositions result in a finite number of mixed-

integer programming sub-problems. They finally apply a genetic algorithm heuristic to solve the two problems.

3.1.2.4 Heuristic methods

Heuristic techniques are widely used to address the cable harness routing and similar problems since they are capable of handling the highly nonconvex search space of the routing problem [161].

Of the early works on routing cable harnesses, Conru's and Cutkosky's method for concurrent design of cable harnesses using heuristics drew attention [162]. After voxelizing the workspace, to make the feasible space of the optimization problem convex, they initially neglect the obstacles and find a globally optimal solution for the locations of the harness transitions (breakouts). Next, if any of the transitions are placed in the obstacle space, it must be moved to the closest cell in the free space. Then, a heuristic path planning method locally optimizes the path between the endpoints of the cables and transitions. However, the final path may still not be optimal due to the local optimizations, and further human input is required to reroute the harness that is stuck in a local optimum. Additionally, some case-specific constraints such as minimum bend radii may not have been considered in the initial optimization problem and human user needs to take those into account to make the final solution feasible. Thus, human interaction is crucial in this method to guarantee the globality of the optimal solution.

In another study by Conru [161], a genetic algorithm (GA) is utilized to route the bundles and locate the transitions between the end connectors that define the connection points on the components. The algorithm starts with an initial configuration for the harness

which includes the connection information between the nodes (nodes are the end connectors and transitions). Assuming the free space graph is known, Dijkstra's algorithm is applied to generate the shortest route for the wire between each pair of the desired connectors. After the shortest routes are generated, an objective function is defined that minimizes the total cost of all the bundles consisting of a number of wires. GA is deployed to locate the transitions optimally using mutation and crossover operations on the initial configuration. After the optimal locations of the transitions are found locally, the algorithm explores the other configurations using another GA to develop close-to-global optima. Hence, the problem is decomposed into two domains and GA is applied to each to find the optimal solution.

In another study [163], Kimura employed a GA technique to address the problem of finding an optimal arrangement for ship pipes with branches. He simplified the problem by removing the branches and considering them as equipment in the design space instead.

Zhu et al. [164] have also innovated an approach to integrate optimization and knowledge-based engineering to optimize the location and number of harness breakouts. They proposed a two-step optimization method: initialization step, which benefits from a roadmap path planning to define an initial configuration for the harness, and a refinement step, which refines the locations to further improve the solution and satisfy all constraints. The initialization is solved as a bi-level optimization problem since the problem is multi-destination path planning: a branch level and a harness level. The branch level finds the shortest path for each branch on a predefined roadmap using the A* algorithm on a predetermined grid. In the harness level, Hill Climbing heuristic is deployed to locate the

harness breakouts. To eliminate the likely violations of the constraints at the initialization step and improve the near-optimal solution achieved at initialization, the initial harness configuration is refined using Generalized Pattern Search (GPS) optimization.

Most of the research studies done on the routing and design of cable harnesses consider cables as a series of rigid segments. However, Kabul et al. argued that in addition to geometric and collision constraints, physical and mechanical constraints of the cables need to be accounted for to obtain a more realistic routing solution [165]. They, consequently, asserted that cable must be considered as a deformable body for which a motion needs to be planned. Taking the functional and manufacturing constraints noted by Kabul et al, Hermansson et al. [166] presented a heuristic grid-based method for routing of flexible 1D components in three-dimensional space.

3.1.3 Comparison of the methods

To summarize, all the related work on the study of multipath connection systems including but not limited to cables and pipes classify into two main categories: design-related research and optimization-related research. The design-related research primarily focuses on the design process that leads to the final layout for the connectors. Researchers over the past few decades have developed design tools such as CAD and computer-based models, virtual reality environments, and design guidelines that can assist designers in their decision-making pertinent to the selection of sizes and routes for multiple connectors in a densely populated region. Additionally, design methods such as case studies were followed to further investigate the industrial design of such systems in order to make improvements to the practiced processes. Regardless of all the efforts, the developed tools still require

different levels of human intervention and thus the process lacks automation. For example, as noted by Ng et al. [4], cable lengths, paths, and location of breakouts are decided based on trial-and-error using physical prototypes in final stages of design (detail design stage). More importantly, the design-based methods may not yield a final optimal layout which could bear significant costs for the manufacturing and maintenance of the cables or pipes [4].

Unlike the design-based methods, the optimization methods are mainly concerned with optimizing the layout of the connectors though some of the proposed methods may not apply to all real-world problems in practice, as claimed in [89]. Of the relevant studies, tree-based methods have gained popularity in designing interconnected networks. Minimal Steiner trees, in particular, are extensively employed to address problems where adding extra nodes to a network is allowed to further minimize its total length. This fact makes the method a well-suited candidate for cable/pipe routing problems where branching is permitted. The original Steiner tree, however, does not deal with obstacle-avoiding constraints; hence, researchers have to make modifications to adopt the method for cable/pipe routing in the presence of obstacles. In fact, adding obstacles to the environment of a Steiner tree significantly increases the complexity of the problem [133]. Therefore, the research conducted to address these problems is limited to the use of approximations and heuristic to find an optimal solution. Although exact solution methods are proposed [134,135], they generally are computationally expensive and may not apply to large scale problems without using any approximations. Hence, the obstacle-avoiding Steiner tree may not be an efficient solution to the cable/pipe layout optimization problem.

Design and optimization of wind farm layout could be deemed analogous to cable harness layout problems as both may be simplified to a network of connected nodes. Wind farm layout design is mainly solved using MIP models. Often, the planar workspace of the problem is discretized to a grid. With a known number of wind turbines, their optimal locations are assigned from the grid points by solving the MIP. This is, however, unlikely to occur in the cable harness layout problem as the locations of the components need to be connected are known a priori. Further, the wind farm layout problem has multiple *Start* nodes but only one *Goal* node, known as the station, where all the wind turbines are connected. The cable harness layout problem, on the other hand, can have multiple *Start* and multiple *Goal* nodes connected via breakouts. Since not all the physical constraints of the cable layout problem may be mapped to the wind farm layout optimization problem, the corresponding solution methods are not further considered for potential applications to the cable layout optimization problem.

When the focus in the cable layout problem is shifted from the length of the cables to the determination of the optimal location of the cable breakouts, an immediate set of candidate methods can be considered from the Location Theory. Location problems in the presence of obstacles have been among the challenging NP-hard problems in operations research[152]. Though many solution methods are presented over the past four decades, they still cannot address the problem in its entirety. For example, the methods can only deal with convex obstacles [150,151,167], since the objective function is non-convex, the discretization of the workspace is used [152] which results in locally optimal solutions, and

finally, the bi-objective multi-facility problem in presence of freeform objects remains unsolved.

Last but not the least, heuristic methods are widely applied to solve different instances of multipath planning problems with branches due to their efficiency in solving NP-hard problems, although the solutions found are not necessarily global. Table 3.1 summarizes the efforts in the design and optimization of multipath connectors applicable to cable harness layout optimization problem.

The review of the literature shows a scarcity of research efforts in developing computationally efficient methods to tackle optimization of cable harness layout in presence of freeform objects to global optimality. Additionally, there exist few studies that consider other objectives besides the minimization of the total length of the cable layout.

Apart from the limitations, it is understood that the chosen optimization method highly depends on the specifics of the problem which stems from its real-world application. For example, the constraints of cable harness layout optimization are different from wind farm design and pipe routing in ships. Hence, the problem must be well-defined in terms of its constraints and criteria to be aligned with its application so that the algorithm is practical for real-world problems and could assist designers in their decision making regarding the selection of connectors in a complex interconnected system.

By this background, the objectives of the first part of the present study are outlined in the next section.

Table 3.1 Comparison of design and optimization methods for multipath connection problems

Classification	Reference	Contributions	Limitations
Design tools (CAD)	[87–91]	- human-computer interface for designers - geometric kernel for modeling cables/pipes	- Sub-optimal solutions - lacks automation - Based on trial-and-error
Design tools (VR)	[4,86,94,95,97,98,168]	- Consideration of human expertise in design - Design automation	- Sub-optimal solutions - Designer-dependent
Design Heuristics	Design guideline [93,169]	- Instructions for cost minimization for wire routing and sizing	- Sub-optimal solutions
	Knowledge-based and concurrent engineering [3,99,101,102,104]	- Value human knowledge in design	- Sub-optimal solutions - lacks automation
Optimization-Obstacle-avoiding Steiner/spanning trees	Winter [133]	- introduction of Steiner visibility - problem breakdown into subproblems	- approximate solution - convex polygonal obstacles only
	Zachariasen and Winter [134]	- exact visibility-based method for subtree problem	- computationally expensive
	Parque and Miyashita [136]	- Steiner tree with n-star topology	- known topology, not applicable to the general layout design problem
Optimization-Location theory	Katz and Cooper [150]	- first to consider obstacle in location problems	- only one circular obstacle considered
	Aneja and Parlar[151]	- multiple obstacles	- applicable to single-facility only
	Klamroth et al. [152–154,158,160]	- new distance metric - discretization of workspace - multi-facility	- local optimal - convex obstacles only
Heuristic optimization	Conru and Cutkosky [170], Kimura[163], Zhu et al. [164]	- Computationally efficient in solving NP-hard problems	- Sub-optimal solutions

3.2 Research objective and proposed solution

The limitations of the existing methods in addressing cable harness layout optimization in its general form, drive the first part of this research to explore optimal

solutions to the following problem: *For a given number of start and goal points that connect different components in a cluttered environment using flexible connectors (e.g. wires), a layout is to be found for the connectors defined by their routes and the locations of a finite number of breakouts to minimize the total lengths of needed connectors while maximizing their commonality such that the connectors do not cross any objects and the breakouts are not placed inside an occupied area.*

The optimization objectives are set to minimize the cost of the wiring connection systems while providing more accessibility and traceability for maintenance purposes through maximizing the common length of the connectors (or bundling as many connectors as possible for the longest possible distance).

The goal is to provide this insight for the designer at any stage of design by being able to run the algorithm and based on the outcome, make appropriate recommendations regarding the final layout of cable connectors. The underlying assumptions based on which the problem needs to be formulated and solved are as the following:

- The problem is modeled on a **2D plane**.
- Since the wiring connectors are flexible, the **Euclidean** distance metric is used to calculate distances between the points in the plane.
- Obstacles are **arbitrary polygons** scattered on the plane.
- The cartesian coordinates of the nodes that need to be connected are given.
- The number of required breakouts is prespecified.

In addition, the problem is bi-objective and constrained, and the decision variables of the optimization problem are the cartesian coordinates of the breakouts.

The first part of this research answers the question: How can this bi-objective nonlinear optimization problem be solved without approximating the lengths of cable routes?

Two approaches are proposed to address this research question. First, looking at the limitations of the existing methods to tackle the location problem in presence of obstacles without approximating the distances (e.g. using polyhedral gauges), this work investigates the possibility of formulating the cable harness layout as bi-objective location problem in presence of obstacles using Euclidean norm and solving the problem with a suitable optimization method. Second, this study aims at investigating the potential of the convex hull based routing, introduced in Chapter 2, in solving the cable harness layout as a multipath planning problem with two objectives. The efficiency of this method in finding the shortest path between any two points of a cluttered planar environment is shown in the previous chapter. In this chapter, its extension and application to multipath planning problems with more than one objective are further discussed.

The remainder of this chapter is allocated to the explanation of the two approaches proposed to address the cable harness layout optimization problem as well as a discussion on the results of applying the methods to sample problems.

3.3 Mixed-binary layout optimization using Euclidean norm

As discussed in the previous section, the goal is to develop an algorithm to find the optimal layout of a cable harness assembly by finding the optimal location(s) of the breakout(s). The problem, therefore, becomes analogous to the well-known Weber's problem of locating a new facility in the vicinity of existing facilities and outside forbidden

regions to achieve minimum traveling cost or other objectives (e.g. maximum distance from existing facilities).

One challenge of the location problem in the presence of obstacles is that the distances between the nodes that are not visible to each other changes and the conventional Euclidean norm can no longer be used to determine such distances. To overcome this challenge, Klamroth has introduced polyhedral gauges that approximate the distance between two points not visible to each other [152]. This approximation, however, affects the final optimal solution.

That said, the objective of this section is to further investigate the possibility of formulating the objective functions of the cable harness optimization problem explicitly in terms of Euclidean norm and to solve the formulated optimization problem. The notion of visibility is utilized in defining the objective functions as discussed in the next section.

3.3.1 Visibility map for location-allocation

When an object blocks the direct path between a pair of points in an environment, the traveling distance between them also changes and a waypoint (or a series of waypoints) needs to be located in the unoccupied region to enable traveling from one point to the other. The direct path, as a result, is broken into segments between the found waypoints, *Start,* and *Goal*. The locations of these waypoints highly affect the distance to be traveled to reach the goal point or a node.

Thus, the presence of an obstacle decomposes the free space into areas that are either visible or invisible with respect to each node. Knowing to which of these areas the *Start*/*Goal* node(s), the breakouts, or the waypoints belong, helps to determine the distance

between the points. For example, the location of a breakout is to be found for the cable harness of Figure 3.11 with one *Start* node and two *Goal* nodes while avoiding its placement on and traveling through the line barrier, $\overline{O_1O_2}$. The objectives are to minimize the overall distances between the respective start and goal nodes and maximize the common length of wires between the *Start* node and the breakout. As seen in this figure, the presence of the line barrier divides the workspace into two regions based on the visibility of points with respect to one another.

The decomposition is inspired by Klamroth's [153] subdivision using the shadows of the existing nodes (here S_1, G_1, and G_2). Looking at Figure 3.11, the shadow of each node is outlined with dashed lines. In addition, the convex hull of the nodes and the intersecting obstacle is shown in a solid blue line to specify the bounded region inside which the breakout must be located based on Klamroth's proof.

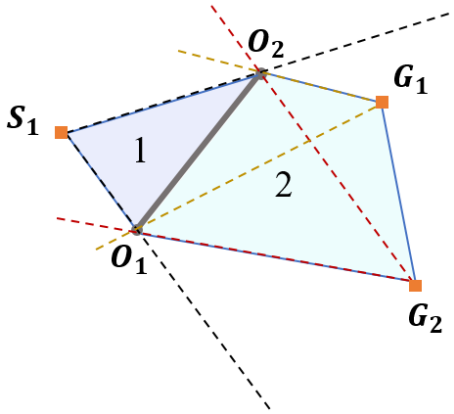


Figure 3.11 Sample subdivision using shadows of existing nodes

This subdivision based on visibility is then used to define a set of objectives and constraints per region. That is, depending on the region where the breakout is placed, the distances can be calculated and optimized. For instance Figure 3.11 shows that every point

in region 1 is visible to S_1 but invisible to G_1 and G_2. Vice versa, every point in region 2 is invisible to S_1 but visible to G_1 and G_2. We call this decomposition of the workspace on the grounds of visibility of the nodes, the *visibility map* of the workspace with respect to the breakout. The table below summarizes the visibility information based on the visibility map of Figure 3.11. The checkmark is for visible and the cross mark is for the invisible locus with respect to each node in the top row.

Table 3.2 Summary of visibility information for Figure 3.11

Breakout location	S_1	G_1	G_2
Region 1	✓	×	×
Region 2	×	✓	✓

It is noteworthy that the location of the existing nodes highly affects the subdivision of the feasible domain. Suppose, for instance, that the three nodes of Figure 3.11 were located as in Figure 3.12. The difference between the figures is that the node G_1, previously inside the shadow of S_1, now lies outside this shadow which creates more regions in the feasible domain based on the visibility information of Table 3.3.

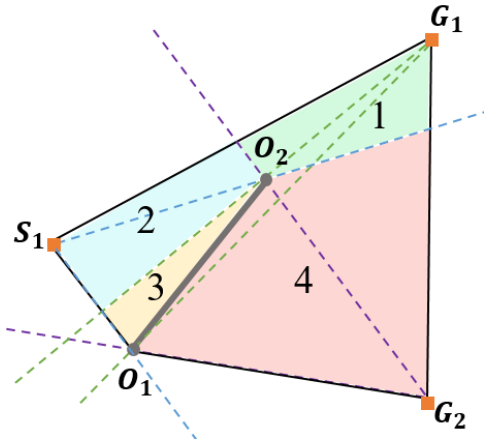


Figure 3.12 Effects of node locations on the subdivision of the workspace

Table 3.3 Summary of visibility information for Figure 3.12

Breakout location	S_1	G_1	G_2
Region 1	✓	✓	✓
Region 2	✓	✓	×
Region 3	✓	×	×
Region 4	×	✓	✓

The visibility map of the workspace enables defining the objective function(s) explicitly using the Euclidean norm by introducing binary variables. Two sets of binary variables are introduced to formulate the problem based on a visibility map: the first set is used to activate the region housing the optimal location of the breakout and the second is used to activate the potential waypoints where the optimal path needs to make a turn to avoid an obstacle.

For example, for the workspace of Figure 3.11, two binary variables, w_1 and w_2, are required to denote which region is activated to yield the optimal location of the breakout. Binary variables are deployed since they can serve as on/off switches which activate/deactivate a region if the value of the variable is equal to 1/0.

Additionally, due to the presence of the line barrier in Figure 3.11, a waypoint is required to facilitate travel from the *Start* node to either of the *Goal* nodes. The optimal locations of the waypoint are the two ends of the line barrier, O_1 and O_2. Depending on which endpoint is decided in the final optimal solution, binary variables, y_i, can be introduced to reflect this decision and the calculation of the Euclidean distances. The problem can now be formulated as in **Problem 1**.

Problem 1

$$\min_{X\in\mathbb{R}^2} \mathrm{Z}_1 = wD_1 + (1-w)D_2 \ ,$$

$$\max_{X\in\mathbb{R}^2} \mathrm{Z}_2 = w\|S_1, X\| + (1-w)\left[y_3\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_3)\left(\|S_1, O_2\| + \|O_2, X\|\right)\right]$$

$$D_1 = \|S_1, X\| + y_1\left(\|X, O_1\| + \|O_1, G_1\|\right) + (1-y_1)\left(\|X, O_2\| + \|O_2, G_1\|\right)$$
$$+ y_2\left(\|X, O_1\| + \|O_1, G_2\|\right) + (1-y_2)\left(\|X, O_2\| + \|O_2, G_2\|\right)$$
$$D_2 = y_3\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_3)\left(\|S_1, O_2\| + \|O_2, X\|\right) + \|X, G_1\| + \|X, G_2\|$$

$$S.t. \quad X \notin \overline{O_1O_2}$$
$$X \in C$$
$$X \in \mathbb{R}^2$$
$$y_i, w \in \{0,1\}, \quad i = 1,2,3$$

Where
X : the breakout location in the plane;
C : the convex hull of the set points S, G, and the intersecting obstacles.

In **Problem 1**, the first objective is to minimize the total distances between the nodes and the breakout. Two functions, D_1 and D_2, are defined, respectively for regions 1 and 2, to calculate the total lengths of wires. It is clear that distances change as the location of the breakout changes from region 1 to region 2, which entails the introduction of D_1 and D_2 (e.g. D_1 must be used if the breakout is located in region 1). The binary variable, w, serves as a switch for region selection in this problem. For example, if the breakout is placed in region 1, w activates D_1, that is $w = 1$, and deactivates D_2, and vice versa.

It should be noted that two binary variables, w_1 and w_2, are required to switch the distance metrics on/off. However, since at any time only one location for the breakout is plausible, only one variable can become active, therefore, $w_1 + w_2 = 1$. To minimize the number of variables used in the optimization problem, the relationship between the two

binary variables is taken advantage of and one variable is written in terms of the other, $w_1 = 1 - w_2 = w$.

As can be seen in **Problem 1**, all the distances are calculated using the Euclidean norm. In D_1 distance function, the first term is to calculate the distance between the *Start* node and the breakout, *X*. The second term in this function benefits from the introduction of a new binary variable, y_1, which indicates which route is taken to reach the first *Goal* node. Since the *Goal* nodes are in areas invisible to any point in region 1, there needs to be a waypoint to facilitate traveling to the *Goal* nodes. Two routes are conceivable to reach the *Goal* nodes, one that passes from O_1 and the other that passes from O_2 (for the proof that these points yield the optimal solution, please refer to [83]). If, in the second term of D_1, $y_1 = 1$, the route that passes from O_1 is activated which deactivates the path with the waypoint at O_2. On the contrary, if $y_1 = 0$, the path that passes from O_2 becomes activated (third term). The same rationale is used to add the fourth and fifth terms to D_1 by introducing another binary variable that switches between the two possible routes to G_2.

As discussed, the second distance function is activated in the objective function when the breakout is in region 2. Locating the breakout in region 2 makes it invisible to the *Start* node. Therefore, a turning point must be selected (similarly at O_1 or O_2) to enable traveling from S_1 to G_1 or G_2 which results in the introduction of the third binary variable that works similarly to y_1 and y_2 and forms the first two terms in D_2. The last two terms in this function calculate the distances from the breakout to either of G_1 and G_2 both of which are visible from *X*.

The second objective function is to maximize the common length, here the distance between S_1 and X. It is observable in the formulation of **Problem 1** that the choice of the region for placing the breakout as well as the routes to invisible points are reflected in the terms of the maximization function by the binary variables.

The constraints force the breakout to lie inside the convex hull of the nodes and the barrier, but outside the barrier. Further, the decision variables are X, the cartesian coordinates of the breakout in the plane, and all of the binary variable, w and y_i.

Following the same procedure and based on the visibility map of the workspace, **Problem 2** is formulated for the cable harness in Figure 3.12. Since the visibility map of Figure 3.12 has four regions, four binary variables, w_i , $i = 1, \dots, 4$, are required to activate one and deactivate the other three at each time. In addition, the second set of binary variables, y_j , $j = 1,2,3$, is used to activate/deactivate the waypoints to be passed to reach the nodes in the invisible regions.

Problem 2

$$\min_{X \in \mathbb{R}^2} Z_1 = \sum_{i=1}^{4} w_i D_i \ ,$$

$$\max_{X \in \mathbb{R}^2} Z_2 = \left[\sum_{i=1}^{3} \left(w_i \left\| S_1, X \right\| \right) \right] + w_4 \left[y_3 \left(\left\| S_1, O_1 \right\| + \left\| O_1, X \right\| \right) + (1 - y_3) \left(\left\| S_1, O_2 \right\| + \left\| O_2, X \right\| \right) \right]$$

$$D_1 = \left\| S_1, X \right\| + \left\| X, G_1 \right\| + \left\| X, G_2 \right\|$$

$$D_2 = \left\| S_1, X \right\| + \left\| X, G_1 \right\| + y_1 \left(\left\| X, O_1 \right\| + \left\| O_1, G_2 \right\| \right) + (1 - y_1) \left(\left\| X, O_2 \right\| + \left\| O_2, G_2 \right\| \right)$$

$$D_3 = \left\| S_1, X \right\| + y_2 \left(\left\| X, O_1 \right\| + \left\| O_1, G_1 \right\| \right) + (1 - y_2) \left(\left\| X, O_2 \right\| + \left\| O_2, G_1 \right\| \right)$$
$$+ y_1 \left(\left\| X, O_1 \right\| + \left\| O_1, G_2 \right\| \right) + (1 - y_1) \left(\left\| X, O_2 \right\| + \left\| O_2, G_2 \right\| \right)$$

$$D_4 = y_3 \left(\left\| S_1, O_1 \right\| + \left\| O_1, X \right\| \right) + (1 - y_3) \left(\left\| S_1, O_2 \right\| + \left\| O_2, X \right\| \right) + \left\| X, G_1 \right\| + \left\| X, G_2 \right\|$$

$$S.t. \quad X \notin \overline{O_1 O_2}$$

$$X \in C$$

$$X \in \mathbb{R}^2$$

$$\sum_{i=1}^{4} w_i = 1$$

$$w_i, y_j, \in \{0,1\}, \quad i = 1,...,4, \; j = 1,2,3$$

Where

X : the breakout location in plane;

C : the convex hull of the set points S, G, and the intersecting obstacles.

A more complex example with a triangular obstacle is shown in Figure 3.13 with its visibility information summarized in Table 3.4.

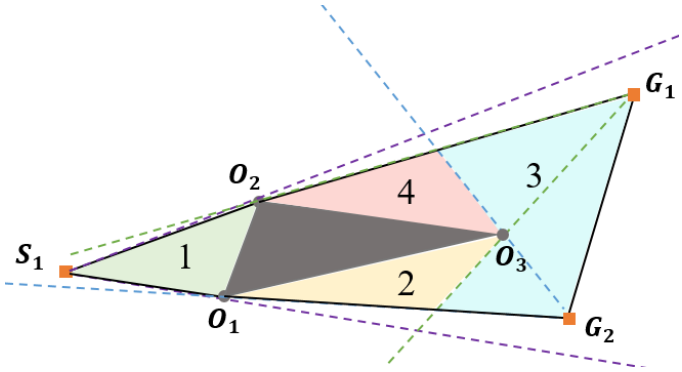


Figure 3.13 Sample visibility map for workspace with one triangular obstacle

Table 3.4 Summary of visibility information for Figure 3.13

Breakout location	S_1	G_1	G_2
Region 1	✓	×	×
Region 2	×	×	✓
Region 3	×	✓	✓
Region 4	✓	✓	×

As seen in Figure 3.13 and Table 3.4, four regions are created based on the visibilities of the existing nodes with respect to the breakout. Note that in Figure 3.13, multiple paths are conceivable to reach the breakout from S_1 depending on the waypoint(s) taken to reach the breakout; hence, the distances can change. As a result, region 3 needs to

be further decomposed to areas inside each the distance from S_1 to breakout is consistent. This second level of decomposition is shown in Figure 3.14.

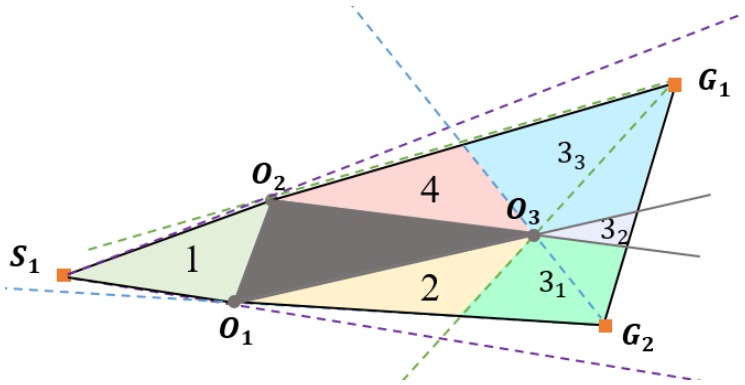


Figure 3.14 Level 2 decomposition of the workspace of Figure 3.13

Looking at Figure 3.14, it is discernable that for the breakout in region 3_1 the path from S_1 to X passes through $\boldsymbol{O_1}$. Even though another route is feasible through $\boldsymbol{O_2}$ and then $\boldsymbol{O_3}$, this route is longer and therefore discarded from the formulation of the optimization problem. It is also evident that the distance from S_1 to X is different in the region 3_2 than in the region 3_1. This difference comes from the visibility of the waypoints $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$ from X in different subareas of region 3. For example, X in region 3_1 sees $\boldsymbol{O_1}$ but not $\boldsymbol{O_2}$ while X in region 3_2 can see both $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$. Therefore, two paths from S_1 to an X in 3_2 are plausible without clear superiority of one over the other (unlike the two paths from S_1 to an X in 3_1). The situation in the region 3_3 is closer to that of 3_1's where the route traveling from S_1 to $\boldsymbol{O_2}$ to X is clearly shorter than the path from S_1 to $\boldsymbol{O_1}$ to $\boldsymbol{O_3}$ and then X.

Following the same logic in formulating **Problem 1** and **Problem 2** and using the visibility map of Figure 3.14, a formulation of the optimization problem is provided as in **Problem 3**.

Problem 3

$$\min_{X\in\mathbb{R}^2} Z_1 = \sum_{i=1}^{6} w_i D_i \ ,$$

$$\max_{X\in\mathbb{R}^2} Z_2 = \left[\sum_{i=1,6}\left(w_i \left\|S_1, X\right\|\right)\right] + \left[\sum_{i=2,3} w_i \left(\left\|S_1, O_1\right\| + \left\|O_1, X\right\|\right)\right]$$
$$+ w_4\left[y_1\left(\left\|S_1, O_1\right\| + \left\|O_1, X\right\|\right) + (1-y_1)\left(\left\|S_1, O_2\right\| + \left\|O_2, X\right\|\right)\right] + w_5\left(\left\|S_1, O_2\right\| + \left\|O_2, X\right\|\right)$$

$$D_1 = \left\|S_1, X\right\| + \left(\left\|X, O_2\right\| + \left\|O_2, G_1\right\|\right) + \left(\left\|X, O_1\right\| + \left\|O_1, G_2\right\|\right)$$
$$D_2 = \left(\left\|S_1, O_1\right\| + \left\|O_1, X\right\|\right) + \left(\left\|X, O_3\right\| + \left\|O_3, G_1\right\|\right) + \left\|X, G_2\right\|$$
$$D_{3,1} = \left(\left\|S_1, O_1\right\| + \left\|O_1, X\right\|\right) + \left\|X, G_1\right\| + \left\|X, G_2\right\|$$
$$D_{3,2} = y_1\left(\left\|S_1, O_1\right\| + \left\|O_1, X\right\|\right) + (1-y_1)\left(\left\|S_1, O_2\right\| + \left\|O_2, X\right\|\right) + \left\|X, G_1\right\| + \left\|X, G_2\right\|$$
$$D_{3,3} = \left(\left\|S_1, O_2\right\| + \left\|O_2, X\right\|\right) + \left\|X, G_1\right\| + \left\|X, G_2\right\|$$
$$D_4 = \left\|S_1, X\right\| + \left\|X, G_1\right\| + y_2\left(\left\|X, O_1\right\| + \left\|O_1, G_2\right\|\right) + (1-y_2)\left(\left\|X, O_2\right\| + \left\|O_2, G_2\right\|\right)$$

$$S.t. \quad X \notin \Delta O_1 O_2 O_3$$
$$X \in C$$
$$X \in \mathbb{R}^2$$
$$\sum_{i=1}^{6} w_i = 1$$
$$w_i, y_j, \in \{0,1\}, \quad i = 1,...,6, \ j = 1,2$$

Where
X : the breakout location in plane;
C : the convex hull of the set points S, G, and the intersecting obstacles.

3.3.2 Results and discussion

The problems formulated in this section using the visibility map and binary variables can be solved with bi-objective optimization solvers that handle integer variables and nonlinear objective functions and constraints. A few solvers are developed that satisfy the aforementioned criteria to solve these problems. To the best of our knowledge, no software exists to solve this class of problems with exact optimization methods. Therefore,

a heuristic solver in MATLAB is sought to solve sample problems formulated in the previous section.

Since the objective functions of a bi-objective optimization problem conflict with each other, meaning the increase in the value of one function could cause a decrease in the value of the other and vice versa, the problem does not have a unique solution. Instead, the Pareto set, or the set of non-dominated solutions, is generated that shows the tradeoff between the values of the objective functions. A Pareto non-dominated solution, shown in Figure 3.15, is the one in which improving one objective requires degradation of the other.

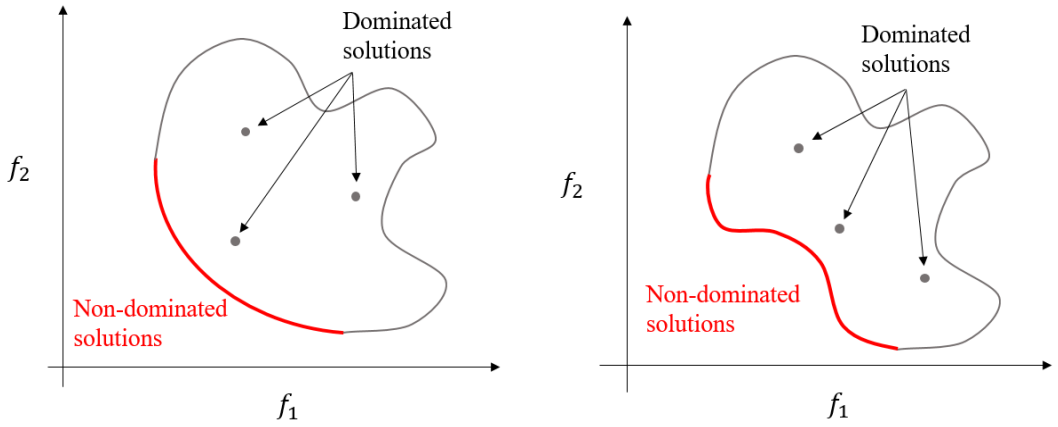


Figure 3.15 Examples of Pareto non-dominated solutions

MathWorks has released two multi-objective optimization solvers in MATLAB: ParetoSearch (PS) and Multi-Objective Genetic Algorithm (MOGA), both of which are heuristic-based and generate the set of non-dominated solutions. PS uses pattern search method on a set of points and iteratively searches for non-dominated points [171]. It requires an initial guess for the decision variables. MOGA, on the contrary, is developed based on Deb's NSGA-II [172], an elitist genetic algorithm. Unlike PS, MOGA creates a random initial population for the decision variables to be selected from. Some parameters

can affect the creation of the initial population, e.g. population size or initial population range. For a list of user-defined parameters please refer to [173]. For this research, the MOGA solver is selected to solve all of the bi-objective optimization problems. It is noteworthy that using a heuristic-based solver cannot guarantee to find the true Pareto set and one may only be able to obtain non-dominated solutions up to a known number of generations. For this reason, in the remainder of this manuscript, the outcome of the MOGA is referred to as non-dominated solutions, not Pareto set.

The default settings of MOGA do not allow having integer decision variables. Thus, in the properties function that MOGA solver reads, the initial population alongside the mutation and crossover functions are modified to accept binary variables[1]. A new set of constraints specifying the ids of the binary variables is added to the MATLAB functions of the initial population, mutation, and crossover. Also, in the main program, upper and lower bounds of 1 and 0, respectively, are added to specify the limits of the binary variables. The bounds as well as the modified functions are then sent to the solver to read and set up the variables accordingly during the optimization process.

In addition to setting up the variables, following **Problem 1**, separate MATLAB functions are created to quantify the constraints' violation and evaluate the objective functions. For the second objective function, which is the maximization of the common length, the negative of the distance between the *Start* node and the breakout is used. Since MATLAB's default definition of an optimization problem comes only with the

[1] All codes are written in MATLAB and can be accessed from: https://github.com/nmasoud/Routing-algorithms.git

minimization of a function, for maximization problems, the negative of the function is used to comply with MATLAB's default definition.

The GA parameters that affect the non-dominated solutions such as the population size and the number of generations must also be decided. For **Problem 1**, since the objective functions and constraints are rather simple (due to the few numbers of nodes and the presence of only one line barrier), the population size of 50 and 500 generations are considered in the MOGA solver. A sample workspace is generated to mimic the visibility map of Figure 3.11 wherein the coordinates of the *Start* and *Goal* nodes are $S_1 = (0,0)$, $G_1 = (6,2)$, and $G_2 = (8,-5)$. Additionally, a line barrier with endpoints located at $O_1 = (5,3)$ and $O_2 = (4,-4)$ is added to the workspace. Using the above-mentioned settings, **Problem 1** is solved in MATLAB via `gamultiobj` solver.

To solve this problem, the constraint of avoiding the placement of the breakout on the obstacle, $X \notin \overline{O_1O_2}$, is expanded and broken into two constraints that reflect the region the breakout belongs to as shown in **Problem 1-2**. Region 1 is to the left of the line and setting $w = 1$ activates it, while region 2 is to the right and *w* must be zero to activate it.

Problem 1-2

$$\min_{X\in\mathbb{R}^2} \mathrm{Z}_1 = wD_1 + (1-w)D_2 \ ,$$

$$\max_{X\in\mathbb{R}^2} \mathrm{Z}_2 = w\|S_1, X\| + (1-w)\left[y_3\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_3)\left(\|S_1, O_2\| + \|O_2, X\|\right)\right]$$

$$D_1 = \|S_1, X\| + y_1\left(\|X, O_1\| + \|O_1, G_1\|\right) + (1-y_1)\left(\|X, O_2\| + \|O_2, G_1\|\right)$$
$$+ y_2\left(\|X, O_1\| + \|O_1, G_2\|\right) + (1-y_2)\left(\|X, O_2\| + \|O_2, G_2\|\right)$$

$$D_2 = y_3\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_3)\left(\|S_1, O_2\| + \|O_2, X\|\right) + \|X, G_1\| + \|X, G_2\|$$

$$
\begin{aligned}
S.t.\quad & w(AX+b) \le \underline{0} \\
& (1-w)(-AX-b) \le \underline{0} \\
& X \in C \\
& X \in \mathbb{R}^2 \\
& y_i, w \in \{0,1\}, \quad i = 1,2,3
\end{aligned}
$$

Where
X : the breakout location in the plane;
C : the convex hull of the set points S, G, and the intersecting obstacles.

After `gamultiobj` solver is applied, the set of non-dominated solutions is generated. The solver stopped at 202 generations since the average change in the spread of the non-dominated solutions becomes less than the set tolerance. The final set of non-dominated solutions is shown in Figure 3.16, which corresponds to the objective space and the local optimal locations of the breakout (efficient solutions for the preimages of the non-dominated solutions) corresponding to each of the non-dominated solutions are shown in Figure 3.17. A colormap is used to map every solution in the objective space (Figure 3.16) to its relevant solution in the feasible space (Figure 3.17) using the same color. It can be seen from Figure 3.17 that all the optimal locations are in region 2 of the visibility map which increases the maximum common length.

Figure 3.18 shows the evolution of the non-dominated solutions from early generations to the final found at the 202nd generation. The solution set found at iteration (i+1)th dominates all the non-dominated points found previously at the 1st, 2nd, …, and ith generations.

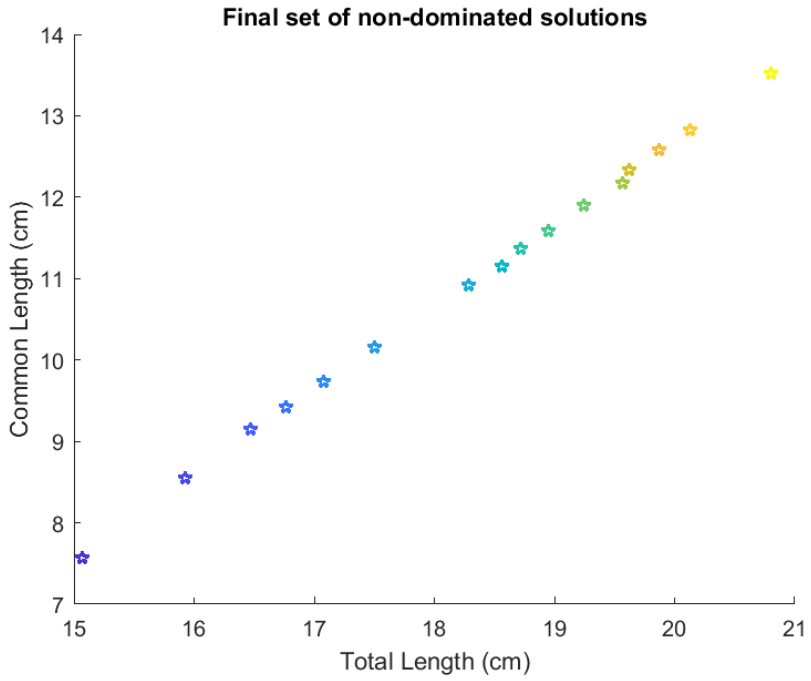


Figure 3.16 Final set of non-dominated solutions for Problem 1-2

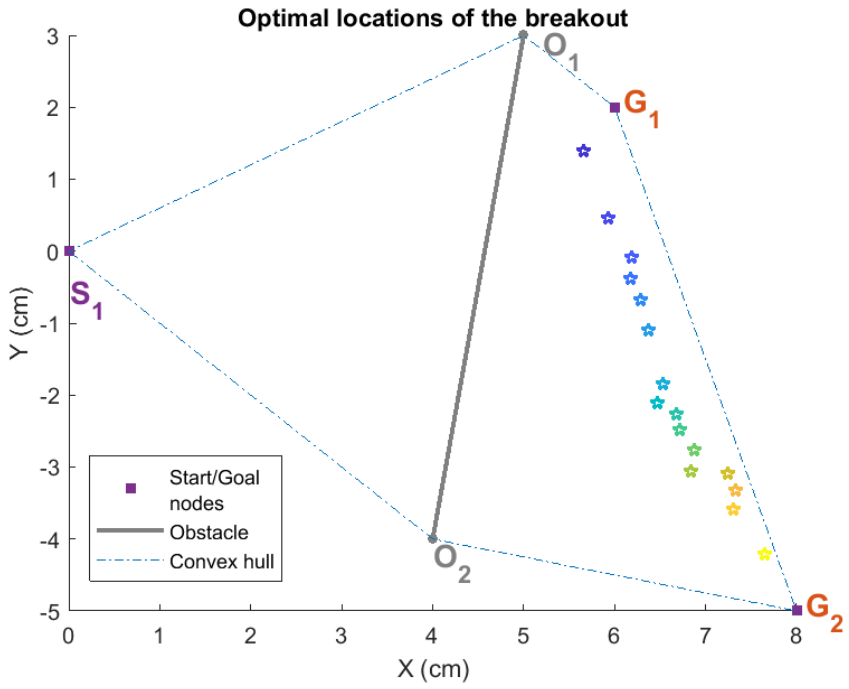


Figure 3.17 Optimal (efficient) locations of the breakout for Problem 1-2

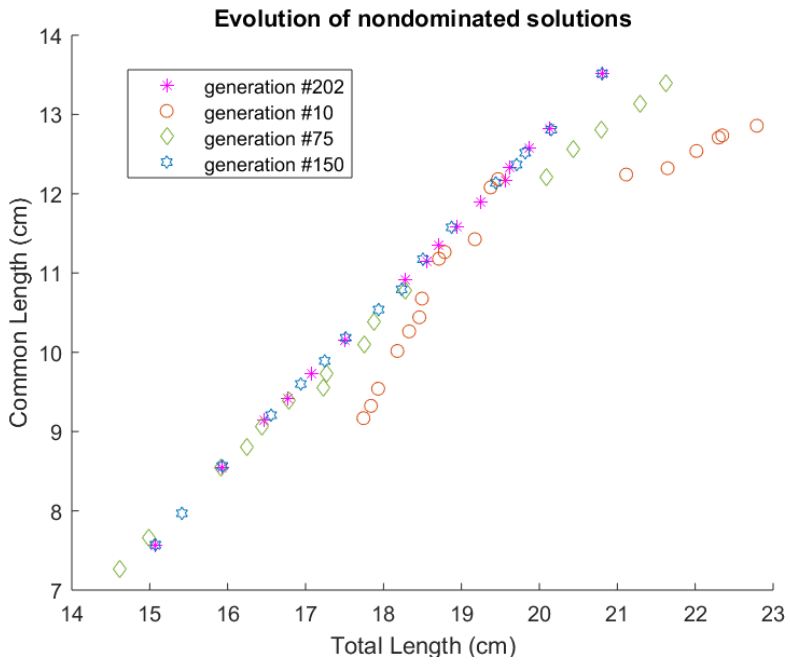


Figure 3.18 Evolution of non-dominated fronts

Additional details of the optimal locations and their corresponding optimal values of the objectives are provided in Table 3.5.

Table 3.5 Optimal values of decision variables and objective functions for Problem 1

Optimal breakout location coordinates, X^* (cm)	**Min total length with the breakout, Z_1^* (cm)**	**Max common length, Z_2^* (cm)**
(7.6456 -4.2147)	20.8059	13.5154
(5.6618 1.3906)	15.0731	7.5712
(7.6456 -4.2147)	20.8059	13.5154
(6.186 -0.0907)	16.474	9.1413
(6.1787 -0.3836)	16.7669	9.414
(6.6784 -2.266)	18.7143	11.358
(5.6618 1.3906)	15.0731	7.5712
(5.9235 0.4521)	15.9251	8.5411
(6.5375 -1.8523)	18.2815	10.921
(7.3274 -3.3309)	19.8693	12.5761
(6.3669 -1.0959)	17.4983	10.1489
(7.2494 -3.0952)	19.6214	12.3279
(6.8802 -2.7686)	19.244	11.8982

(6.4667 -2.1137)	18.5592	11.1508
(7.3011 -3.5989)	20.1334	12.8195
(6.2822 -0.6831)	17.0749	9.7308
(6.8447 -3.0693)	19.5636	12.1744
(6.7182 -2.4896)	18.9485	11.5831

A more complex example of a location problem in the presence of an obstacle is **Problem 3** where the line barrier is replaced by a triangular obstacle that increases the number of regions in the visibility map. In addition to the obstacle avoiding constraint presented in **Problem 1**, **Problem 3** has a linear equality constraint that imposes the sum of the binary variables attributed to the region selection to be equal to one. MATLAB's `gamultiobj` solver cannot handle linear equality constraints concurrent with integer variables. Therefore, an approach to solve the bi-objective problem by reducing it to a single objective problem must be followed. Two common methods of solving a multi-objective optimization problem by converting it to a single objective problem are weighted sum and ε-constraint.

The weighted sum method benefits from the introduction of a vector of weights multiplied by the objectives to convert the vectorized objectives to a scalar. The weights are chosen proportionately to the importance of the objective and their sum should be equal to one. Despite its simplicity, the weighted sum method has difficulty reaching the entire set of non-dominated solutions when the feasible domain is non-convex (like the non-dominated set in Figure 3.15, right). Therefore, a portion of the Pareto front would never be found with the weighted sum.

Unlike the weighted sum, the ε-constraint method, first introduced by Haimes [174], works with both convex and non-convex feasible sets and yields the Pareto set. The

method minimizes one of the objectives and expresses the other(s) in the form of inequality constraints (i.e. the value of objective i expressed in the constraints must be less than or equal to ε_i). Since the ε-constraint method has the advantage of obtaining solutions that are not reachable using the weighted sum, it is selected to solve **Problem 3**.

Similar to **Problem 1**, the obstacle-avoiding constraint, $X \notin \Delta O_1O_2O_3$, is further broken into six constraints to reflect each of the six regions the breakout can be located. The formulation of **Problem 3** is therefore updated as in **Problem 3-1.**

Problem 3-1

$$\min_{X\in\mathbb{R}^2} \mathrm{Z}_1 = \sum_{i=1}^{6} w_i D_i$$

$$D_1 = \|S_1, X\| + \left(\|X, O_2\| + \|O_2, G_1\|\right) + \left(\|X, O_1\| + \|O_1, G_2\|\right)$$

$$D_2 = \left(\|S_1, O_1\| + \|O_1, X\|\right) + \left(\|X, O_3\| + \|O_3, G_1\|\right) + \|X, G_2\|$$

$$D_{3,1} = \left(\|S_1, O_1\| + \|O_1, X\|\right) + \|X, G_1\| + \|X, G_2\|$$

$$D_{3,2} = y_1\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_1)\left(\|S_1, O_2\| + \|O_2, X\|\right) + \|X, G_1\| + \|X, G_2\|$$

$$D_{3,3} = \left(\|S_1, O_2\| + \|O_2, X\|\right) + \|X, G_1\| + \|X, G_2\|$$

$$D_4 = \|S_1, X\| + \|X, G_1\| + y_2\left(\|X, O_1\| + \|O_1, G_2\|\right) + (1-y_2)\left(\|X, O_2\| + \|O_2, G_2\|\right)$$

$$S.t. \quad \left[\sum_{i=1,6}\left(w_i \|S_1, X\|\right)\right] + \left[\sum_{i=2,3} w_i\left(\|S_1, O_1\| + \|O_1, X\|\right)\right]$$

$$+ w_4\left[y_1\left(\|S_1, O_1\| + \|O_1, X\|\right) + (1-y_1)\left(\|S_1, O_2\| + \|O_2, X\|\right)\right] + w_5\left(\|S_1, O_2\| + \|O_2, X\|\right) \le \varepsilon$$

$$w_i(A_i X + b_i) \le \underline{0}, \quad i = 1,...,6$$

$$X \in C$$

$$X \in \mathbb{R}^2$$

$$\sum_{i=1}^{6} w_i = 1$$

$$w_i, y_j, \in \{0,1\}, \quad i = 1,...,6, \; j = 1,2$$

Where

X : the breakout location in the plane;

C : the convex hull of the set points S, G, and the intersecting obstacles.

Following the ε-constraint method, the problem is converted to a constrained single-objective optimization problem with binary variables. The best solver in MATLAB that satisfies the requirements of **Problem 3-1**, is the GA solver. The magnitude of ε varies from 0.5 to 8.5 which is found based on testing the single objective of maximizing the common length. The optimal (efficient) locations of the breakout as well as the final set of non-dominated solutions are shown in Figure 3.19 and Figure 3.20 respectively.

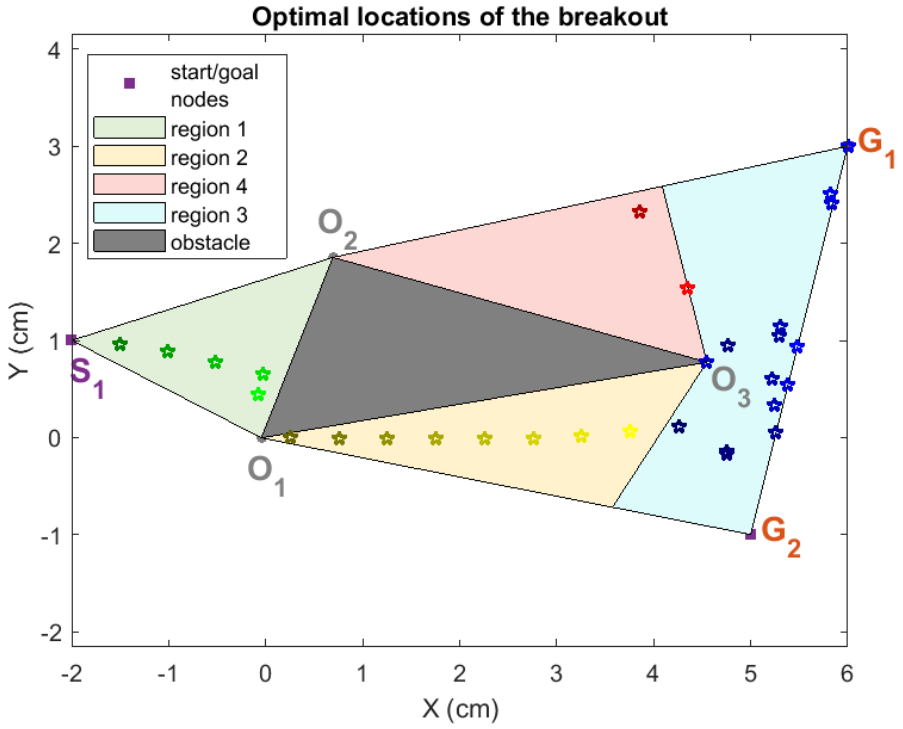


Figure 3.19 Optimal (efficient) locations of the breakout for Problem 3-2

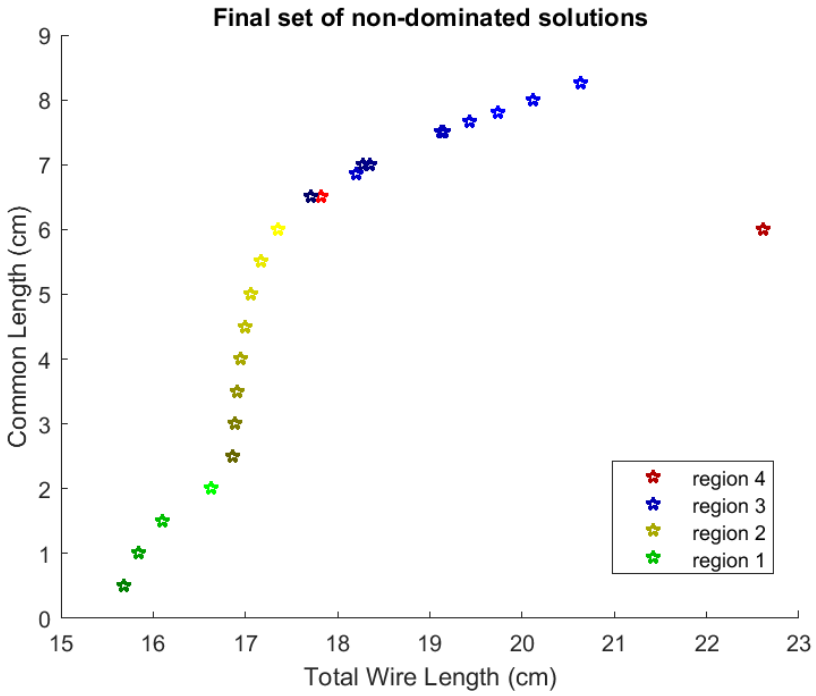


Figure 3.20 Set of non-dominated solutions for Problem 3-2

It is observed from Figure 3.19 and Figure 3.20 that the set of non-dominated solutions attributed to each of the four regions (color-coded in Figure 3.20) in the visibility map of the problem (Figure 3.13) is convex while the union of these sets shown in Figure 3.20 is non-convex. This behavior is caused by using binary variables to reflect the region selection in the location problem. Once a region is selected for locating the breakout and the corresponding binary variables are set, the problem, within the chosen region, becomes convex; thus, the found non-dominated set in the outcome space also becomes convex. However, the original problem described in **Problem 3-2** is a non-convex optimization problem. Therefore, when all the resulting non-dominated sets (created per each region) are combined to generate the overall set of non-dominated solutions, the outcome is a non-convex set as in Figure 3.20.

In addition, the numerical values of the optimal locations of the breakout as well as the two objectives can be found in Table 3.6.

Table 3.6 Optimal values of the decision variables and objective functions for Problem 3-2

ε	Optimal breakout location coordinates, X^* (cm)	Min total length with the breakout, Z_1^* (cm)	Max common length, Z_2^* (cm)
0.5	(-1.5033, 0.9526)	15.6877	0.499
1	(-1.0074, 0.8872)	15.8373	0.999
1.5	(-0.5177, 0.7766)	16.103	1.499
2	(-0.0322, 0.6479)	16.6294	1.999
2	(-0.0807, 0.4407)	16.6274	1.999
2.5	(0.2566, -0.0071)	16.8633	2.499
3	(0.7565, -0.0126)	16.8841	2.999
3.5	(1.2565, -0.0168)	16.9107	3.499
4	(1.7565, -0.0184)	16.9457	3.999
4.5	(2.2566, -0.0165)	16.9936	4.499
5	(2.7566, -0.0086)	17.0629	4.999
5.5	(3.2566, 0.0110)	17.1707	5.499
6	(3.8520, 2.3200)	22.6108	5.999
6	(3.7561, 0.0559)	17.3561	5.999
6.5	(4.2552, 0.1056)	17.7107	6.499
6.5	(4.3527, 1.5357)	17.8191	6.499
7	(4.7543, -0.1526)	18.2701	6.999
7	(4.7537, -0.1709)	18.27	6.999
7	(4.7663, 0.9454)	18.354	6.999
7.5	(5.2563, 0.0497)	19.1211	7.499
7.5	(5.296, 1.046)	19.1422	7.499
7.5	(5.2227, 0.5955)	19.136	7.499
7.5	(5.3112, 1.1379)	19.1438	7.499
7.5	(5.2462, 0.3285)	19.1249	7.499
8	(4.5487, 0.7756)	18.2051	6.8566
8	(5.8364, 2.4132)	20.1214	7.999
8	(5.3853, 0.5404)	19.4329	7.6549
8	(5.8259, 2.5028)	20.1237	7.999
8.5	(6.000, 3.000)	20.6322	8.2546
8.5	(5.4837, 0.9339)	19.7331	7.805

3.3.3 Final remarks

In this section, sample location problems are formulated using binary variables and visibility maps. Even though the method has the advantage of providing a formulation of the optimization function with explicit Euclidean distances between the points, the complexity of the problem formulation (which indicates the complexity of the solution) highly relies on the problem structure. For example, as discussed, a change in the locations of the existing nodes can completely change the visibility map of the workspace provided the geometry of the workspace remains unchanged.

In addition, it is shown that adding an obstacle or changing the shape of an obstacle can drastically increase the nonlinearity of the objectives and/or constraints which has a direct impact on the solution method. Therefore, this method is most efficient for workspaces with as few as one simple obstacle. Further, the obstacle must be polygonal and without any curved edges as having a curvature increases the nonlinearity of the constraints.

Apart from the geometric structure of the workspace of a location problem, care must be taken when formulating the problem using binary variables. For example, looking at Figure 3.20, an outlier is present in the set of non-dominated solutions with objective values of (22.611, 5.999). As seen in Figure 3.19, this point is located in region 4 of the visibility map. The reason why the total length of the harness is 22.611 by placing the breakout on this outlier is that the distance from S_1 to this breakout is calculated using the route passing from O_1 and O_3 instead of the shorter route passing from O_2. Although from the mathematical point of view this solution is feasible, it may not be realistic or optimal

from the design perspective. Hence, to avoid the attainment of such solutions and outliers in the non-dominated set, additional constraints can be introduced to the problem formulation to block the longer routes. If, however, more layouts are preferred to choose from, considering other physical constraints of the wiring harnesses (e.g. accessibility), solutions like this can remain in the non-dominated set and the constraints may not be modified in the problem formulation.

As future extensions of this work, the following research questions can be further investigated; (1) Is it possible to develop an algorithm that outputs the constraints and criteria of the problem using binary variables? (2) what is the effect of non-convex obstacles on the problem formulation and final optimal solutions? (3) can other criteria (e.g. minimizing the number of turns in the path) be added to the optimization problem?

3.4 Layout optimization using convex hull based routing

Although the method discussed in the previous section enables the formulation of the cable harness layout optimization problem with explicit objective functions, it may not be computationally efficient in solving complex problems where multiple freeform objects are scattered in the workspace. The convex hull based routing method explained in Chapter 2, on the other side, is proven efficient in generating the shortest collision-free path between any two points in a cluttered planar environment. This section further investigates the potential of this method in optimizing the layout of a cable harness assembly with the constraints and criteria outlined in section 3.2.

3.4.1 Problem formulation

Suppose a layout for a cable harness assembly needs to be generated to connect *n* components from a list of *Start* components to a *Goal* list of *m* components. It is assumed that two breakouts are required; the first is to bundle *n* wires from the *Start* list and extend to reach the second breakout, where the cables branch to reach the *m* components from the *Goal* list.

The constraints are to avoid crossing the obstacles and placing a breakout inside an obstacle. The objectives are (1) to minimize the total lengths of wires needed to connect all the components including the breakouts and (2) to maximize the length between the two breakouts for the longest possible commonality. The general mathematical formulation of this problem is provided in **Problem 4**.

Problem 4

$$\min_{B_1,B_1\in\mathbb{R}^2} Z_1 = \left(\left[\sum_{i=1}^{n} D(S_i, B_1) \right] + n_w D(B_1, B_2) + \left[\sum_{j=1}^{m} D(B_2, G_j) \right] \right), \ \max_{B_1,B_1\in\mathbb{R}^2} Z_2 = \left[D(B_1, B_2) \right]$$

$$S.t. \quad \mathrm{B}_1, \mathrm{B}_2 \notin \bigcup_{k=1}^{l} \mathrm{int}(P_k)$$

Where

B_1, B_2 : the two breakouts of the cable harness;

S_i : ith start point, $i = 1,2,\dots,n$;

G_j : jth goal point, $j = 1,2,\dots,m$; and

P_k : kth polygonal obstacle, $k = 1,2,\dots,l$; and

n_w: the number of wires passing through the length covered between B_1 and B_2.

$$D(a,b) = \begin{cases} \|a,b\| & \overline{ab} \cap (\mathrm{int} \bigcup_{k=1}^{l} P_k) = \varnothing \\ D(a,b) & otherwise \end{cases}$$

$D(a,b)$: the shortest distance between a and b calculated on from the route found by applying the C-hull based roadmap

In **Problem** 4, the minimization objective function has three terms: the sum of the distances between each start terminal and the first breakout, the distance between the two breakouts multiplied by the number of wires passing through it, and the sum of the distances between the second breakout and each of the goal terminals. The number of wires passing from B_1 to B_2, n_w, is found by taking the maximum of the number of *Start* and *Goal* nodes. In other words: $n_w = max\{|S|, |G|\}$, where $|\bullet|$ is the cardinality of a set. The decision variables are the (x, y) coordinates of the breakouts in $\mathbb{R}^2$ (plane). The constraints are to avoid locating a breakout inside a polygonal obstacle.

It should be noted that the breakouts might be located on the borders of an obstacle depending on the potential application of the optimization problem. It is also noteworthy that the constraint of having wires not cross the interior of any obstacles is implicitly addressed by calling the convex-hull based routing function when any two points are invisible to each other. Therefore, the explicit representation of this constraint in the optimization problem is not further provided.

The distance function, $D(\bullet,\bullet)$ shown in **Problem** 4 outputs the Euclidean distance, $\|\bullet,\bullet\|$, if the two points are visible to each other. Otherwise, the modified distance function, $\widetilde{D}(\bullet,\bullet)$, calculated based on the shortest collision-free path that the convex-hull based routing finds, is utilized.

The formulation shown in **Problem 4** requires the solver to search the entire feasible space which is the $\mathbb{R}^2$ plane, except the areas occupied by the obstacles, to find the optimal locations of the breakouts. This could significantly slow down the optimization process, especially for large-scale problems. Hence, it is recommended to adapt Klamroth's

iterative convex hull [158] to limit the feasible domain inside the convex hull created by the *Start* and *Goal* nodes. As explained previously, the boundary of this convex hull needs to expand iteratively by including obstacles crossing the convex hull boundaries, until all of the hull edges become collision-free. Using this idea, a new constraint is added to **Problem 4**, and the problem is reformulated as in **Problem 5**.

Problem 5

$$\min_{B_1,B_1\in\mathbb{R}^2} Z_1 = \left(\left[\sum_{i=1}^{n} D(S_i, B_1) \right] + n_w D(B_1, B_2) + \left[\sum_{j=1}^{m} D(B_2, G_j) \right] \right), \ \max_{B_1,B_1\in\mathbb{R}^2} Z_2 = D(B_1, B_2)$$

$$S.t. \quad \mathrm{B}_1, \mathrm{B}_2 \notin \bigcup_{k=1}^{l} P_k$$

$$\mathrm{B}_1, \mathrm{B}_2 \in C$$

Where

C : the convex hull of the set points *S*, *G*, and the intersecting obstacles.

$$D(a,b) = \begin{cases} \|a,b\| & \overline{ab} \cap (\mathrm{int} \bigcup_{k=1}^{l} P_k) = \varnothing \\ D(a,b) & otherwise \end{cases}$$

$D(a,b)$: the shortest distance between *a* and *b* calculated on from the route found by applying the C-hull based roadmap

In **Problem 5**, *C* is a convex polygonal region defined by its vertices and edges. To form this new constraint, a set of linear inequalities is added to dictate the location of the breakouts inside this convex hull.

3.4.2 Optimization solver

This problem can be formulated and set up in MATLAB as an optimization problem. In the main program, the workspace geometric data that includes the VRML data of the obstacles alongside the *Start* and *Goal* sets of nodes with their coordinates are taken as inputs. Next, the linear constraints that impose the breakouts to stay inside the

Klamroth's convex hull are created. The flowchart of Figure 3.21 describes the process used to create this convex hull.

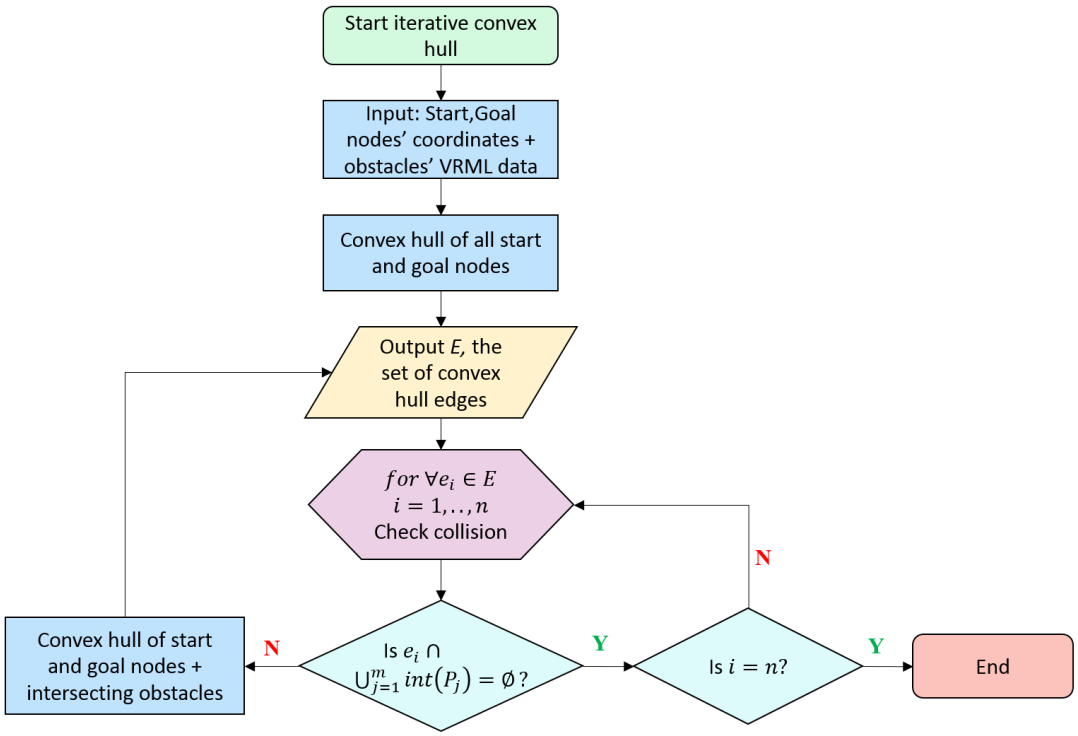


Figure 3.21 Flowchart for the iterative convex hull creation

In this flowchart, first, the convex hull of all the nodes in the *Start* and *Goal* sets is created using MATLAB's "`convhull`" function. Next, the edges of the convex hull are stored in the set E using their endpoints (denoted by their coordinates). Every edge in the set E is then checked for intersections with all the existing obstacles using the intersection detection algorithm developed in the convex-hull based roadmap [83]. If the edge is found crossing any of the obstacles, the corresponding obstacle is included to generate the updated convex hull. The process is continued until all the edges of the convex hull become collision-free. In the flowchart of Figure 3.21, P_j is the jth obstacle, where $j = 1, \dots, m$.

After the convex hull is created, its edges are extracted to define the linear constraints of the problem. These linear constraints specify a convex region inside which the breakouts can be located without the need to search the entire feasible region. Using this convex hull, the next step is to identify the obstacles that lie inside the convex hull. This information is to be passed to the nonlinear constraint function where the optimizer checks that the breakouts are not located inside or on the boundary of any obstacle (depending on whether the breakouts are allowed to be located on the boundary of a component or not). By determining the obstacles bounded inside the convex hull, the nonlinear constraint checks for every obstacle if the breakout is placed inside or outside this polygonal region.

A separate MATLAB function is created to set up the nonlinear constraints. These constraints are vectorized. For example, if l obstacles are identified inside the convex hull region, an $l \times 1$ vector is created that quantifies the output of the constraints using Boolean values. In more detail, if a breakout is located inside or on the boundary of obstacle k, $k \in \{1,2, \dots, l\}$, the value of the k^{th} row in the above-mentioned vector is 1; otherwise, it is zero. The pseudocode for setting up the nonlinear constraints as explained here is shown as in **Algorithm 3.1**.

Algorithm 3.1

Input: The set P of P_k, $k \in \{1,2,\dots,l\}$, the obstacles bounded inside the convex hull, and $X = [(x_1, y_1), (x_2, y_2)]$, the coordinates of the breakouts (the decision variables

Output: a Boolean vector *C*, showing which obstacles contain the breakout(s)

$C \leftarrow l \times 1$ vector of zeros

for ($k = 1$ to l), **do**:

  if $InPolygon(X, P_k)$ true

    $C_k \leftarrow 1$

  endif

end for

return *C*

This algorithm makes use of the `InPolygon` function [175] written by Redish and Jacquenot that detects if a set of points are inside a polygonal region. The function takes, as input, the coordinates of all the points to be checked and the vertices of the polygonal region in either clockwise or counterclockwise order.

Since the geometric data of the obstacles is provided in the tessellated format of VRML, the triangles that form each obstacle can be used as the set of polygonal regions. This may, however, increase the computation time as the algorithm needs to check every breakout point against every single triangle of an obstacle. Additionally, placing a breakout inside the convex hull of a non-convex obstacle may cause sharp and often undesirable turns of wires at these breakouts (see Figure 3.22).

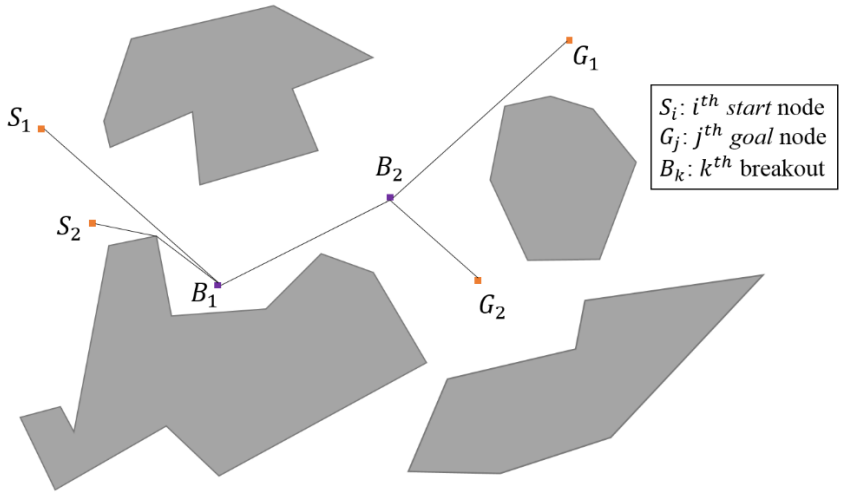


Figure 3.22 Example of a breakout located inside the convex hull of a nonconvex obstacle

To avoid these unwanted turns and to improve the computation time, instead of using the triangles in each obstacle as the polygonal regions, this study uses the convex hull of each obstacle as the polygonal region. We, however, recommend using the exact border of the nonconvex obstacle (or the triangles defining the shape) for densely populated workspaces where there may exist a *Start* or *Goal* node that is inside the convex hull of a nonconvex obstacle. This case is further discussed in section 3.4.3.

The output of the `InPolygon` function is a Boolean vector that shows whether any of the points is inside an obstacle. The code can be modified to output three types of vectors: strictly IN, which shows if a point lies in the interior of the polygon, IN/ON, which shows whether a point is in the interior or on the boundary of the polygon, and finally, ON, which turns to 1 if a point lies on the boundary of the polygon, not its interior. Since the purpose of this research is to avoid placing a breakout on a component of the workspace, the IN/ON check is used to output the nonlinear constraint value. The MATLAB code can,

however, be modified to use only the interior points such that placing a breakout on the boundary of a component is permitted. When searching for the feasible values of the decision variables, if any element in the *C* vector is found nonzero, the assumed decision variables become infeasible and must be excluded.

Lastly, the objective functions need to be set up in the optimization problem. For this purpose, another MATLAB function is created that outputs a vector of objective function values when the decision variables are inputted. **Algorithm 3.2** provides the pseudocode used to create this function.

Algorithm 3.2

Input: $X = [(x_1, y_1), (x_2, y_2)]$, the coordinates of the breakouts (the decision variables
Output: Z, a 2×1 vector of integer values for the two objective functions

$Z \leftarrow 2 \times 1$ vector of zeros
$L \leftarrow 0$
for ($i = 1$ to $|S|$), **do**:
  $L = L + D(S_i, B_1)$
end for
$L = L + n_w D(B_1, B_2)$

for ($j = 1$ to $|G|$), **do**:
  $L = L + D(B_2, G_j)$
end for
$Z_1 \leftarrow L$
$Z_2 \leftarrow -D(B_1, B_2)$
return Z

Following **Problem 5**, the first objective, the total lengths of wires, is decomposed into three segments: the length between each *start* node and the first breakout, the length between the two breakouts, and the length between the second breakout and each *goal* node. Analogous to the mixed-binary optimization, for the second objective function,

which is the maximization of the common length, the negative of the distance between the breakouts is used.

After the objective and constraint functions are set up correctly in MATLAB, a solver should be called to solve the optimization problem. Since the two objective functions in the bi-objective optimization problem of **Problem 5** conflict, it is expected to obtain a Pareto set of optimal solutions instead of a single value for the optimal functions.

The present problem is NP-hard with nonconvex constraints and criteria; hence, hardly could it be solved using an exact solution method. Even if an exact method exists to solve this problem, it would not be computationally efficient. Therefore, we need to resort to heuristic techniques. Though they may not be the best approach in finding the global solution, their efficiency in addressing NP-hard problems outweighs their inability to guarantee to find the global optimum. For this research, the MOGA solver in MATLAB is deployed to solve problems in this section.

An example workspace with 12 scattered obstacles, 3 *Start* nodes, 4 *Goal* nodes, and 2 breakouts, the locations of which are to be found, is shown in Figure 3.23. In this figure, S_i is the i^{th} *Start* node and G_j is the j^{th} *Goal* node. Also shown in this figure is the convex hull of the nodes and intersecting objects in blue.

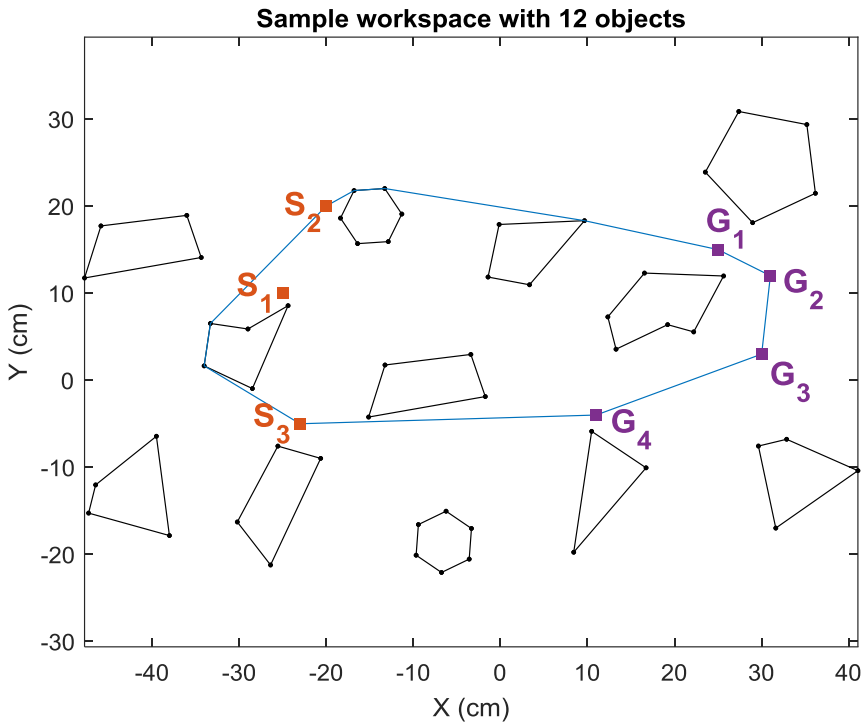


Figure 3.23 Sample workspace with start and goal nodes

The problem is solved using the explained setup and MATLAB's MOGA solver with 100 generations and a population size of 50. The final set of non-dominated solutions can be seen in Figure 3.24. It should be reminded that due to the utilization of a heuristic solver, at each execution of the GA a new set of non-dominated solutions is generated and the non-dominated solutions at the last generation cannot be guaranteed to match the true Pareto set.

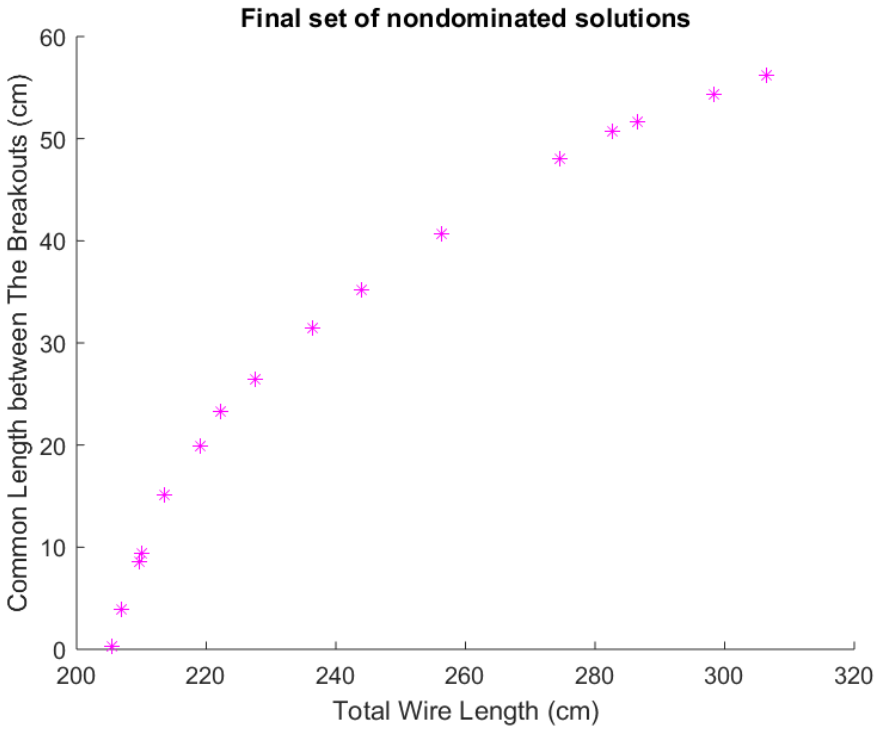
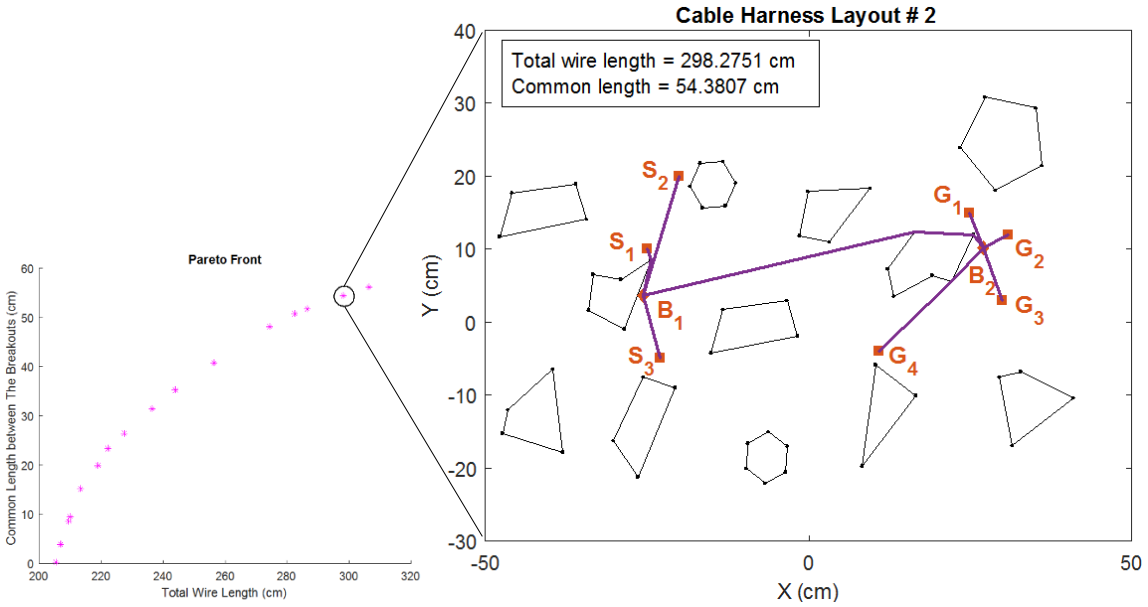
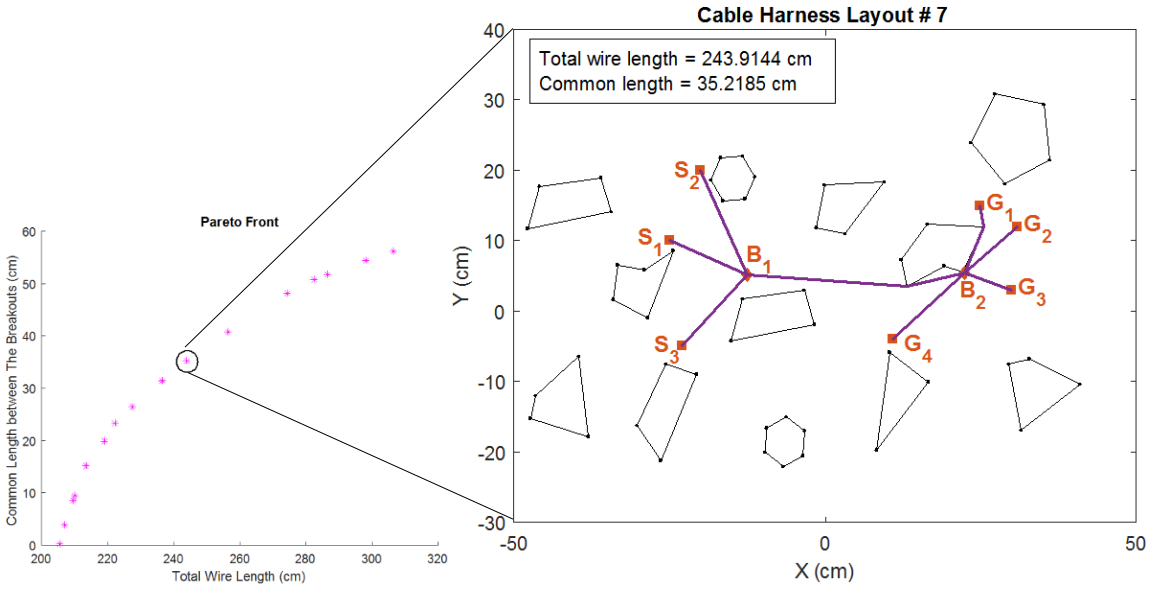


Figure 3.24 Non-dominated set of solutions for Figure 3.23

For every point in the non-dominated or eventual Pareto set, there is an associated optimal layout for the cable harness found by locating the breakouts. Four sample layouts are depicted in the following figures.


Cable Harness Layout # 2
Total wire length = 298.2751 cm
Common length = 54.3807 cm
Pareto Front
Common Length between The Breakouts (cm)
Total Wire Length (cm)
Y (cm)
X (cm)
S1
S2
S3
B1
G1
G2
G3
G4
B2


Cable Harness Layout # 7
Total wire length = 243.9144 cm
Common length = 35.2185 cm
Pareto Front
Common Length between The Breakouts (cm)
Total Wire Length (cm)
Y (cm)
X (cm)
S1
S2
S3
B1
G1
G2
G3
G4
B2

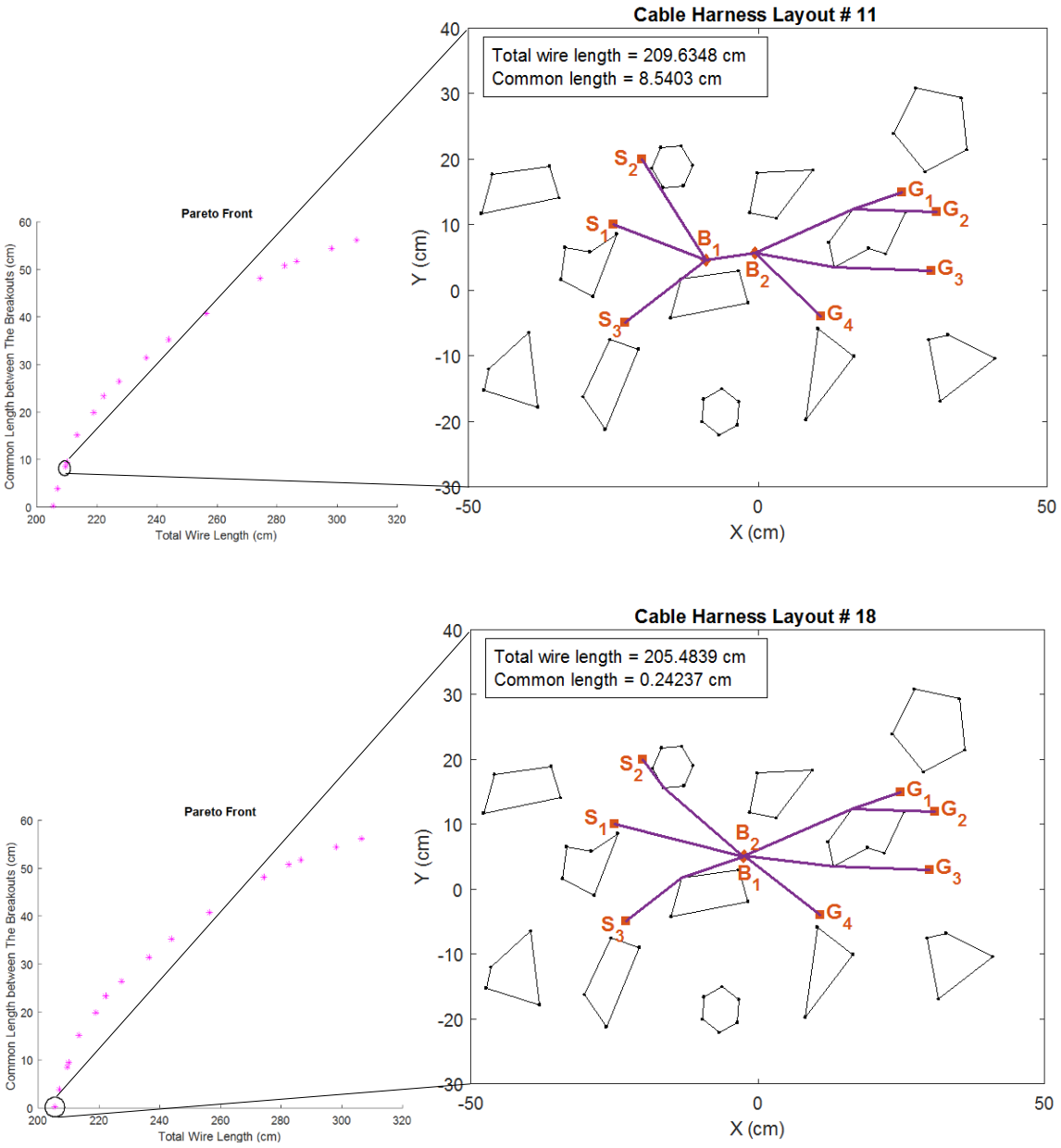


Figure 3.25 Sample optimal layouts for Figure 3.23 example

In examples of Figure 3.25, the layouts are selected from the set of non-dominated solutions (local Pareto optimal solutions) and drawn in a separate figure (right). It can be seen that changing the locations of the breakouts could change a layout significantly. It is evident that maximizing the common length of wires between the two breakouts will result

in an overall longer wire harness. An interesting case is layout 18 where the two breakouts coincide at the same location, zeroing the total common length. While this layout may not provide any commonality for bundles of wires, it can still bring insight to the designer when deciding about the final layout. It would, therefore, be worthwhile to compare this solution with the case where no breakout is used and the goal is to only minimize the total lengths of wires. The case of separate paths without any breakouts is created for the example in Figure 3.23 and the final layout is shown in Figure 3.26.

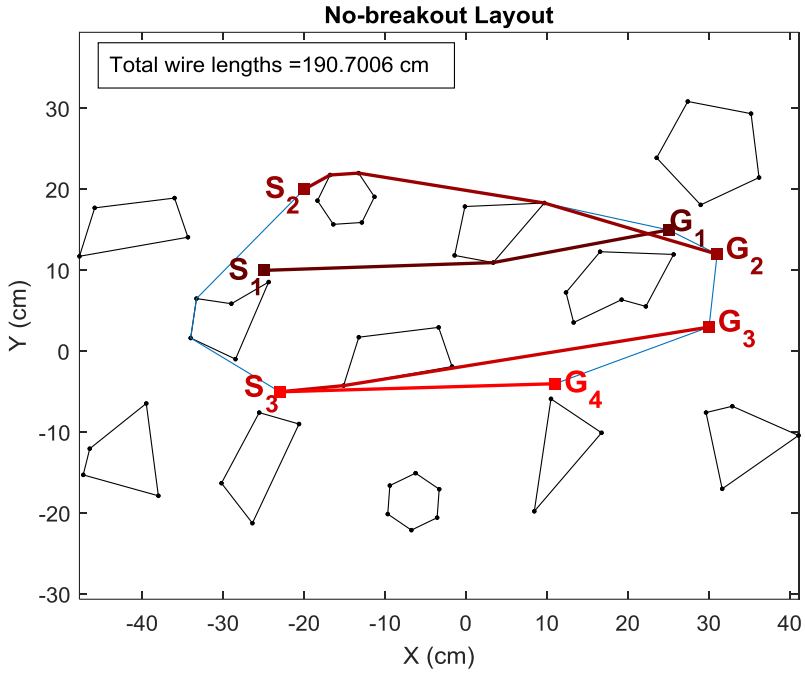


Figure 3.26 No-breakout layout example for Figure 3.23

In the next section, the effects of changing the number of *Start* or *Goal* nodes and the density of the workspace, measured by the ratio of the occupied regions inside the convex hull over the total area of the convex hull, on the optimal layout are further investigated.

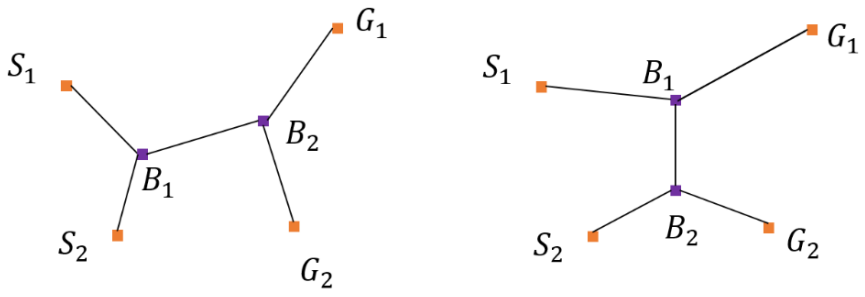
3.4.3 Results and discussion

This section evaluates the effects of the geometric structure of the workspace on the optimal solution to the cable harness layout problem. Since the optimal solution is not unique, to make the comparison of different layouts more meaningful, three solutions are selected from the Pareto set: the solution with the maximum distance between the two breakouts, the solution with the minimum total lengths of wires, and finally the solution with no breakouts.

3.4.3.1 Effects of the number of nodes and the number of breakouts

While having more components to connect evidently requires more wires and therefore increases the total length of wire harness, other factors such as the locations of the nodes (components) also affect the total and common lengths. Hence, it is inconclusive as to how increasing the number of nodes in the workspace alone could affect the optimal layout of the harness without considering where the new nodes are located.

Further, analyzing the effects changing the number of breakouts has on the optimal solution requires the knowledge of the topology of the harness. The topology of the harness shows which nodes are connected to each breakout and how the breakouts are connected. For example, Figure 3.27 shows two different topologies for the case with 4 total nodes and 2 breakouts.



Topology (a) Topology (b)

Figure 3.27 Two different topologies for 4 nodes and 2 breakouts

Note that these are to show different topologies created with the same number of nodes and breakouts and they may not necessarily satisfy the physical requirements of a cable harness.

3.4.3.2 Effects of the workspace density

One of the challenges the designer of a cable harness layout faces is the limited feasible space remained to route all the wires and locate the breakouts in the detail design stage. Adding more objects to the same workspace results in a more densely populated environment. Therefore, the designer must know the effects of the density of the environment on the optimal layout of a cable harness assembly. The density of the workspace, in this research, is defined as the ratio of the area occupied by the obstacles inside Klamroth's convex hull over the area of the convex hull:

$$density(\%) = \frac{area(obstacles)}{area(Conv\ hull)} \times 100 \qquad (3.1)$$

Since the iterative convex hull is used to further bound and downsize the feasible domain for faster computation of the optimal solutions, it is reasonable to only consider the objects inside this convex hull as the obstacles to be avoided by the wire connectors.

MATLAB's `convhull` function, which is used for the calculation of the 2D convex hull in this research, also outputs the convex hull area. For the calculation of the area of each of the obstacles inside the convex hull, MATLAB's `polyarea` function is used that is capable of finding the area of any polygonal region (convex as well as nonconvex) as long as the vertices of the polygon are in clockwise or counterclockwise order. The VRML format used to store and represent the obstacles' geometry does not necessarily come with ordered vertices. Thus, an algorithm is developed that sorts the vertices of the obstacles in clockwise (or counterclockwise) order.

To evaluate the effects of density on the optimal solutions, 11 different test cases are generated by varying the density from 14.25% to 52.36% in the feasible region of the workspace. Since the density of the workspace cannot be controlled, in this research, the density is increased by adding objects inside the convex hull until the computation time increased beyond one hour (for the density of 54.5%, which did not yield a solution within one-hour runtime of the algorithm). To make the comparison of the test cases possible, two *Start* and two *Goal* nodes are used with fixed locations across all the tests. The locations are $S_1 = (-25,10)$, $S_2 = (-20,20)$, and $G_1 = (25,15)$, $G_2 = (11,-4)$.

Additionally, the number of required breakouts is kept at 2. The workspaces of these test cases are shown in Appendix A. The data of maximum common length, minimum total

wire lengths with and without the breakouts, and the total computation time for each test case is compiled and recorded in Table 3.7.

Table 3.7 Results for testing the effects of density on optimal layout

Test ID	Workspace density (%)	Max common length (cm)	Min total length with breakout (cm)	Min total length without breakout (cm)	Total computation time (sec)
1	14.25	39.1647	89.934	89.454	20.4921
2	16.80	44.3307	90.942	89.6053	50.4038
3	21.88	32.6631	89.9212	89.8316	73.0086
4	28.65	44.0532	91.5033	90.2484	154.9674
5	31.09	41.6997	91.2973	90.2484	174.2097
6	34.64	47.7741	92.256	90.6393	262.6323
7	37.75	49.4265	94.4517	91.8152	352.5452
8	42.06	48.933	94.5502	91.8152	544.1885
9	45.36	31.8739	95.3751	92.0603	595.7788
10	49.12	36.1219	94.2169	92.3305	800.6291
11	52.36	33.2051	97.5491	93.0223	1219.9532

It can be seen in Table 3.7 that increasing the density increases the minimum total lengths of wires as well as the computation time (see also Figure 3.28, Figure 3.30, and Figure 3.31). The computation time seemingly increases exponentially with the increase in the density. Unlike the minimum total length, a trend is not observable in the changes to the maximum common length as density increases (see Figure 3.29). Since increasing the density beyond 52.36% in the same workspace results in the exponential growth of the computation time, cases with densities greater than 52.36% are not further explored.

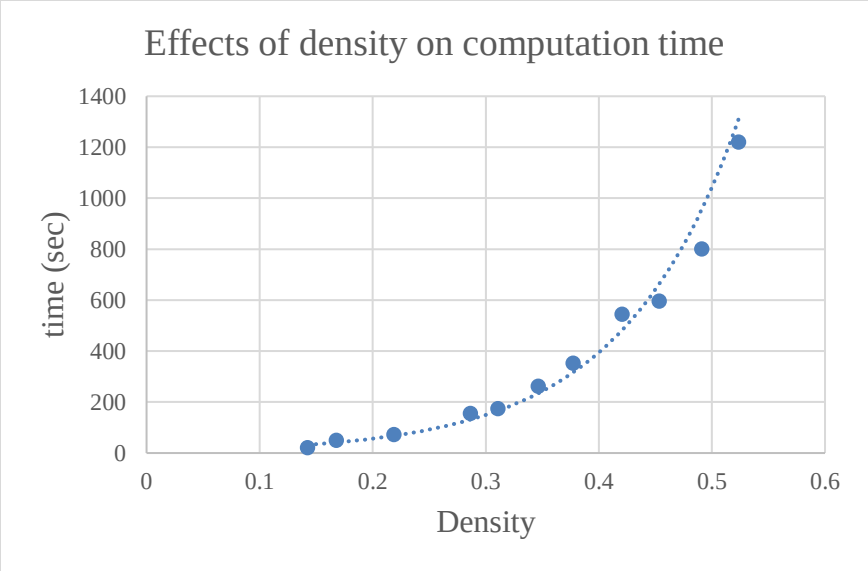


Figure 3.28 Effects of the density of the workspace on the computation time

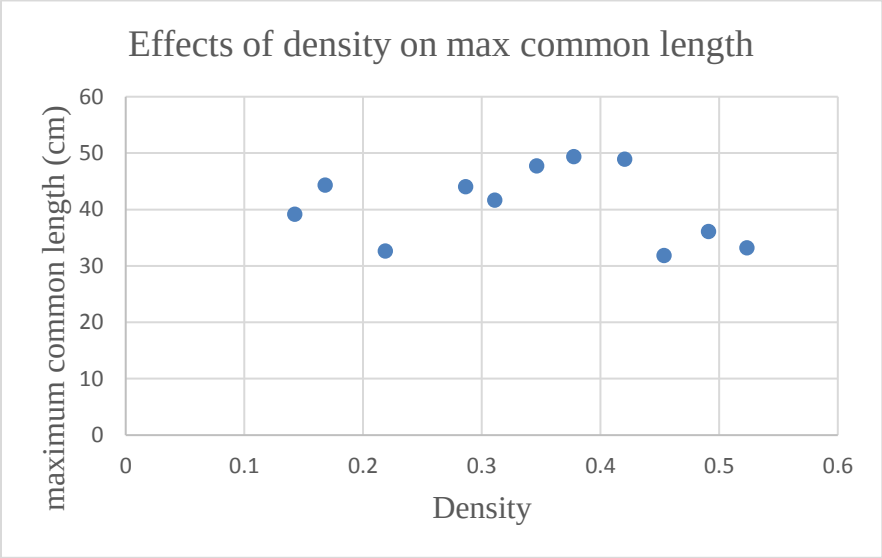


Figure 3.29 Effects of the density of the workspace on the maximum common length

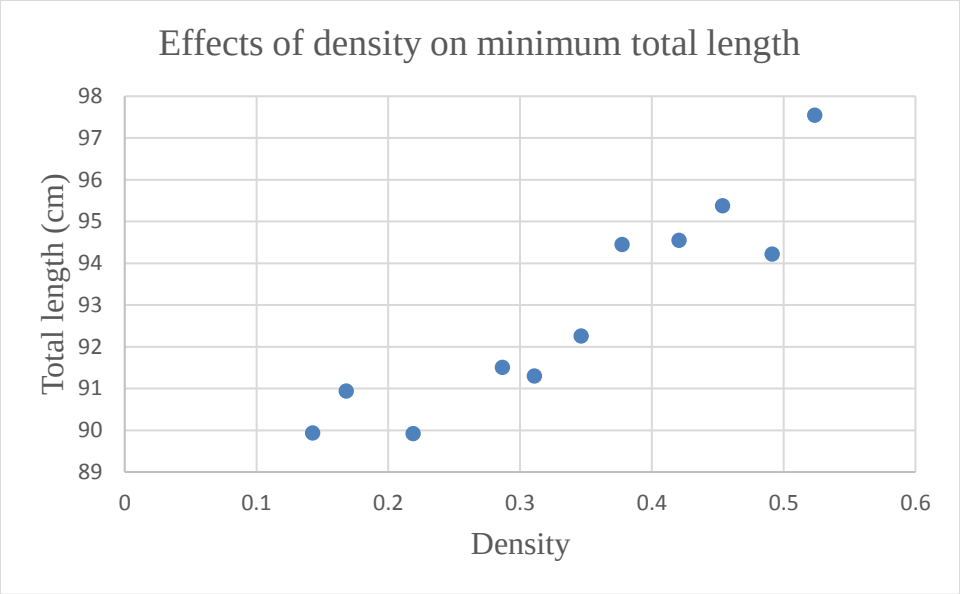


Figure 3.30 Effects of the density of the workspace on the minimum total length

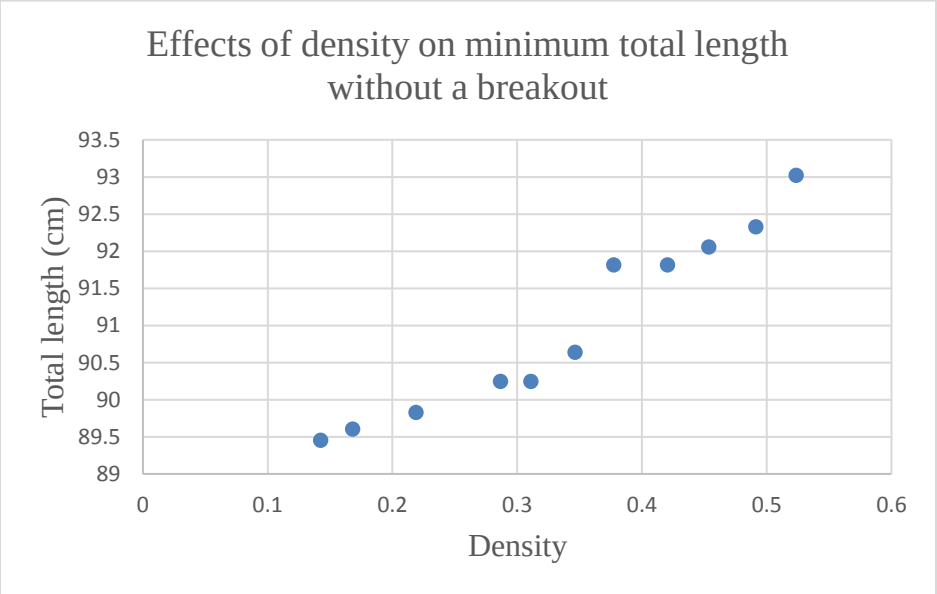


Figure 3.31 Effects of the density of the workspace on the minimum total length without a breakout

While relative conclusions can be drawn, it should not be overlooked that the solver used for this optimization problem is heuristic-based. Therefore, the found solutions are

locally optimal and it cannot be expected to achieve the same results by solving the problem repeatedly. Thus, the values entered in Table 3.7 are subject to change by future executions of the algorithm.

There might also be cases of densely populated environments where the convex hull of a nonconvex obstacle encompasses a part of another obstacle. An example of such a case is depicted in Figure 3.32 where a part of the second obstacle lies inside the convex hull of the first obstacle (dashed green lines).

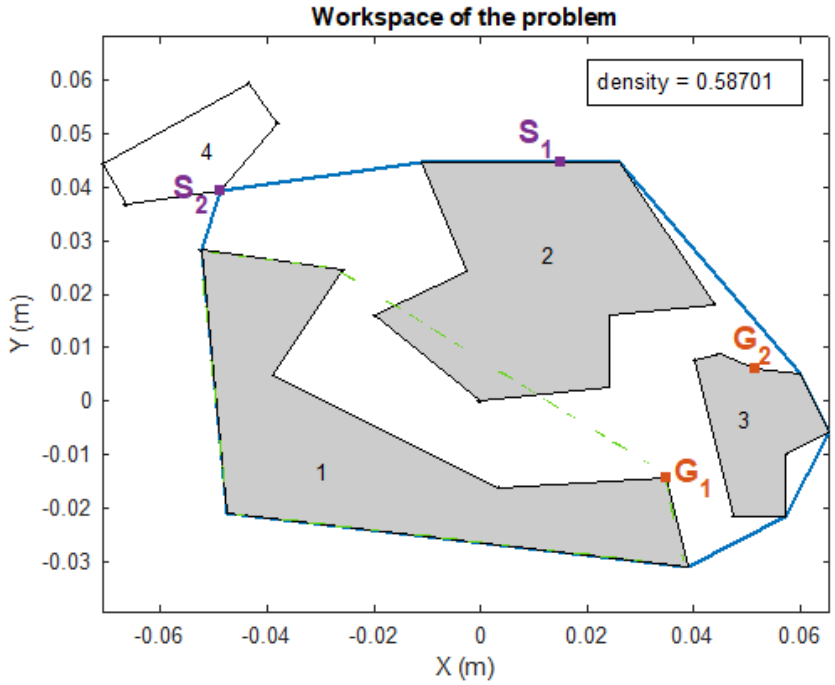


Figure 3.32 Example of interlocking obstacles in a dense environment

In such a case, the convexification of the obstacles that is used as a step in the optimization process would result in entirely blocking the passage between the two obstacles. This blockage might further lead to the omission of some of the optimal solutions from the Pareto set. Therefore, it is suggested in this research to use the actual obstacles' edges and vertices and avoiding convexification of the obstacles inside Klamroth's convex

hull when the `InPolygon` function is called in such environments. Note that to be able to use the actual obstacle's vertices in `InPolygon` function, the vertices must be in either clockwise or counterclockwise solution, as explained before.

Using the actual vertices allows GA to include in the population the locations that are inside the convex hull but outside the actual boundary of a non-convex obstacle which was considered infeasible when the convex hull of the object was used instead. For example, a solution to the workspace of Figure 3.32 is outlined in Figure 3.33. The layout shown in this figure has two breakouts, B_1 and B_2, both of which are located inside the convex hull of the second object. Had the passage between the two interlocking objects in Figure 3.32 been blocked, the wires would have had to go around these objects to reach G_1 and G_2, which would have lengthened their route. Also, the layout with no breakouts is shown in Figure 3.34 for comparison.

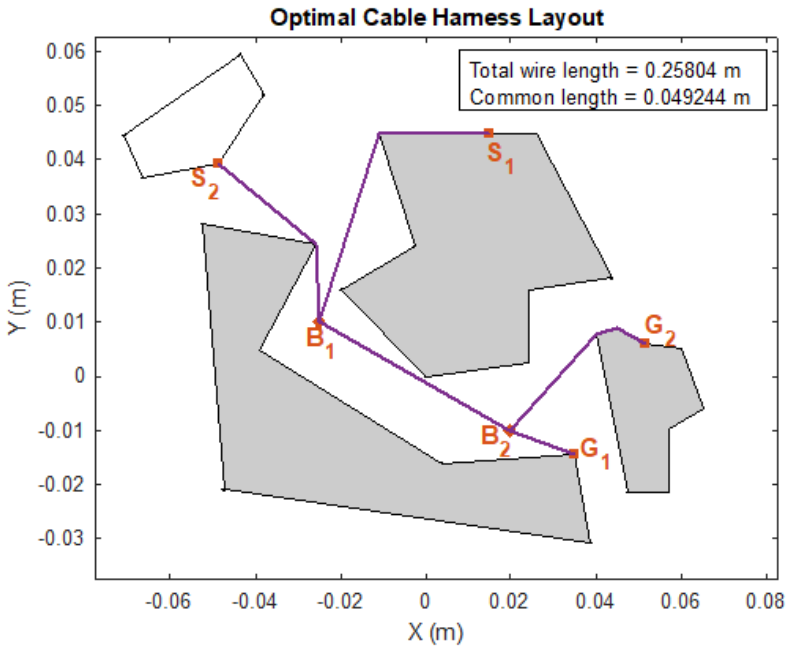


Figure 3.33 Sample optimal layout for the workspace of Figure 3.32

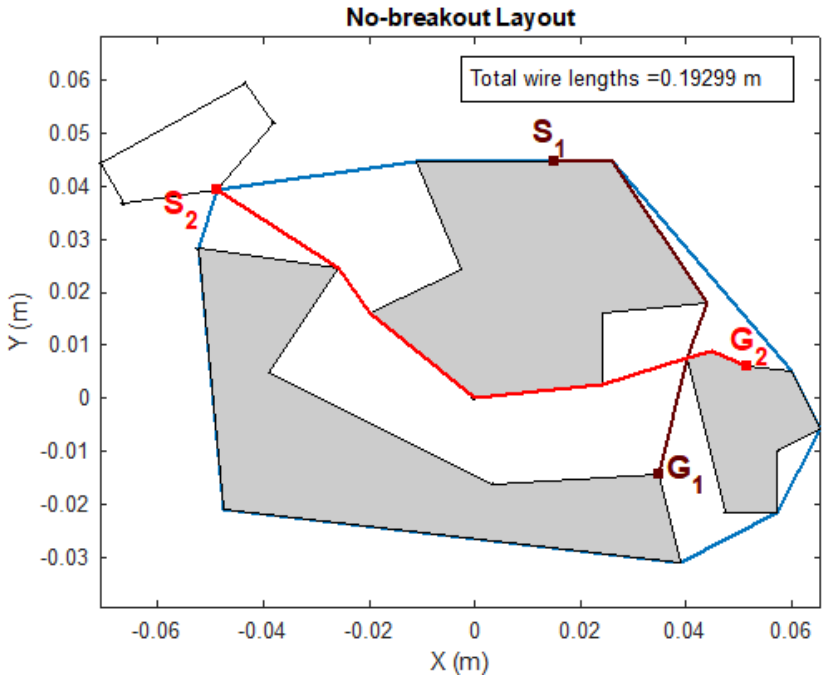


Figure 3.34 Wire layout without breakouts for the workspace of Figure 3.32

The convex hull based multi-path planning is also compared with the mixed-binary optimization using **Problem 3-2** as the test case. The results are shown in Figure 3.35 and Figure 3.36. Even though the locations of the breakout in Figure 3.35 found by the convex hull based routing are quite different from Figure 3.19 generated by solving the mixed-binary optimization problem with the ε-constraint method, the local Pareto fronts look quite similar.

It should, herein, be reminded that despite the strength of the mixed-binary formulation of the problem using exact Euclidean distances, the approach is limited by the increase in the complexity of the workspace such as the number of obstacles, the shape of the obstacles, the number and locations of the existing nodes, and the number of breakouts.

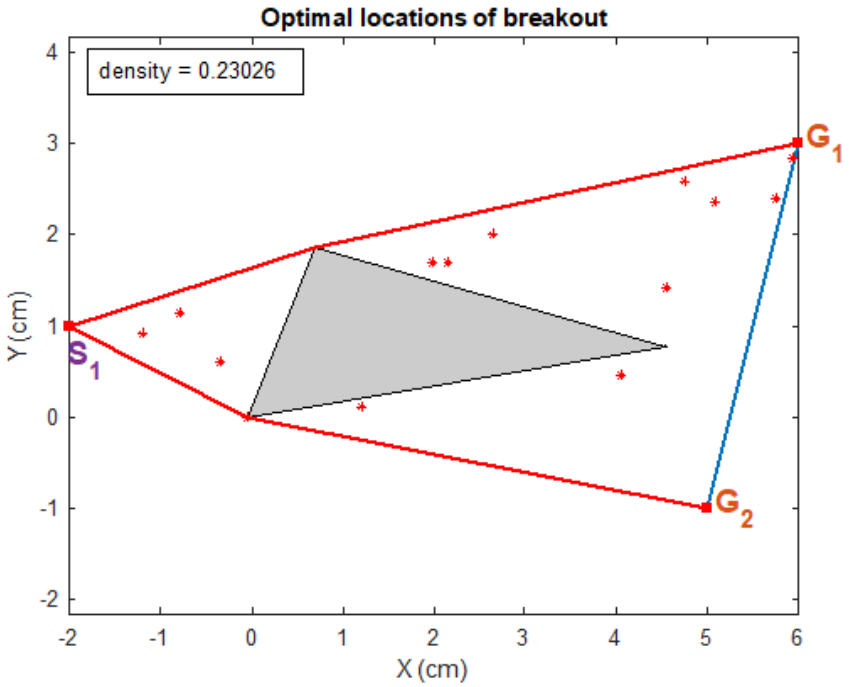


Figure 3.35 Optimal locations of the breakout for Problem 3-2 found using convex-hull based routing

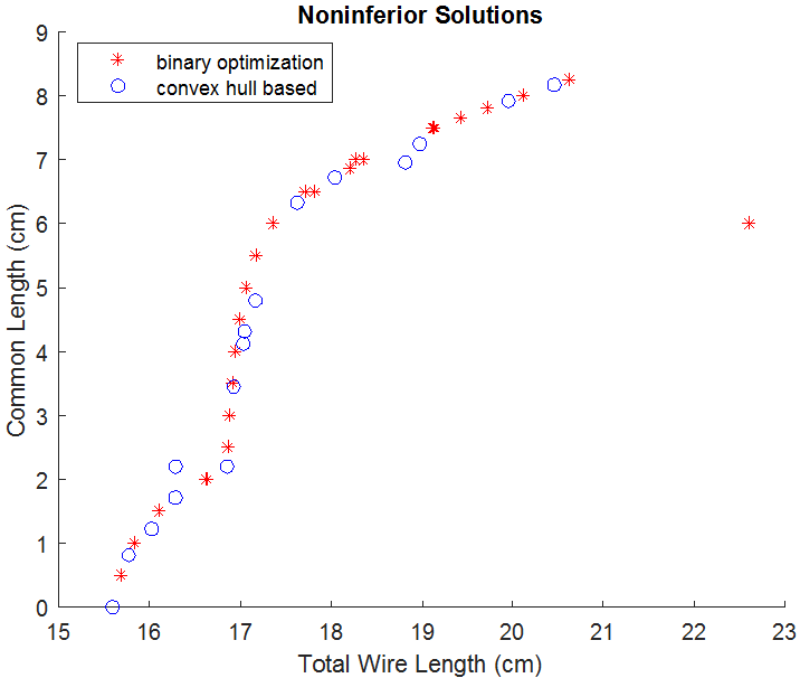


Figure 3.36 Non-dominated sets for Problem 3-2 found using convex-hull based routing vs. mixed-binary optimization

Chapter Four
OVERVIEW OF THE RELATED WORK ON 3D PATH PLANNING METHODS

Although plenty of methods have been introduced and implemented to address path planning problems in 2D environments, as reviewed in Chapter 2, due to the inherent challenges of 3D path planning, most of those methods are found inefficient in dealing with 3D problems (e.g. the classical visibility graph) and some may not even apply (e.g. Voronoi diagrams). In fact, according to Canny and Reif [7], the path planning problem in its general form in 3D environments is an NP-hard problem, i.e. it cannot be solved in polynomial time. Therefore, to find a solution to 3D shortest path problems within a reasonable computation time, researchers mainly resort to stochastic and heuristic techniques for which there is no guarantee to find a globally optimal solution. Some, on the other hand, value the optimality of the solution higher than the computation time and attempt to adopt deterministic methods to solve 3D problems. In an effort to reach polynomial-time complexities, algorithms have been developed that generate approximate shortest paths [176]. These efforts have resulted in methods that either make simplifying assumptions and generate an approximate shortest path or address the special cases of the general 3D problem.

This chapter highlights the two popular classes of methods for 3D path planning problems: variants of visibility graph, based on a deterministic method covered in Chapter 2, and non-deterministic (including heuristics and stochastics) approaches.

4.1 3D Visibility graph

While the discussed methods of constructing visibility graphs apply to 2D environments, the construction of a 3D visibility graph may not be as straightforward. The notion of visibility graph in 3D spaces may differ from its 2D counterpart. O'Rourke [10] provided different definitions for visibility graphs. According to him, a 3D visibility graph may be created between two objects instead of a pair of nodes, which can greatly simplify the graph construction. In that case, each object serves as a node. Using this definition of visibility may, however, result in non-optimal solutions to shortest path problems. Nonetheless, using the classical definition of a visibility graph, the construction of that graph requires the determination of all visible points of an object from a given point in the space. By the classical definition, in 2D planning, the graph nodes are the vertices of the intersecting polygonal obstacles as shown in [83]. In 3D, on the other hand, the visible points used in the shortest path lie anywhere on the edges of the polyhedral obstacles [24]. Therefore, the construction of visibility graphs in 3D becomes an NP-hard problem [10].

In addition, moving from 2D to 3D, the definition of intersection also changes. In 3D environments, the intersections occur between a line segment and an interior of a polyhedron, not a polygon. Therefore, intersections need to be checked between a line segment and an object's edges as well as its surfaces. If tessellated models of objects are used, it suffices to only check intersections between the line segment and the triangles that compose the surface of the solid model. This fact alone can increase the time complexity of any graph construction algorithm drastically. Further explanation of the intersection detection algorithms used in this study is provided in Chapter 6.

Despite the discussed challenges, work has been done on developing construction algorithms for 3D visibility graphs with simplifying assumptions (e.g. the approximate shortest path) or for special cases. For example, Lozano-Pérez and Wesley [13] extended their approach to 3D path planning based on visibility graphs by introducing new vertices along the edges of polyhedral obstacles, further subdividing an edge. According to them, a 3D visibility graph whose nodes consist only of the obstacles' vertices is not guaranteed to contain the shortest collision-free path. The new vertices they introduce can lie anywhere on an edge of an obstacle (or its dilation in the configuration space) such that the length of each subdivision does not exceed a pre-specified value. The addition of the new vertices can lead to a reasonable approximation of the shortest path on the visibility graph though the computation time may be significant depending on the size of the graph.

Sharir and Schorr [24] presented a doubly exponential (has the form of a^{b^x}, where a and b are constants) algorithm in terms of the number of wall edges to find a sequence of edges of obstacles through which the shortest path passes. They identify the contact points on the edges of the obstacles by solving a system of m equations (for m segments of the shortest path) which sets the arriving and leaving angles of the path segments at each edge equal. They prove that the shortest path on a sequence of edges is unique and the i^{th} turning point on an edge is such that the angle created between i^{th} segment of the path and the edge is equal to the angle between $(i+1)^{th}$ segment and the same edge. Even though the analytical solution to this problem is computationally expensive (even numerical methods take $O(m^m)$ to solve a system of m equations), they considered a special case of finding the shortest 3D path along the surface of a convex polyhedral object which is solvable in

$O(n^3 log\ n)$ for n vertices of the polyhedron. This path is also known as a *geodesic* path [24].

To further improve the computational time of the 3D shortest path algorithm suggested by Sharir and Schorr, Papadimitriou [15] proposed an algorithm capable of finding approximate 3D shortest paths that are at most (1+ε) times longer than the globally shortest path. His algorithm can be run in the polynomial-time of $O(n^4(L + \log(n/\varepsilon))^2/\varepsilon^2)$, where n is the number of vertices and L is the precision of the integers used (for example for the coordinates of the vertices, L is the base 2 logarithm of the largest integer used in a coordinate). The proposed algorithm subdivides each edge into at most $N = O(n(L + \log(1/\varepsilon))/\varepsilon)$ segments. He defines visible edges as a pair of edges with two points, one on each edge, visible to each other. If such points exist, the segments are visible. Next, he calculates the distances between visible edges as the distance between their midpoints. Finally, applying Dijkstra's algorithm to the visibility graph, the shortest path can be found.

Clarkson's method [30] discussed in Chapter 2 is also applicable to 3D path planning problems. In fact, he provided improvements to the time complexity of Papadimitriou's algorithm for the 3D visibility graph. The idea here is analogous to the 2D problem. the conical regions are created for the nodes of the graph. This time, however, the nodes are not necessarily the vertices of the obstacles. Instead, the apex of the cones (or nodes of the reduced graph) lies on the edges of the obstacles which implies infinitely many vertices. To avoid this burdensome computation, Clarkson subdivides the edges to a finite number of segments. After the graph is constructed, the search algorithm presented

in [31] is applied to obtain the shortest path. In addition to Clarkson, Choi et al. [177] also revisited Papadimitriou's algorithm and defined a new subdivision scheme to further lower its running time.

An $O(n^{6k-1})$ time algorithm is developed to find the 3D shortest collision-free path amidst vertical obstacles, resembling buildings in an urban setting, with a total of *n* vertices and *k* distinct heights [178]. Vertical obstacles are such that each of their faces is either parallel or perpendicular to the *xy* plane. The authors also proposed speedup techniques that improve the time complexity up to $O(n^2)$ though the resulting paths are longer by a maximum of 8% due to the deployed approximations and simplifying assumptions.

In this algorithm, a visibility graph is constructed per level. After all the graphs are found, they are connected to form a 3D graph which is searched for the shortest path. The algorithm benefits from the orthogonality of the objects when using their projections to construct visibility graphs and find the waypoints.

A 3D Reduced Visibility Graph (3DRVG) is introduced in [179]. The proposed construction algorithm has polynomial computational time in terms of the number of vertices and exponential time in terms of the number of obstacles, $O(n^3 v^f)$, where *n* is the number of vertices, *f* is the number of obstacles, and *v* is the maximum number of vertices in any one obstacle.

To construct the 3DRVG, the authors explained a perspective projection referred to as collineation that projects the obstacles on a plane perpendicular to the line connecting the start and goal. The projection viewpoint is the start point. An example of the defined collineation is shown in Figure 4.1.

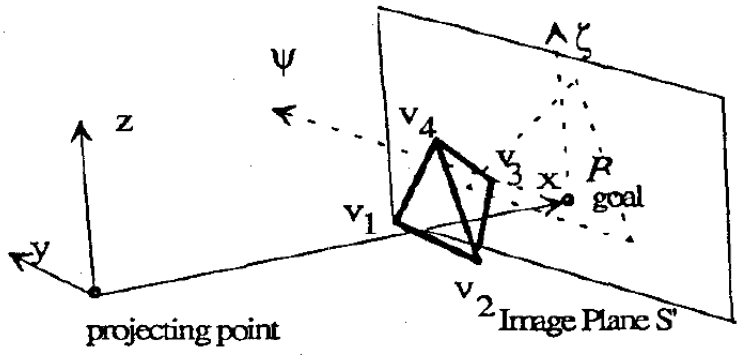


Figure 4.1: Illustration of collineation concept [179]

Using the projected image of the obstacles, non-overlapping edges are identified by removing the hidden lines in the projected image. Using the information of visible edges in the 2D projection plane, the corresponding edges on the 3D obstacles are identified. After the visible edges are found, they are connected in a sequence starting from the start point, passing through the midpoint of each edge and ending at the goal point (see Figure 4.2). However, since the midpoints of the edges may not yield the shortest path, an elastic string analogy is used to resemble each path to minimize the total potential energy of the elastic strings. This optimization leads to minimizing the length of the path by moving the path turning points on the visible edges of the obstacles.

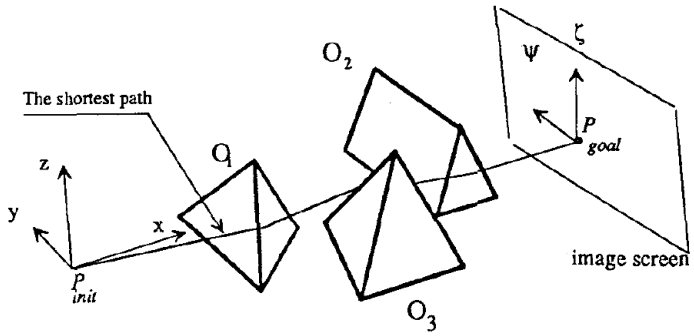


Figure 4.2: The shortest path through the midpoints of the found edge sequence [179]

A modification to Gewalli's algorithm is provided in a more recent study by Frontera et al. [180]. Their modified algorithm, known as ApVL, reduces the number of vertices used to find the shortest path and as a result, improves the computation time to $O(n^3)$. Similar to [178], they construct 2D visibility graphs at different levels *i*, $1 \leq i \leq k$, for *k* distinct levels of obstacles. However, their classification of levels differs from that of [178] in that they have evenly spaced levels and no matter how many distinct heights the obstacles have, they keep *k* constant. They then use projections of the visibility nodes at different levels to create connecting edges between visibility graphs at different levels and find the shortest path in $O(k^2n^3)$ which, assuming a constant value for *k*, is reduced to $O(n^3)$. A sample three-level visibility graph is shown in Figure 4.3. The first step in their algorithm is the determination of the intersecting obstacles. These obstacles are then passed to the approximate graph generation to build the visibility graph which is later searched using A*.

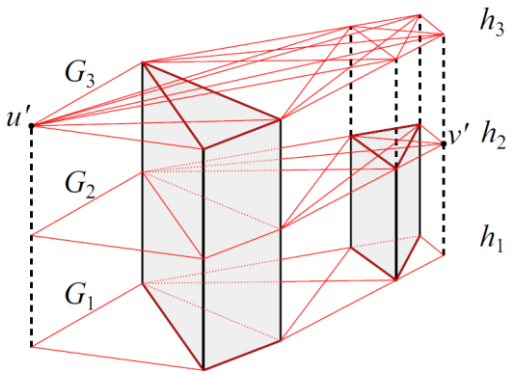


Figure 4.3: Three-level visibility graph [180]

A downside to their approach is the discarding of non-intersecting obstacles. This contributes to the intermittent collision between the final shortest path and the discarded

obstacles. To avoid this, they check for collisions once again after the shortest path is created to ensure the path is collision-free. If collisions are found, the process is performed iteratively to converge to a collision-free path between the two given points. This algorithm is also compared with the sub-sampling algorithm [13], approximations by Gewalli et al. [178], visibility line-based [181], and stochastic methods of PRM and RRT. The results show that it outperforms the rivals in finding the 3D shortest path in an urban environment with regular obstacles both in computation time and the path length.

Looking at the same problem but adding extra constraints and criteria (such as the number of links and the maximum height of the obstacles), Tran et al. [9], developed an algorithm that generates even shorter paths than ApVL's output, among convex vertical polyhedra. Their algorithm has two main steps: (1) the construction of a visibility graph based on the obstacle segments that are fully or partially visible to each other and (2) solving a Mixed Integer Linear Program (MILP) that attempts to find the location of waypoints on the nodes of the visibility graph. An example of two partially visible segments identified based on Tran's algorithm is shown in Figure 4.4. It should be noted that the nodes of the visibility graph are in fact edges of the obstacles. Therefore, instead of constructing a visibility graph based on visible nodes, the authors construct the graph for visible edges. They also deploy a linear approximation of the Euclidean distance metric to be able to model the problem and solve it as a MILP.

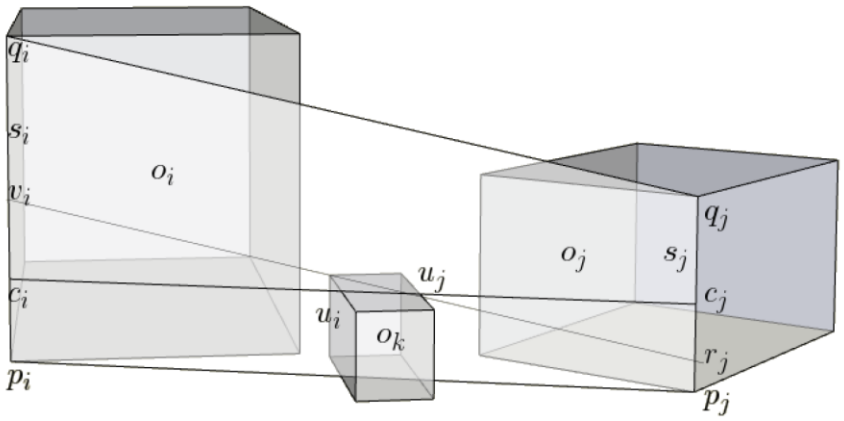


Figure 4.4: The notion of partial visibility for segments [9]

Although the method is competitive in finding reasonable approximate solutions to the shortest path problem using an exact geometric-based approach, hardly could it be applied to solve 3D path planning problems with non-vertical obstacles. The method of determining the visibility of obstacle edges is significantly simplified by using vertical obstacles that have either parallel or perpendicular faces to the *xy* plane. In addition, the distance metric used approximates the Euclidean metric, which requires post-processing steps to further refine the path for a shorter one.

The methods developed in [178], [180], and [9] all make an assumption that traveling below the base surface of any obstacle is not permitted; hence mimicking an urban environment for finding the path. Although this assumption is valid for the case of UAV path planning in an urban setting, for the problem of wire routing it may not be realistic to assume wires can only extend over the top (or sides) of the obstacles. Therefore, the proposed approach may not apply to the general problem of 3D path planning.

Liang et al. [182] developed a geometric-based path planning approach in two and three-dimensional workspaces scattered with regular objects including rectangles and

ellipses in 2D and cubes and cones in 3D. The algorithm starts with identifying the intersecting obstacles and ordering them based on their distance from the start point of the path. Then, the points of intersections with edges (2D) or faces (3D) are determined and the closest point to the path start point is selected. The distances from the selected point to the vertices (2D) or edges (3D) of the respective edge (2D) or face (3D), to which the point belongs, are calculated and the vertex or edge with the minimum distance is chosen as the sub-goal of the path. An example of a sub-goal is illustrated in Figure 4.5 (point A).

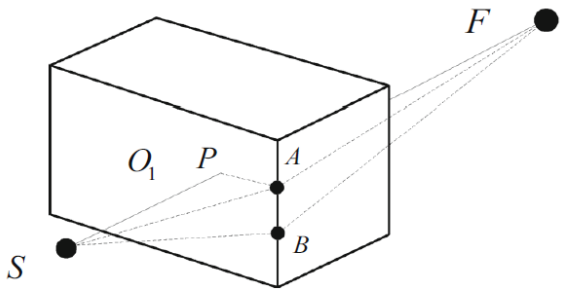


Figure 4.5: The sub-goal for a cuboid obstacle[182]

They showed that for any arbitrary point, B on the edge with minimum distance from P, the inequality of $\|SB\| + \|BF\| \geq \|SA\| + \|AF\|$ holds; therefore, A is on the shortest path from S to F. In the next iteration, the sub-goal is set as the new start point and the process is iteratively performed. These sub-goals act as the waypoints and if they form non-intersecting segments, they will be appended together to create a collision-free path. Figure 4.6 shows an example of the final shortest path in a 2D environment using this method.

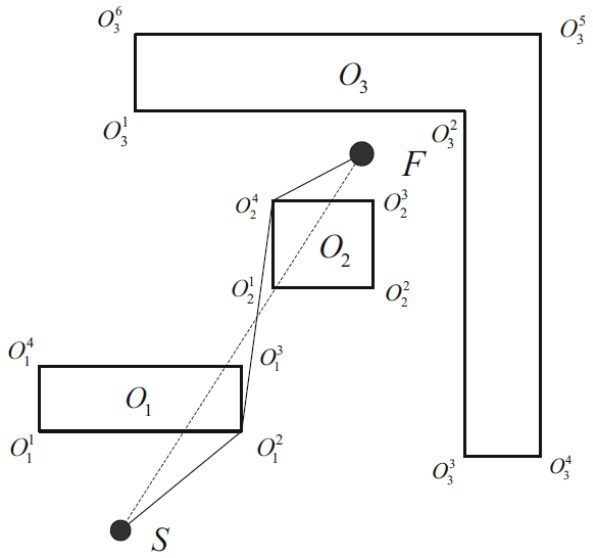


Figure 4.6: The shortest path for a 2D workspace [182]

Though claimed by the authors that the path found by this algorithm is a good approximation of the shortest path, no proof is provided that such paths are near-optimal. Comparisons are provided with 2D and 3D heuristic methods that show the superiority of the method in the optimality of the solution. However, the heuristics may not be a valid reference as they cannot guarantee to find a globally optimal solution. Further, the proposed algorithm can only handle regularly shaped obstacles and the process of determining the sub-goals highly depends on the obstacles' shapes; therefore, it may not apply to a more general case of 2D problems or any 3D planning problems.

4.2 Non-deterministic methods

Heuristic and stochastic methods are popular in addressing 3D path planning problems as they can often generate an adequate solution in a reasonable time. If a solution exists, they generally find it, however, it could take drastically long for these methods to converge to the exact solution depending on the scale of the problem; thus, often there is

no guarantee the found solution is the global optimum. Stochastic methods (e.g. PRM and RRT) are useful for environments that contain levels of uncertainty where also real-time routing is required. Here we refer to a few examples of heuristic and stochastic planning approaches for 3D problems.

One of the early efforts to use evolutionary algorithms for path planning problems was made by Szykman and Cagan [183]. They proposed an approach based on Simulated Annealing (SA) to produce non-orthogonal routes for pipes in a 3D environment. Given the locations for a pair of terminals, an initial route, which is the straight line between the two terminals, is chosen. Then, the optimizer based on SA moves the locations of bend points, which are design variables, to minimize an objective that consists of the sum of three components: the total length of the route, the number of bends, and the degree of penetration inside obstacles. Weights are used to distribute the importance of the three objectives, and the aim is to drive the third one (obstacles interference) to 0. Figure 4.7 shows an example of an optimal layout for a four-story chemical plant using the approach introduced in [183].

Later, Sandurkar and Chen [76] addressed the pipe routing problem in 3D space using the tessellated format (triangulated meshes for the surface approximation of solid models) to represent components in the workspace and implemented a Genetic Algorithm (GA) that determines angles and lengths of each segment of a single pipe.

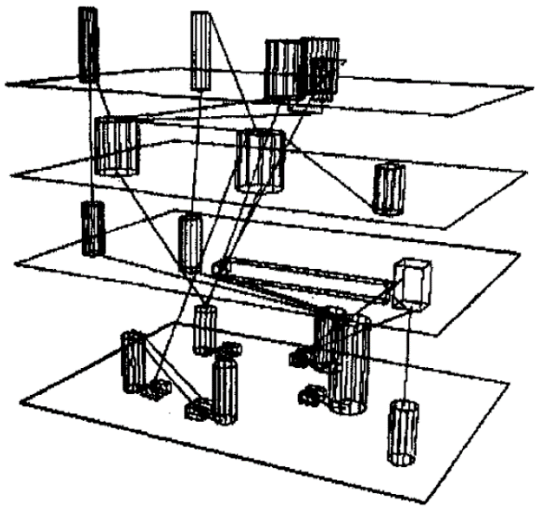


Figure 4.7: Optimal layout for a chemical plant using SA[183]

While GA and SA are among the most popular heuristic techniques for 3D routing, researchers have also applied Ant Colony (e.g. for pipe routing in ships [184] and 3D hose routing[185,186]), Particle Swarm (e.g. for pipe-assembly in aero-engine [187], pipe routing [188,189], and robot path planning [190]), and Tabu Search (e.g. for vehicle routing [78]).

In addition to heuristics, other non-deterministic methods such as sampling-based methods of PRM (e.g. see [69,191,192]) and RRT (e.g. see [193–195]) have drawn a lot of attention by 3D path planning researchers and many have applied them to solve 3D planning problems in complex environments.

Heuristics and sampling-based methods are popular due to their simple implementation and computational efficiency and therefore there is a multitude of studies on these routing methods for different applications. However, since the focus of the present work is on deterministic methods, we skip the further discussion of the related work based on non-deterministic methods.

4.3 Other methods for 3D path planning

Although visibility based, heuristics, and sampling-based methods are popular among scholars for solving 3D routing problems, other methods discussed in Chapter 2 have also been applied to 3D problems and their performance has been evaluated. In this section, a brief explanation of the most common of these methods is provided.

Potential fields- As explained in Chapter 2, PF is a method that benefits from defining potential functions for various points in the workspace. The potential associated with the goal point is zero and the objective is to minimize the total potential function as the router moves from the start point toward the goal. By this definition, the PF method can be similarly applied to 3D path planning problems (for example see references [196–198]). Despite the strength of the method in addressing dynamic environments, similar to 2D, the PF method has a downside of trapping at local minima in 3D environments as well, especially in environments with closely spaced obstacles. Different solutions are proposed to overcome this problem by defining new potential functions [198] or placing an imaginary goal point near the trap for the router to escape the local minimum [65]. These potential functions may, however, approximate the shortest path, therefore, could result in sub-optimal paths.

Voronoi- Retraction methods using the Voronoi diagram are the best candidates for planning the safest path among scattered objects. Although the method is more popular for 2D safe routing problems, some researchers have benefitted from generating Voronoi diagrams for 2.5D environments or 2D projections of 3D environments [199,200] while

others looked at blending the method with other planning methods (such as heuristics[201] or RRT[50]) to increase the path safety[50,201].

Octree- Octrees are a subset of the approximate cell decomposition method reviewed in Chapter 2. Using octrees, the free space in a given environment is further decomposed into 8 cubic cells in multiple iterations, until a termination criterion is reached, or a cell is located completely inside or outside the obstacle space. Depending on the desired resolution for the decomposition, the method can have low to high computation time simultaneously trading off the optimal path found on the cellular map. The method is extensively used in the literature for collision avoidance in robot motion planning problems [56,202,203]. Some scholars went farther and combined octrees with other planning methods such as Ant Colony to benefit from both the collision avoidance capabilities of octrees and computational efficiency of Ant Colony [204].

Dubins- Named after its developer [205], Dubins path is the shortest curve that connects two points in a plane with a constraint on the radius of curvature and known velocity vectors (tangents) at the two points. Dubins further proved that for 2D Euclidean planes, the curve is continuously differentiable and the path consists of no more than three segments each of which is either an arc of a circle with the radius no greater than the set curvature (constraint) or a line segment [205]. Other researchers looked into adopting the method for 3D path planning by considering different planes for the initial and final configurations [206]. As a result, Dubins paths are commonly used to address UAV path planning in cluttered environments with known UAV configurations at the initial and final points of flight [207,208]. Figure 4.8 shows an example of a Dubins path with an initial

position at (0,0,0) and a final position at (51,18,51). As can be seen in this figure, the curved path consists of three segments: an initial arc, a straight line segment, and a final arc that lands on the final point of the path, consistent with Dubins' proof.

Contrary to roadmaps and cell decompositions where the workspace is mapped to a connectivity graph, akin to the discretization of the workspace, Dubins paths are continuous parametrized curves with predefined initial and final orientations (or configurations). In addition, due to considering velocity and position of the moving point (e.g. UAV), Dubins' method is a suitable candidate for dynamic and time-dependent routing where in addition to the length of the path, the time it takes to complete the path may also need to also be minimized.

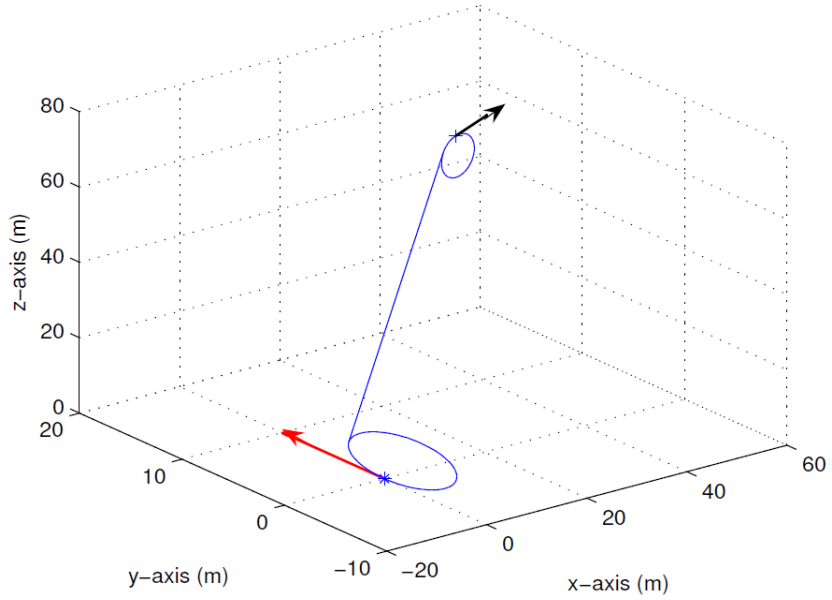


Figure 4.8: An example of a Dubins path

In the next section, a comparison of the 3D path planning methods discussed in this chapter is provided followed by a discussion of their limitations which leads us to the research questions to be addressed in the remainder of this study.

4.4 Comparison of path planning methods

The planning methods discussed in this study are classified as in Figure 4.9 based on their approach to solving the problem and their optimization models. The three main classes are (1) representation-based methods, which make use of the graphical representation of the workspace, (2) reasoning-based methods that follow a logical instead of the geometric approach to solving the problem, and (3) hybrid, which uses a combination of the two previous methods to benefit from the advantages of both.

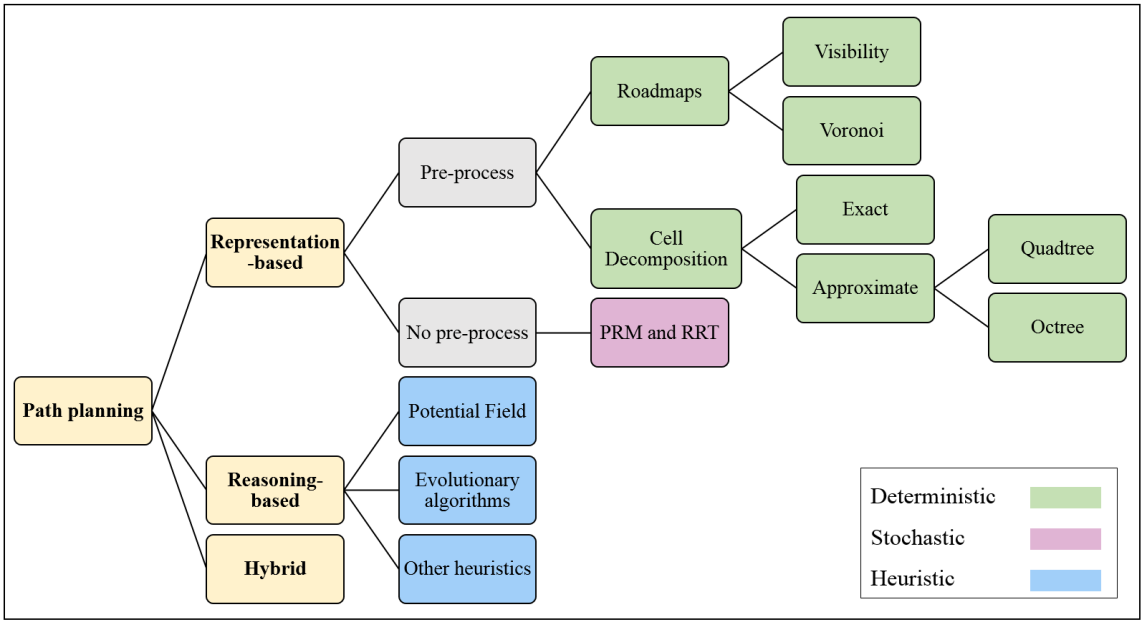


Figure 4.9 Taxonomy of path planning methods

The representation-based methods are further broken into two groups: methods that only generate a free space graph (pre-processes) and usually need a post-process that finds

the solution on the graph and methods that not only generate the graph but are also capable of determining the shortest path on the graph in an all-at-once approach.

The methods are also color-coded following the legend on the bottom right corner of the figure based on the optimization models they use to address the problem. Three themes are found in the planning methods reviewed in the literature: deterministic models, heuristic models, and stochastic models. *Deterministic* models, as introduced in Chapter 3, generate the same fully determined output per each execution of the algorithm when the same input is provided. If an optimal solution exists, deterministic models can find it and prove its optimality.

Heuristic models, on the other hand, define functions (e.g. fitness function in GA or potential functions in PF) to generate and score optimal solutions. The goal in heuristics is to find a solution with acceptable accuracy or optimality degree more quickly, which results in making approximations. Heuristics cannot guarantee to find the global optimum and may trap at local optima or take longer to converge to a global optimum.

When some levels of randomness exist in the problem, *stochastic* methods are the best candidates for the mathematical model. Random variables are often used in the mathematical model of a stochastic optimization problem and instead of a single output, a distribution of possible outcomes may be generated as the solution. Unlike, heuristics that cannot provide proof of optimality, stochastic optimization methods can provide and prove the optimal solution with a known probability.

Heuristic and stochastic approaches are the two widely used classes of methods to address 3D routing which provide an approximate solution. The popularity of these

methods comes from their computational efficiency that could overshadow the accuracy of the solution. Among the three classes of optimization methods, only deterministic methods are capable of finding the exact solution, though their time complexity can significantly increase as they explore the solution space more comprehensively. As shown in Figure 4.9, all representation-based methods, except PRM and RRT families of methods, are deterministic.

Visibility methods are the most exact, among the deterministic methods, for the shortest path problem in 3D. Constructing the complete visibility graph in 3D workspaces is, however, computationally expensive, if at all possible. Hence, as discussed, the available methods can only address special cases (such as specific shapes [179,182], vertical obstacles [9], or only one convex polyhedron[24]) or generate approximate solutions by subdividing the obstacle edges [15,30,178] and therefore restricting the solution space. Table 4.1 below summarizes the studies on 3D visibility-based planning methods.

Table 4.1 Comparison of 3D visibility-based planning methods

Methodology	Reference	Limitations
Subdivision of obstacle edges	Lozano-Pérez & Wesley [13], Papadimitriou [15], Clarkson [30],	- Approximate solution - Computationally expensive
Multi-level 2D visibility graph	Gewali [178], Frontera [180]	- Approximate graph - Vertical objects only
	Kuwata and How [209]	- Approximate path - Vertical block obstacles
Projection on plane	Jiang [179]	- Specific shapes (pyramids, cubes)
	Huang [210], Omar and Gu [181]	- Approximate 2D graph using rotational planes - Vertical block obstacles
Visibility segment graph	Tran [9]	- Approximate Euclidean length - Vertical block objects

Shown in this table are the limitations of the visibility based methods which are one of the two categories: approximate paths on subdivided edges or special shapes/topologies of obstacles.

4.5 Research objectives- Part II

With the discussed limitations, a deterministic approach is required that does not subdivide obstacle edges and can apply to obstacles not limited to vertical or specific polyhedral shapes. Based on this discussion, the objective of the second part of this research is to develop and test a geometric-based and deterministic approach based on the visibility notion to generate optimal solutions to the 3D path planning problems. The focus is more on the optimality of the solution than the computation time while, when possible, speedup techniques are also implemented.

As mentioned, the waypoints of the piecewise shortest path between start and goal points in a 3D cluttered space lie on the edges of the obstacles [24]. Therefore, an approach is sought that can produce the optimal edge sequence to be followed by the path and the optimal locations of turning points of the path on those edges. Different paths found are then appended together to form the 3D visibility graph and later Dijkstra's search algorithm is applied to yield the shortest path on the graph.

Chapter Five
3D PATH PLANNING PROBLEM SETUP, DEFINITIONS, AND FORMULATION

This chapter provides an overview of the definitions of the fundamental terms and assumptions used to simplify the problem. Other preliminary steps taken toward constructing the 3D visibility graph are also explained here. These steps include the workspace geometric representation, data types/structures used, and finally the mathematical formulation of the problem.

5.1 Definitions of fundamental terms

See the following definitions for the terms used in the problem statement.

Definition 1. As defined by O'Rourke [10], a polyhedron is "a region of space whose boundary is composed of a finite number of flat polygonal faces, any pair of which are either disjoint or meet at edges and vertices."

Definition 2. By ***Definition 1***, a convex polyhedron is the one whose faces are all convex polygons.

Definition 3. Intersection (or collision) between a line segment and a polyhedron occurs if and only if the line intersects with at least two faces of the polyhedron at points with different coordinates. Note that for a convex polyhedron, an intersection occurs if the line segment intersects with exactly two faces of the polyhedron at two distinct points.

By this definition, in a 3D space, the direct path connecting two points X and Y is not collision-free if and only if the line segment $\overline{XY}$ intersects with the interior of at least one obstacle, that is:

$$\exists P_i \subset W : \overline{XY} \cap \text{int}(P_i) \neq \varnothing \tag{5.1}$$

Where P_i is the ith obstacle, W is the 3D workspace, and $int(P_i)$ denotes the interior of the i[th] polyhedral obstacle.

Definition 4. A path R from the start (s) to the goal (g) is said to be the *shortest collision-free* if it is the shortest path among all the collision-free paths from s to g.

5.2 Assumptions

Several assumptions are made to model the problem mathematically and geometrically to be able to solve it as an optimization problem. The assumptions are:

Assumption 1. Obstacles are convex polyhedra (please see ***Definition 1*** and ***Definition 2***).

Assumption 2. The location and geometry of all the obstacles are known a priori.

Assumption 3. The obstacles are static (their location does not change at any time) and disjoint, meaning no two obstacles touch. If any two obstacles touch, they are considered one obstacle. Since this new obstacle becomes non-convex, the methods developed in this research may not apply to those cases.

Assumption 4. The obstacles are modeled using tessellations that approximate each object's surface by triangular polygons.

This assumption also simplifies the collision check between a line segment and a polyhedron as explained in the next chapter.

Assumption 5. The obstacles can take any arbitrary convex shape and geometry. The shape is not constrained as long as it satisfies the definition of a convex polyhedron.

Assumption 6. The start and goal points of the path are not interior to any polyhedral obstacle.

Note that this assumption together with ***Assumption 3*** implies that there is always a collision-free path between the given points amidst the obstacles.

Assumption 7. A path can touch the boundary of an obstacle or its configuration space; however, traveling through the interior of any of the obstacles is not allowed (see ***Definition 3***).

Assumption 8. Should the environment of the routing problem be enclosed, collisions will be avoided with the walls of the enclosure.

Assumption 9. The wire routing problem is modeled as a 3D problem; however, the algorithm is capable of routing in 2.5D workspaces (such as robots moving on a floor plan or UAVs flying in an urban environment). For an exact algorithm for 2D routing problems, readers are referred to [83].

Assumption 10. Collision-free paths between the given points are piecewise linear if the baseline connecting the endpoints regardless of the obstacles intersects with at least one obstacle. Otherwise, the shortest collision-free path is trivially the line segment connecting them.

Although path smoothening is possible using B-Spline or NURBS parametric models of curves, the output path in this research is piecewise linear and no post-processing steps are taken to smoothen the path; therefore, focusing on the exact shortest paths.

Assumption 11. The given environment is the configuration space of the problem. Therefore, the agent that is routing through obstacles is assumed to be a point.

This is critical in case the agent has a two or three-dimensional geometry (such as a robot or a wire with circular cross-section) which needs to be shrunk to a point and grow the obstacles correspondingly.

5.3 Modeling the workspace: representation and exchange format

The geometric representation of the workspace is at the core of the proposed path planning method. An appropriate geometric model contributes to speeding the collision detection between the path and the objects. The identification of the intersecting obstacles is the basis of the algorithm suggested in this manuscript. Therefore, it is of high importance to select the geometric representation and CAD format that best describe the geometric data of the workspace and facilitate the geometric operations including collision detection.

Different geometric representation paradigms exist to define and model 3D objects, the two most common of these are Constructive Solid Geometry (CSG) and Boundary Representation (B-rep) [211].

According to Zeid [211], CSG benefits from primitives (or building blocks) that can be manipulated using Boolean operations to generate more complex 3D models. These primitives are typically basic shapes such as rectangular block, cylinder, cone, plane, and sphere. B-rep, on the other hand, is based on the idea that a physical object is bounded by a finite number of faces that are closed (i.e. no breaks or holes exists on their boundary) and orientable (i.e. the two sides of a face are distinct by having surface normals pointing to opposite directions). Therefore, faces, vertices, and edges are the building blocks of B-rep that construct a physical object [211]. As path planning methods are generally

concerned with faces, edges, and vertices, the suitable geometric representation for this application is seemingly B-rep, and therefore this representation is adopted in this research.

After the geometric representation is specified, the next step is to decide how the geometric data of this representation is to be stored and reported. CAD software packages provide a variety of data formats. An appropriate data format is the one that could be easily exchanged between these packages. Accordingly, to overcome the interoperability issues of using platform-specific 3D models (proprietary formats), this research benefits from open-source (neutral) formats. These are the formats of 3D models that are common among different CAD software packages. Examples of these neutral formats include IGES, XBF, SET (for shape data exchange), and STEP and PDES (for product data exchange) [212].

Table 5.1 Common neutral CAD formats [213]

Data format	Geometric representation	Shape and product data	Application
IGES	CSG and B-rep	Surface geometry and color data	High precision engineering (e.g. aerospace)
STEP	CSG and B-rep	Surface geometry, topology, and appearance data	High precision engineering
OBJ (neutral in ASCII format)	Approximate and precise mesh in B-rep	surface geometry, appearance data	3D printing, 3D graphics
STL (STereoLithography)	Approximate mesh in B-rep	Surface geometry only	3D printing, CAM
VRML (Virtual Reality Modeling Language)	Approximate mesh in B-rep	Surface geometry and appearance data	Internet and the web
COLLADA	B-rep	surface geometry, appearance data, animation	Graphics (gaming and film industries)

As discussed by Owen and Bloor [212], some of the issues with the initial data exchange formats included storage and accuracy. The early versions of IGES, for example,

required more space to store the same data than the native CAD formats did. In addition, the accuracy of the transferred data in the early formats could be diminished. To alleviate these issues, more recently other formats are introduced. Table 5.1 provides the properties of some of the common neutral formats used both commercially and scientifically as described in [213].

As can be seen in Table 5.1, OBJ, STL, and VRML are the formats that use approximate meshes in a B-rep geometric representation. Approximate mesh formats render faster than precise mesh formats. STL is one of the primary tessellated based formats and is widely deployed in additive manufacturing industries. It approximates the surfaces of solid models by triangular meshes. An STL file of a solid model includes the X, Y, and Z coordinates of each triangle's vertices as well as the outward normal vector to the surface of that triangle. An edge must be shared by no more than two triangles.

For 3D printing applications, OBJ is gaining more attraction these days as it encodes color and appearance data in addition to the shape, which is useful if parts with multiple colors or textures are to be printed. Further, its approximate mesh is not limited to only triangular surfaces. It can, for example, use quadrilateral meshes to approximate surfaces. VRML is another tessellated based format that also encodes appearance data and is best for web applications.

Following **Assumption 3**, obstacles are modeled using a tessellated format in CAD software. Thus, STL and VRML are two candidates for the format of the workspace data storage. As noted by Fadel and Kirschman [214], STL causes loss of accuracy due to round-off errors when computing the approximations of curved surfaces by a series of triangles.

This error results in the generation of multiple very close points despite pointing to the same single point. This could cause a hole inside a tessellated object since the edge that two triangles share is no longer common due to different coordinates of the "common points". This situation can be seen in Figure 5.1.

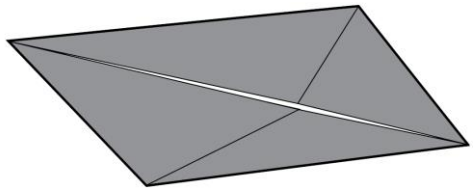


Figure 5.1: Round-off error in tessellations

Another issue relates to the chordal tolerance in a triangulation. Chordal tolerance, as defined by Fadel and Kirschman, is the distance from the surface of a solid model to the vector that represents a side of the triangle. To improve the accuracy of the tessellation, one needs to reduce the chordal tolerance by increasing the number of triangles.

The STL creates a tessellated format by getting the coordinates of the points of each triangle and representing them. Hence, the coordinates of two points would be repeated as an edge is shared between the two triangles. The repetition of the coordinates of a point may increase the chance of getting the round-off error at that point. Additionally, it results in more space required to store the large data of STL format. VRML, on the other hand, first gets all the points and then creates the triangles, reducing the possibility of the round-off error. This format, however, does not come with the normals to the triangles. If such normals are desired for an application, the user has to compute them numerically. Figures

5.2 and 5.3 below show, in ASCII format, an STL and a VRML representation of data of a cube modeled in SolidWorks, respectively.

```
solid cube
   facet normal -1.000000e+00 -0.000000e+00 -0.000000e+0
      outer loop
         vertex 0.000000e+00 4.000000e+01 4.412251e+01
         vertex 0.000000e+00 4.000000e+01 0.000000e+00
         vertex 0.000000e+00 0.000000e+00 4.412251e+01
      endloop
   endfacet
   facet normal -1.000000e+00 0.000000e+00 0.000000e+00
      outer loop
         vertex 0.000000e+00 0.000000e+00 4.412251e+01
         vertex 0.000000e+00 4.000000e+01 0.000000e+00
         vertex 0.000000e+00 0.000000e+00 0.000000e+00
      endloop
   endfacet
   facet normal 0.000000e+00 0.000000e+00 1.000000e+00
      outer loop
         vertex 3.985081e+01 4.000000e+01 4.412251e+01
         vertex 0.000000e+00 4.000000e+01 4.412251e+01
         vertex 3.985081e+01 0.000000e+00 4.412251e+01
      endloop
   endfacet
   facet normal 0.000000e+00 0.000000e+00 1.000000e+00
      outer loop
         vertex 3.985081e+01 0.000000e+00 4.412251e+01
         vertex 0.000000e+00 4.000000e+01 4.412251e+01
         vertex 0.000000e+00 0.000000e+00 4.412251e+01
      endloop
   endfacet
   facet normal 1.000000e+00 0.000000e+00 -0.000000e+00
      outer loop
         vertex 3.985081e+01 4.000000e+01 0.000000e+00
         vertex 3.985081e+01 4.000000e+01 4.412251e+01
         vertex 3.985081e+01 0.000000e+00 0.000000e+00
      endloop
   endfacet
   facet normal 1.000000e+00 0.000000e+00 0.000000e+00
      outer loop
         vertex 3.985081e+01 0.000000e+00 0.000000e+00
         vertex 3.985081e+01 4.000000e+01 4.412251e+01
         vertex 3.985081e+01 0.000000e+00 4.412251e+01
      endloop
   endfacet
   facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
      outer loop
         vertex 0.000000e+00 4.000000e+01 0.000000e+00
         vertex 3.985081e+01 4.000000e+01 0.000000e+00
         vertex 0.000000e+00 0.000000e+00 0.000000e+00
      endloop
   endfacet
   facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
      outer loop
```

Figure 5.2: Sample STL representation of CAD data in ASCII format

```
#VRML V1.0 ascii
Separator {
MaterialBinding {
value OVERALL
}
Material {
ambientColor [
 0.792157 0.819608 0.933333
]
diffuseColor [
 0.792157 0.819608 0.933333
]
emissiveColor [
 0.000000 0.000000 0.000000
]
specularColor [
 0.396078 0.409804 0.466667
]
shininess [
 0.400000
]
transparency [
 0.000000
]
}
Coordinate3 {
point [
 -0.016279 0.000000 -0.013398, -0.016279 0.000000 0.030724, -0.016279
0.040000 -0.013398, -0.016279 0.040000 0.030724, 0.023571 0.000000 -
0.013398,
 0.023571 0.000000 0.030724, 0.023571 0.040000 -0.013398, 0.023571
0.040000 0.030724
]
}
IndexedFaceSet {
coordIndex [
 3, 2, 1, -1, 1, 2, 0, -1, 7, 3, 5, -1,
 5, 3, 1, -1, 6, 7, 4, -1, 4, 7, 5, -1,
 2, 6, 0, -1, 0, 6, 4, -1, 7, 6, 3, -1,
 3, 6, 2, -1, 4, 5, 0, -1, 0, 5, 1, -1
]
}
}
```

Figure 5.3: Sample VRML representation of CAD data in ASCII format

Despite the discussed challenges with STL, since in this research appearance data is of little importance whereas surface normals are required for the next set of calculations, the selected format is STL to avoid further calculations of the normals by the cross products of the vectors defining triangle edges.

5.4 Data types and structures

The types and structures of the data to be stored and manipulated affect the computational performance of an algorithm. Therefore, in this work, attempts are made to deploy the data types that facilitate the geometric and algebraic operations on the data of the tessellated objects. Here, an explanation of the data types as well as structures used is provided.

5.4.1 Data types

The three main types of data used in implementing the developed algorithms in this research include **floating-point**, **integer**, **Boolean**, and **characters**. The geometric data of the objects imported from the CAD software is composed of an n-by-3 matrix of coordinates of vertices, where *n* is the number of vertices per object, and an m-by-3 matrix of edges, where *m* is the number of triangles in the triangulated object. The data of the vertex coordinates has the type floating-point in double-precision while integer data type is used in the matrix of tringles that denote the ids of the vertices connected in each triangle. Boolean is another type of data used extensively in the algorithm implementation especially where the output of the operation is binary. For example, the output of whether or not an object is on the way is reported in Boolean. Last but not the least, characters are used to create the vertex ids. The id of each vertex in an object is denoted as "$a.b$", where a is the number associated with the object, and b denotes the vertex number in that object. For instance, following the discussed identification method, the id "*2.31"* refers to the thirty-first vertex in the second object.

Now that the data types used in this research are introduced, the structures to organize this data for the most effective usage need to be explored.

5.4.2 Data structures

A list of data structures with their brief explanation is provided in this section starting with the most basic structure, arrays.

5.4.2.1 Arrays

Arrays are one of the basic data structures in every programming language. An array could store vector data of any primitive type so long as the type of the stored data is uniform. Reference to the data in each memory location is made by the index of the array element.

Matrices could be created by concatenating multiple arrays of the same dimensions either in rows or columns depending on the dimension of the array. Arrays are indeed one-dimensional matrices. *Cell arrays*, unlike matrices, are structures that could store and organize different data types including numerical and text data. The data in a single cell must, however, be of the same type. Arrays that have more than two dimensions are called *multidimensional arrays* [215]. In these array types, the first and second dimensions are associated with the row and column numbers while the third dimension is usually referred to as the *page* [215]. Figure 5.4 shows an example of a multidimensional cell array. Shown in the figure, each cell can contain data of different types. Additionally, it is noteworthy that cell arrays can have cells with different sizes as opposed to matrices. An example is shown in Figure 5.4 where cells 1,1,1 and 2,2,1 have dimensions 2x2 and 1x1 respectively.

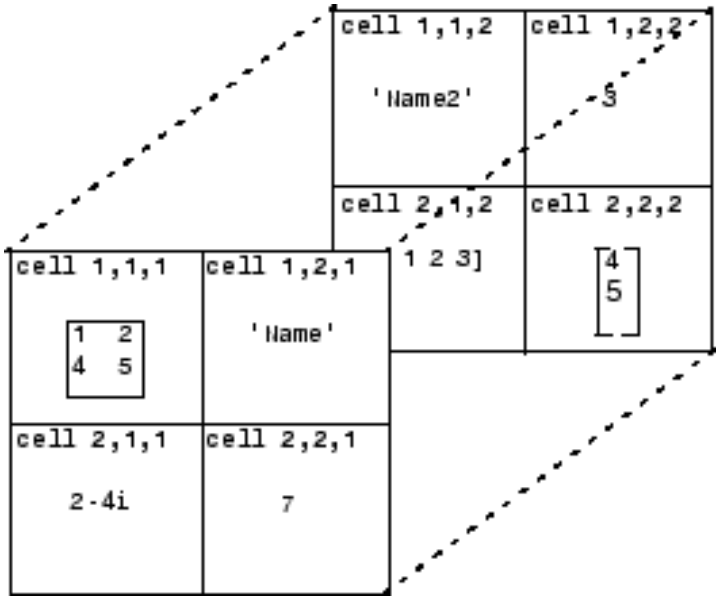


Figure 5.4: Multidimensional Cell Array [215]

Another important array type used extensively in this research is the *dynamic array*. A dynamic array is a variable-size array used when predefining an array is not feasible or the array size is not known a priori. For example, in creating a path consisting of a sequence of connected points, the number of waypoints is not known in advance. Therefore, defining the path as a dynamic array helps to construct the path by appending the next waypoint to the array at each iteration until the goal point is reached.

5.4.2.2 Record or struct

A *struct* (or structure) is a form of organizing data that consists of several fields. A "struct" groups the related data using these fields [216]. Each field can contain data of any type similar to a cell array. Both struct and cell can contain heterogeneous data. However, the two data structures differ in how they provide access to the data of each field or cell. To access the data in a field of a structure one should use dot notation in the form

"`structName.fieldName`" [216]. In a cell array, on the other hand, access to a cell is provided by numeric indexing.

In this research, a struct is used to store the geometric data of the objects. Some of the fields in this struct are "`vertices`", which contains the *x*, *y*, and *z* coordinates of the vertices, "`faces`", which contains the surface triangle data by denoting the ids of the three nodes of each triangle, and "`normals`" to the triangular surfaces.

5.4.2.3 Linked lists

Lists are structures wherein data is not stored in contiguous memory locations. That means unlike arrays where data in each element is easily accessible by numeric indexing, access to the data of a random element in a list may require extra effort in implementing procedures and routines to perform such operations since linked lists can only provide sequential access from the first node[217]. As shown in the below figure, elements of lists are connected via pointers (links). Therefore, a linked list consists of nodes that contain a field for data and a link to the next node of the list [217].

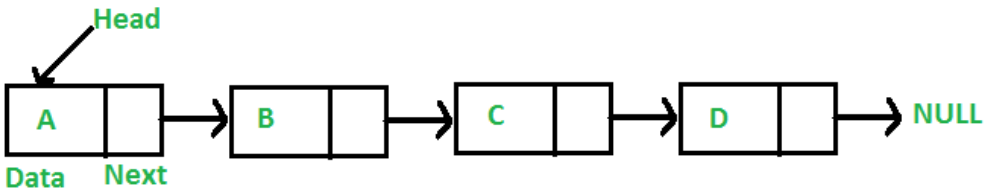


Figure 5.5: Graphical representation of a linked list[217]

Despite the difficulty in accessing random elements of lists, insertion and deletion of elements of lists are easier than arrays since there does not exist continuous memory locations for elements of lists. This advantage of lists makes them a good candidate for

storing the data of the sequence of edges of obstacles to be traversed to achieve the goal point in the 3D path planning problem.

5.4.2.4 Graphs

Graphs are structures that contain the node (or vertex) and edge data of a known network. Edge data shows which nodes are connected in the network. This data structure is critical in any routing problem. Since geometric based planning problems mainly work based on the construction of graphs that are searched for the safe shortest path, the graph data structure needs to be defined and created correctly. In this research, the nodes of the graph include the start and goal as well as the waypoints identified on the obstacle edges for 3D path planning.

5.5 Problem formulation

The path planning problem considered in this research aims at minimizing the length of one-dimensional components (cables, wires, tubes, and hoses) in electromechanical systems. Therefore, the problem can be defined as:

Primary problem: Given an environment scattered with static convex polyhedral obstacles and a start and a goal point, the objective is to develop a deterministic geometric-based optimization algorithm that finds a minimum length path between the two points while avoiding collisions with any obstacles.

Suppose there are f polyhedral obstacles, P_i, (i = 1, 2, …, f) scattered in the 3D space. Following the above-mentioned assumptions, the problem is to construct the free space defined as Eq.(5.2) in the form of a graph.

$$W \supseteq C_{free} = W \setminus \bigcup_{i=1}^{f} P_i \tag{5.2}$$

In Eq.(5.2) C_{free} denotes the free space as a subset of the workspace, W, which could be generated by taking the complement of the union of all obstacles. Additionally, the contact space (C_{ontact}) can be defined as in Eq.(5.3) to capture the boundary space of the obstacles:

$$C_{contact} = \bigcup_{i=1}^{f} \partial P_i \tag{5.3}$$

Where ∂P_i denotes the boundary of the polyhedron i.

The graph G, defined by its set of vertices (V) and edges (E), is desired to capture a collision-free subset of the workspace.

$$G \subseteq C_{free} \cup C_{contact} \ , \ G = \{V, E\} \tag{5.4}$$

Definition 5. The set V is a set of vertices of the free space graph. These vertices are on the edges of the intersecting obstacles and augmented by the start and goal points of the path.

Definition 6. The set E is a set of edges of the free space graph that connect the vertices in V.

After the desired graph is constructed, the shortest path needs to be found by exploring the graph using Dijkstra's algorithm. The formulation of Dijkstra's problem is as below:

Secondary problem: Given the connected graph $G = \{V, E\}$, find the shortest path between nodes 1 and m, $m \neq 1$ such that

$$min \sum_{(i,j)\in G} C_{ij}X_{ij}$$

Subject to : $\sum_{\{j:(i,j)\in G\}} X_{ij} - \sum_{\{i:(i,j)\in G\}} X_{ji} = \begin{cases} 1 & i = 1 \\ 0 & i \neq 1, m \\ -1 & i = m \end{cases}$

Where: $X_{ij} = \begin{cases} 1 & if\, e_{ij}\ is\ in\ the\ path \\ 0 & otherwise \end{cases}$

C_{ij}: cost, the Euclidean length of arc e_{ij}

Chapter Six
INTERSECTION DETECTION ALGORITHMS

Path planning in cluttered environments requires avoiding intersections with obstacles. Therefore, intersection detection is at the core of any path planning problem in the presence of obstacles. Indeed, the first step in constructing the free space graph in geometric-based path planning approaches is detecting the intersections between the direct path (the path connecting the start and goal) and the obstacles. If no intersection is detected, the shortest path is trivially the straight line between start and goal. Otherwise, the path needs to be re-routed until a collision-free path can be achieved. This chapter is allocated to the methods and algorithms of detecting the intersections between objects in 3D space. Readers are referred to ***Definition 3*** for an intersection between a line segment and object in 3D space. In this chapter, two types of intersections are discussed: (1) line segment-triangle intersection, which contributes to identifying the intersecting objects, and (2) triangle-triangle intersection, which is an intermediary step during the construction of the free space graph (discussed in the next section). Triangulated objects highly simplify the intersection calculations as instead of computing the intersections between polyhedra and a line (or between more than two polyhedra), it suffices to determine intersections between a line segment and several triangles (or between multiple triangles).

6.1 Line segment-triangle intersection

Since the shortest path in 3D space passes through the edges of the intersecting obstacles, the first step in constructing the free space graph is to identify the intersecting obstacles. As the obstacles are modeled using tessellations, it suffices to find intersections

between the line segment connecting the start and goal points and all the triangles in an object.

The intersection detection in this research undergoes two main steps: (1) filtering the out-of-bound obstacles and (2) checking for line segment-triangle intersections only for obstacles whose coordinates overlap with the coordinates of the endpoints of the line segment. In what follows the steps to identify the intersecting obstacles alongside the pseudocode for intersection detection are presented.

Step 1: Transformation of the coordinate system

After the workspace is modeled using a tessellated-based solid model in CAD software, the data of the obstacles is imported in the path planning environment. MATLAB is selected for the computational environment of the path planning problem in this research. Therefore, the solid models of the obstacles are imported in MATLAB. Since STL is the selected format for data exchange of the obstacles' solid model, the data needs to be converted to MATLAB-compatible STL data. To achieve this, "`stlRead`", a MATLAB function developed by Micó [218], is used to read the STL data of each obstacle. The data is then stored in a struct whose fields are vertices, faces, and normals to faces the obstacles' faces.

After the geometric data is retrieved, the coordinates of the start and goal points are inputted to the program. Before the intersection check is performed, the coordinate system of the workspace is transformed such that the start point is coincident with the origin (0,0,0) and the start-goal vector is lined up with the *Z*-axis of the coordinate system. Though this may seem like an additional computation that could potentially affect the overall

computational efficiency of the method, this step is essential in simplifying the intersection computations and the determination of the edge sequence that follows the intersection. The latter is discussed in the next section.

The coordinate transformation is performed in the following order. First, a linear translation is required to coincide with the start point with the origin of the coordinate system. To enable matrix multiplication that leads to the desired translation, homogeneous coordinates are used for the points to be transformed. The homogeneous coordinates can be created by augmenting the original coordinates. Augmentation adds a nonzero fourth coordinate to the 3D coordinates of a point. For simplicity, the fourth coordinate is often set equal to one.

Using the homogenous coordinates, the translation matrix is given as:

$$T_T = \begin{bmatrix} 1 & 0 & 0 & \Delta X \\ 0 & 1 & 0 & \Delta Y \\ 0 & 0 & 1 & \Delta Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

With this translation matrix, a point P can be translated by the amounts ΔX , ΔY , ΔZ along the *X*, *Y*, and *Z* axes, respectively, using the equation:

$$[P'] = [T_T].[P]$$

Or

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \Delta X \\ 0 & 1 & 0 & \Delta Y \\ 0 & 0 & 1 & \Delta Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.2}$$

To translate the coordinates such that the start point and the origin coincide, one needs to set $\Delta X = -x_s$, $\Delta Y = -y_s$, and $\Delta Z = -z_s$ where x_s, y_s, and z_s are the 3D coordinates of the start point.

After the coordinates are successfully translated, the start-goal vector needs to be rotated to lie within the YZ plane. This rotation is performed about the Y-axis and the angle of rotation is found using the projection of the vector onto the XZ plane as illustrated in Figure 6.1.

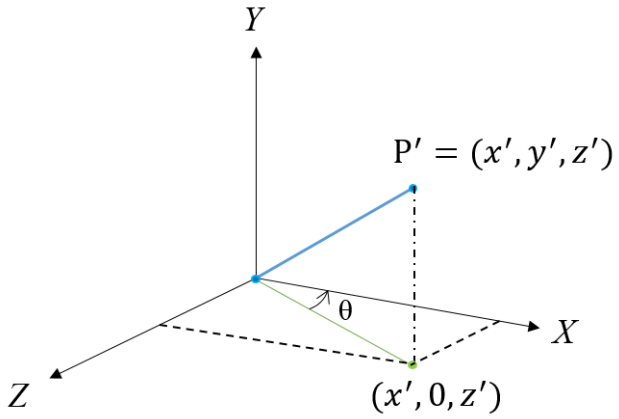


Figure 6.1: Rotation about the Y-axis

As can be seen from Figure 6.1, the projection of the vector on the XZ plane makes an angle θ with the X-axis. To project the vector onto the YZ plane, a rotation about Y-axis is required, the angle of which must be $-(90-\theta)$. Eq.(4.3) provides the rotation matrix about the Y-axis. In order to rotate the vector to make it lie on the YZ plane, one should substitute θ with $-(90-\theta)$ in Eq.(6.3).

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

From Figure 6.1, it can be seen that the angle θ can be found as:

$$\sin\theta = \frac{|z'|}{\sqrt{x'^2 + z'^2}} \tag{6.4}$$

After the vector is rotated and lies in the YZ plane, it must be rotated a second time to line up with the Z-axis. To do so, the second rotation should be performed about the X-axis. Figure 6.2 illustrates this rotation and the rotation angle.

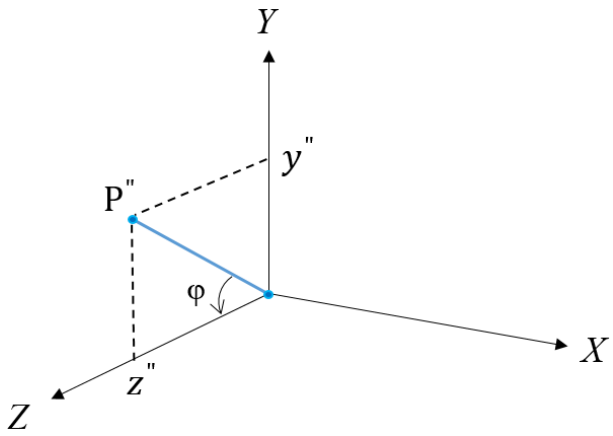


Figure 6.2: Rotation about the X-axis

As can be seen from this figure, a rotation about the X-axis in the amount of φ will put the desired vector along the Z-axis. The rotation matrix and angle are given as in equations (6.5) and (6.6).

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

$$\cos\varphi = \frac{|z''|}{\sqrt{y''^2 + z''^2}} \tag{6.6}$$

After the angles are calculated, the three transformations can be performed at once using the below equation.

$$[P_2] = [T_T].[R_y(\theta - 90)].[R_x(\varphi)].[P_1] \tag{6.7}$$

This transformation is performed on the coordinates of every vertex in all obstacles as well as the start and goal points. A sample coordinate transformation is depicted in Figure 6.3 with 3 obstacles.

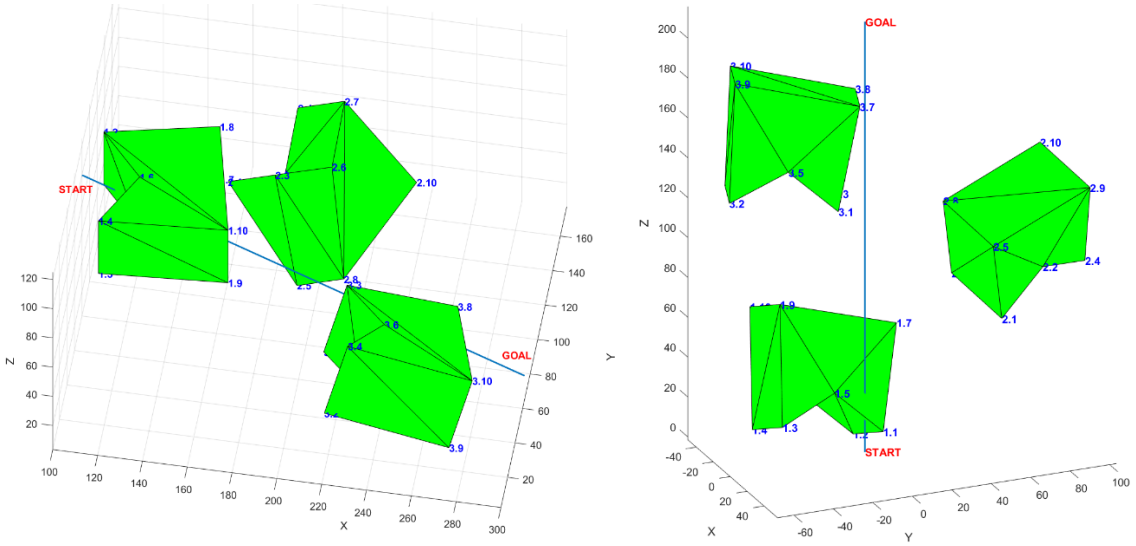


Figure 6.3: Sample coordinate transformation

Step 2: Filter I: out-of-bound obstacles

After all the objects are transformed, it is time to filter out the ones that are out of the scope of the start-goal line. To achieve this, the Axis-Aligned Bounding Box (AABB) of each object is created. The orthogonal bounding box is created by the minimum and maximum x, y, and z values of the vertices of an object as shown in Eq (6.8).

$$AABB = \begin{bmatrix} x_{\min} & x_{\max} \\ y_{\min} & y_{\max} \\ z_{\min} & z_{\max} \end{bmatrix} \tag{6.8}$$

The coordinates of the bounding box should be checked against the coordinates of the start-goal line in the transformed coordinate system. Since the start-goal line is aligned with the Z-axis, it is only necessary to initially check the Z coordinates of each object. If it is found that the maximum z coordinate in the AABB is negative (less than the z coordinate of the start point) or the minimum z coordinate in the AABB exceeds the z coordinate of the goal point, the object is entirely out-of-bound of the line and therefore there is no chance that the line intersects the object. This situation is shown in Figure 6.4. If, however, it is found that the z coordinates of the object are between the z coordinates of the start and goal points, the next step is to check for the x and y coordinates.

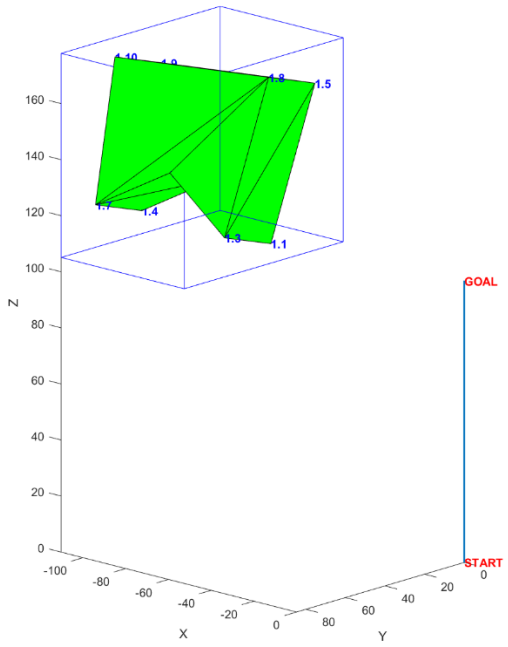


Figure 6.4: Sample out-of-bound AABB

Even if the AABB is within the boundary of the start-goal line, the object may lie entirely at one side of the line, zeroing the chance of intersecting the obstacle. For the object to lie on one side of the line it is sufficient to check whether all x coordinates or all y coordinates have the same sign. This helps since we know the line is aligned with the Z-axis. Therefore, if all x coordinates are positive (or negative) the AABB has no chance to intersect the line. To check this, we merely need to compute the multiplication of minimum and maximum x (or y) coordinates:

if $(x_{min} \cdot x_{max} \geq 0)$ or $(y_{min} \cdot y_{max} \geq 0)$
    AABB ← non-interfering
else
    AABB ← interfering
endif

An example of an AABB that lies at one side of the line is shown in Figure 6.5.

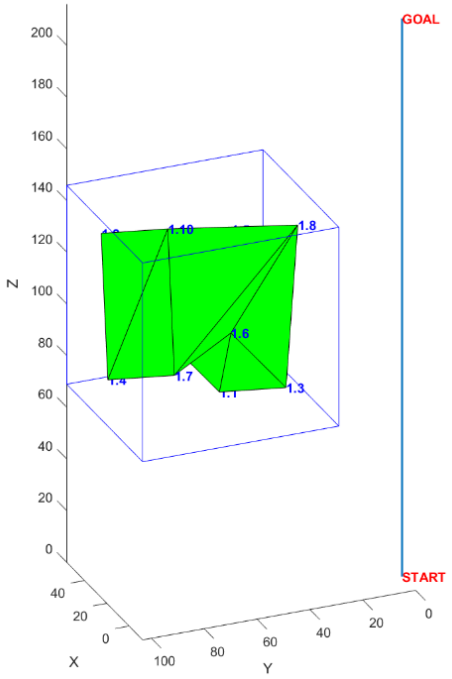


Figure 6.5: Object lying at one side of the line

Step 3: Filter II: ray-triangle intersection check

After filtering out the out-of-bound objects and objects that lie on a side of the line, the final check is to determine if the line intersects the interior of the object that passed all the filters. This step is crucial in separating the intersections from the cases where the line segment touches the object but does not pass through its interior or a case such as the one shown in Figure 6.6.

This is the step where intersections need to be checked between the line segment and all the triangles in the above-mentioned object. Following ***Definition 3***, if the line segment intersects with exactly two faces of the polyhedron at two distinct points, there is an intersection between the line and the object.

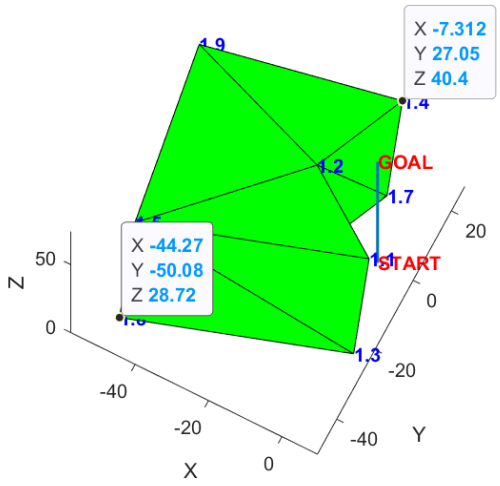


Figure 6.6: Non-intersecting object not filtered in step II

For the line segment-triangle intersection detection, the ray-triangle intersection algorithm developed by Möller and Trumbore [219] is used in this research. The algorithm takes a ray (defined by its origin and normalized direction) and a triangle (defined by its

vertices) as inputs and transforms the origin of the ray. The output of this transformation is a triplet (t, u, v) where t is the distance to the plane to which the triangle belongs and u and v are the barycentric coordinates of intersection inside the triangle. The barycentric coordinates of a point on a triangle is given in [219] as:

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2 \tag{6.9}$$

Where V_0, V_1, and V_2 are the three vertices of the triangle and $u, v \geq 0$ and $u + v \leq 1$. Using this equation, the transformation of the origin can be written as[219]:

$$O + tD = (1-u-v)V_0 + uV_1 + vV_2 \tag{6.10}$$

Where O is the origin and D is the direction of the inputted ray. As shown in [219], the re-arrangement of Eq.(6.10) yields a system of linear equations which can be solved to determine the triplet (t, u, v). The rearrangement is given in Eq.(6.11) below.

$$\begin{bmatrix} -D & V_1 - V_0 & V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{6.11}$$

In this research, the implementation of Möller and Trumbore's algorithm developed by Tuszynski [220] in MATLAB is used for the ray-triangle intersection step of the collision detection. A pseudo-code of the overall intersection detection algorithm is also shown in **Algorithm 1**. The pseudo-code assumes the transformations are performed and the objects, as well as the start and goal points, are given in the transformed coordinate system. Although in this research, it is assumed that all objects are convex, the explained intersection detection algorithm similarly applies to non-convex objects.

Algorithm 6.1

Input: Workspace objects, start and goal points
Output: an array of the ids for intersecting obstacles

Step 1. Transform the coordinate system such that start-goal line becomes aligned with the Z axis. The new coordinates of start and goal are $(0,0,0)$ and $(0,0,z_2)$ respectively
Step 2.

in-bound ← ∅

for ($i = 1$ to number of objects), **do**:
    Create the OBB for i:
    if ($z_{min} \geq z_2$) or ($z_{max} \leq 0$)
        OBB ← out-of-bound
    else
        if ($x_{min}.x_{max} \geq 0$) or ($y_{min}.y_{max} \geq 0$)
            OBB ← non-interfering
        else
            OBB ← interfering
            in-bound ← i
        endif
    endif
end for

intersected ← ∅

$O \leftarrow (0,0,0)$
$D \leftarrow (0,0,z_2)$
for (j= 1 to size(in-bound)), **do**
    n ← number of triangles in j
    V_0 ← n-by-3 matrix of the x,y,z coordinates of the first vertex of all triangles in j
    V_1 ← n-by-3 matrix of the x,y,z coordinates of the second vertex of all triangles in j
    V_2 ← n-by-3 matrix of the x,y,z coordinates of the third vertex of all triangles in j
    call TriangleRayIntersection **(**O,D**,** V_0, V_1, V_2**)**
    return t
        if ($t = 0$)
            intersected ← j
        endif
end for
return intersected}

6.2 Triangle-triangle intersection

Another type of intersection check that is extensively used in this research is the intersections between two triangulated surfaces. This type of intersection is extensively used throughout the graph construction algorithm. As explained in the next chapter, constructing the free space graph initially requires identifying a sequence of edges of obstacles to be explored. This edge sequence from the path start point to the goal point is determined in multiple steps. In one step, for example, a convex hull is created between the start point and the first intersecting obstacle. The edges that are connected to the start point on this convex hull are then extracted and stored as the first set of edges to be explored. For each of these edges, a plane is created that is perpendicular to the edge and contains the start-goal line. The intersection of this plane and the convex hull is then used to identify the next edges to be added to the edge sequence. Further details of this algorithm and other instances where intersections between surfaces need to be determined are explained in the next chapter.

It is therefore clear that the previous ray-triangle intersection detection method can no longer be applied to identify intersections between surfaces. This section is, therefore, allocated to the explanation of the algorithm used for detecting such intersections.

Moller has developed a fast triangle-triangle intersection test [221] that is suitable for collision detection between 3D triangulated objects. This algorithm works in a rather similar way to his ray-triangle intersection check. Here, however, instead of determining the distance between the ray and the plane of the triangle, he determines the distance from the vertices of the first triangle to the plane of the second triangle. The algorithm finds the

distances by simply substituting the vertices of the first triangle in the plane equation for the second triangle as in Eq.(6.12):

$$D_i = N_2 \cdot V_i^1 + d_2 \quad i = 1,2,3 \tag{6.12}$$

Where N_2 is the normal to the plane of the second triangle (plane equation: $N_2 \cdot X + d_2 = 0$ where X is any point on the plane) and V_i^1 is the ith vertex in the first triangle. N_2 and d_2 can be found using equations (6.13) and (6.14) below.

$$N_2 = (V_2^2 - V_1^2) \times (V_3^2 - V_1^2) \tag{6.13}$$

$$d_2 = -N_2 \cdot V_1^2 \tag{6.14}$$

This calculation is repeated between the plane of the first triangle and the vertices of the second triangle. If the calculated distances, D_i, are not equal to zero and all are found to have the same sign, the triangle lies on one side of the plane. On the other hand, if all D_i distances are equal to zero, the triangle and the plane are co-planar. Otherwise, the planes of the triangles intersect, and their intersection is a line segment, the direction of which can be found from $N_1 \times N_2$. The common intervals between the line of intersection and each triangle determine if the two triangles also intersect. For example, as shown in Figure 6.7, when the intervals on the line do not overlap (Figure 6.7 right), the triangles do not intersect either.

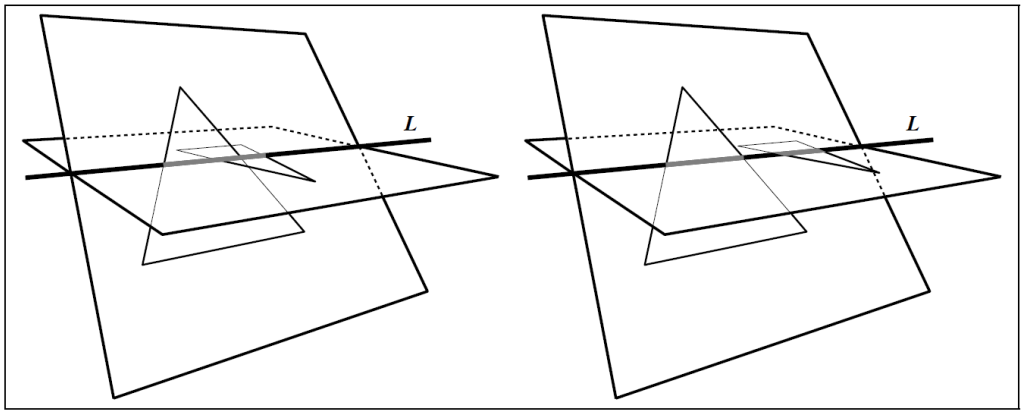


Figure 6.7: Intersection between two triangles [221]

The method goes on to determine the exact surface of intersection which can be represented by triangulated faces and vertices. For further details of the algorithm and the computation of different cases of intersection, readers are referred to [221]. In this research, Tuszynski's implementation of Moller's algorithm in MATLAB is borrowed [222]. Tuszynski has solved an example of intersections between two objects the results of which are shown in Figure 6.8. In this figure, the surfaces of intersection are shown on the right.

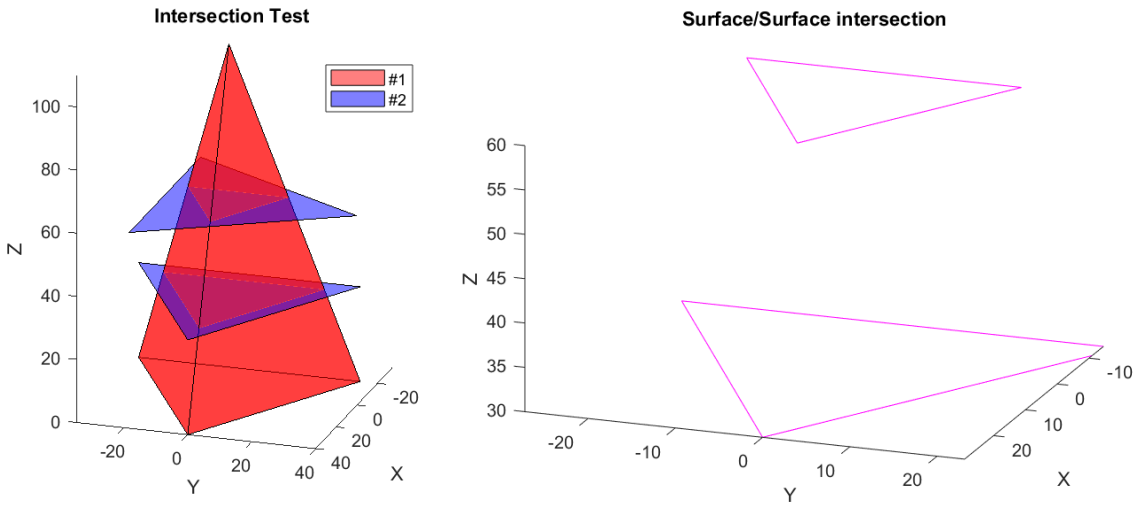


Figure 6.8: Example of Moller's intersection test [222]

Chapter Seven
3D VISIBILITY GRAPH CONSTRUCTION AND THE SHORTEST PATH

After all of the intersecting obstacles are identified using the algorithms of Chapter 6, their information is stored and utilized to construct a 3D connectivity graph that represents the free space of the path planning problem. As discussed previously, the goal of geometric-based path planning methods is to map the workspace onto a connected graph and subsequently search the graph for the shortest path. This chapter details the process proposed in this research, to construct and search the free space graph. The method is tested on different workspaces to evaluate its computational performance and the results are presented in this chapter.

7.1 3D graph construction

As reviewed in Chapter 4, the 3D visibility graph must have turning points on the edges of the intersecting obstacle(s). The objective of this section is, therefore, to first find an optimal sequence (or sequences) of edges that house the waypoints. Next, the exact optimal locations of the waypoints on the edges need to be determined. This section details algorithms developed to address these two problems starting with the problem of finding the edge sequence(s).

7.1.1 Algorithms to find the edge sequences

Suppose the direct path from the *Start* point to the *Goal* point is blocked by a polyhedron in a 3D cluttered environment. To enable traveling to the *Goal*, there needs to be at least one waypoint to facilitate the detour around the intersecting obstacle. To minimize the path length, the waypoint must be on an edge of this obstacle as proven by

Sharir and Schorr [24]. This waypoint, as a result, breaks the path into two segments: the *Start*-waypoint and the waypoint-*Goal* segments.

Based on this observation, the proposed algorithm for graph construction in this research is also broken into two parts: (1) the first leg identifies which edges should be visited first, after the *Start* point, and outlines how these edges can be found following a geometric-based approach; and (2) addresses how the *Goal* point is approached after the edge housing the first waypoint is reached?

To answer the question of "what is the next traveling edge after the *Start* point?" the notion of the convex hull is revisited. Even though the convex hull created by a waypoint (or the *Start*) and an intersecting obstacle in 3D may not contain all the nodes of the final free space graph, in contrast to the 2D convex hull, it still provides practical information based on which the edge sequence can be extracted. Take Figure 7.1 as an example; the convex hull created between the *Start* and the intersecting obstacle in this figure contains the edges that are connected to the *Start* via triangles on the hull. These edges are (1.0005-1.0006), (1.0005-1.0007), (1.0007-1.0008), and (1.0006-1.0008) where the digits before and after the decimal show the object id and vertex id in that object, respectively.

As shown in Figure 7.1, the convex hull between the *Start* and the intersecting obstacle can be used to extract all the potential edges that could house the first waypoint.

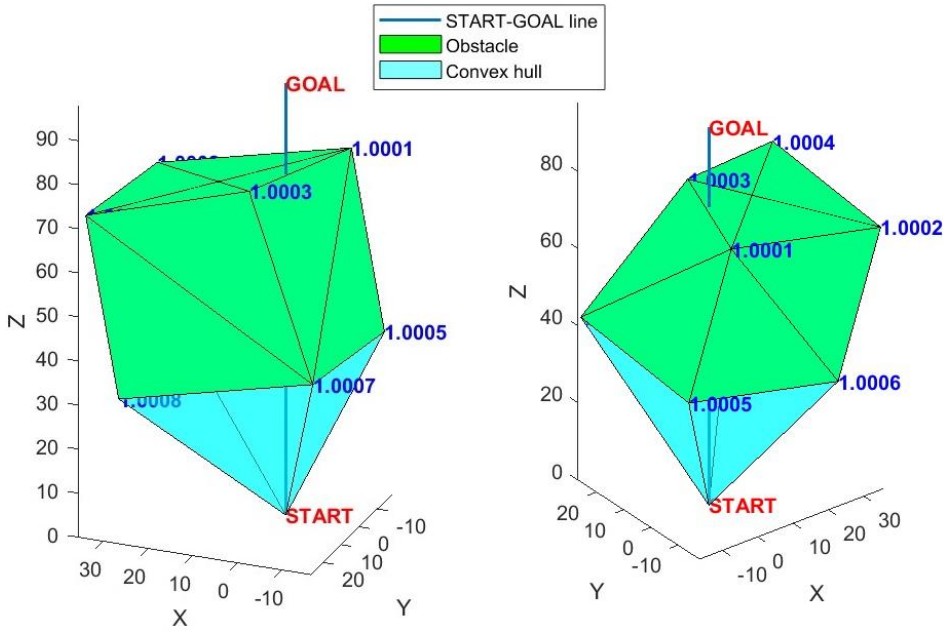


Figure 7.1 3D convex hull with the *Start* and the intersecting obstacle

If the waypoint does not lie on any of these edges of the convex hull, it may either be inside the convex hull or outside its volume. In either case, the location of the waypoint will cause an increase in the path length. Thus, it is concluded that the first point to travel to after the *Start*, must be a point on one of the edges connected to the *Start* on the convex hull. The flowchart of Figure 7.2 follows this rationale to locate the waypoint and subsequently create the first leg of the path in a cluttered 3D environment.

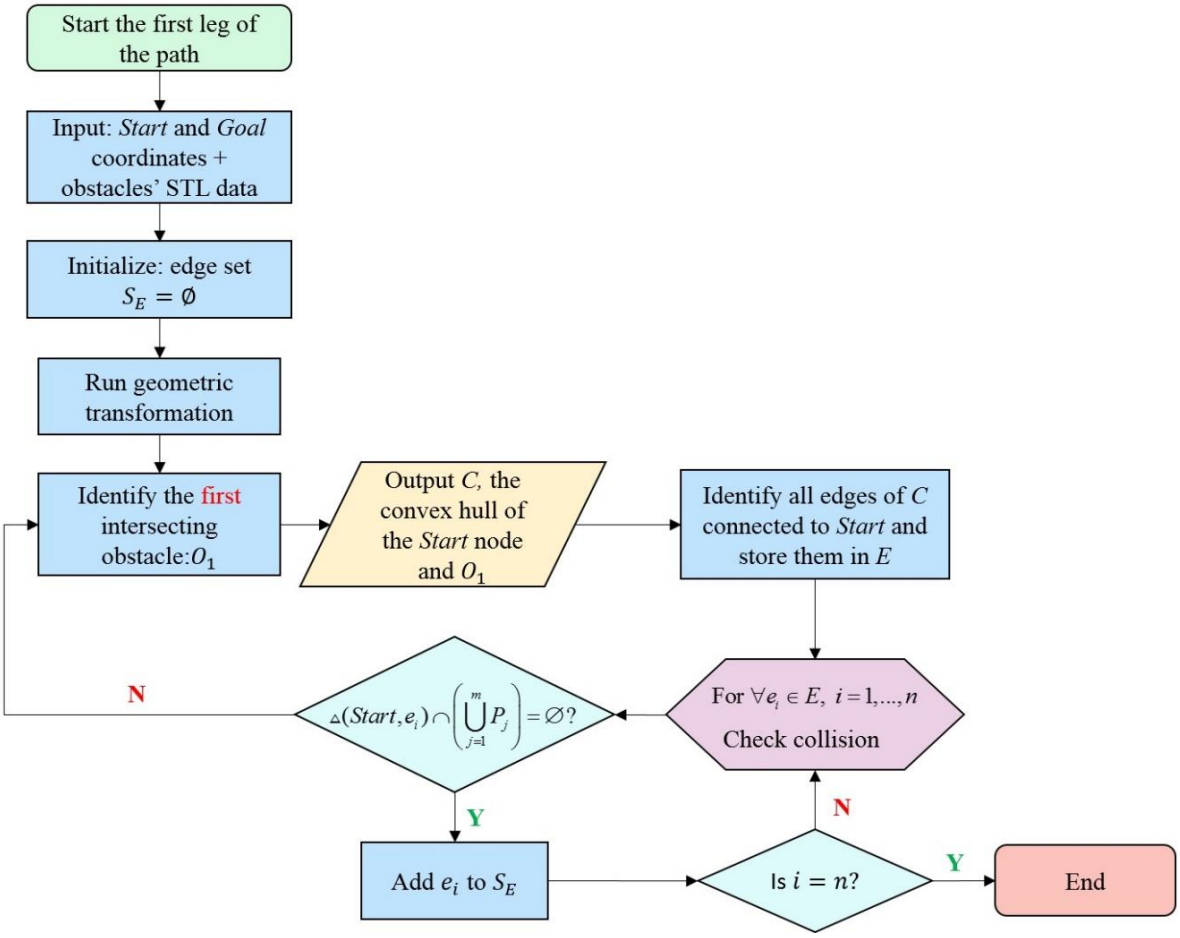


Figure 7.2 Flowchart for determining the first leg of the path

In the flowchart of Figure 7.2, the algorithm starts with taking the cartesian coordinates of the *Start* and *Goal* as well as the STL data of the obstacles. Since the transformation of the *Start-Goal* line and the workspace obstacles is the basis of the intersection detection algorithm discussed in Chapter 6, the first step in finding the set of edges to house the first waypoint is transforming the coordinate system. After the coordinate system is transformed appropriately, all intersecting obstacles can be identified by calling the intersection detection function written in MATLAB[2]. The intersecting

[2] All codes are written in MATLAB and can be accessed from: https://github.com/nmasoud/Routing-algorithms.git

obstacles are then ordered from the closest to the furthest obstacle from the *Start*. This helps to identify the first intersecting obstacle to be bypassed.

After the first intersecting obstacle is identified, its convex hull with the *Start* point is created. This convex hull is then used to extract the edges of the obstacle that may be traversed after the *Start*. Thus, all the edges of the obstacle on this convex hull that are part of a triangle that includes the *Start* (similar to edges shown in Figure 7.1) are found and stored in the set *E*. For all the edges in *E*, If the triangle constructed with the *Start* and an edge from the set *E* is collision-free (which can be checked by calling the triangle-triangle intersection detection procedure discussed in Chapter 6), that obstacle edge is added to the set S_E of the potential obstacle edges to house the first waypoint. Otherwise, the process is iterated by detecting the obstacle nearest to the *Start* that intersects the triangle and creating a new convex hull with the new intersecting obstacle. The process is continued until all the triangles that connect the *Start* to the edges in set *E* on the convex hull of the first intersecting obstacle at each iteration are collision-free. It is worth noting that the triangle between the *Start* and a respective edge is considered as it models the visibility between the *Start* point and all points on that particular edge.

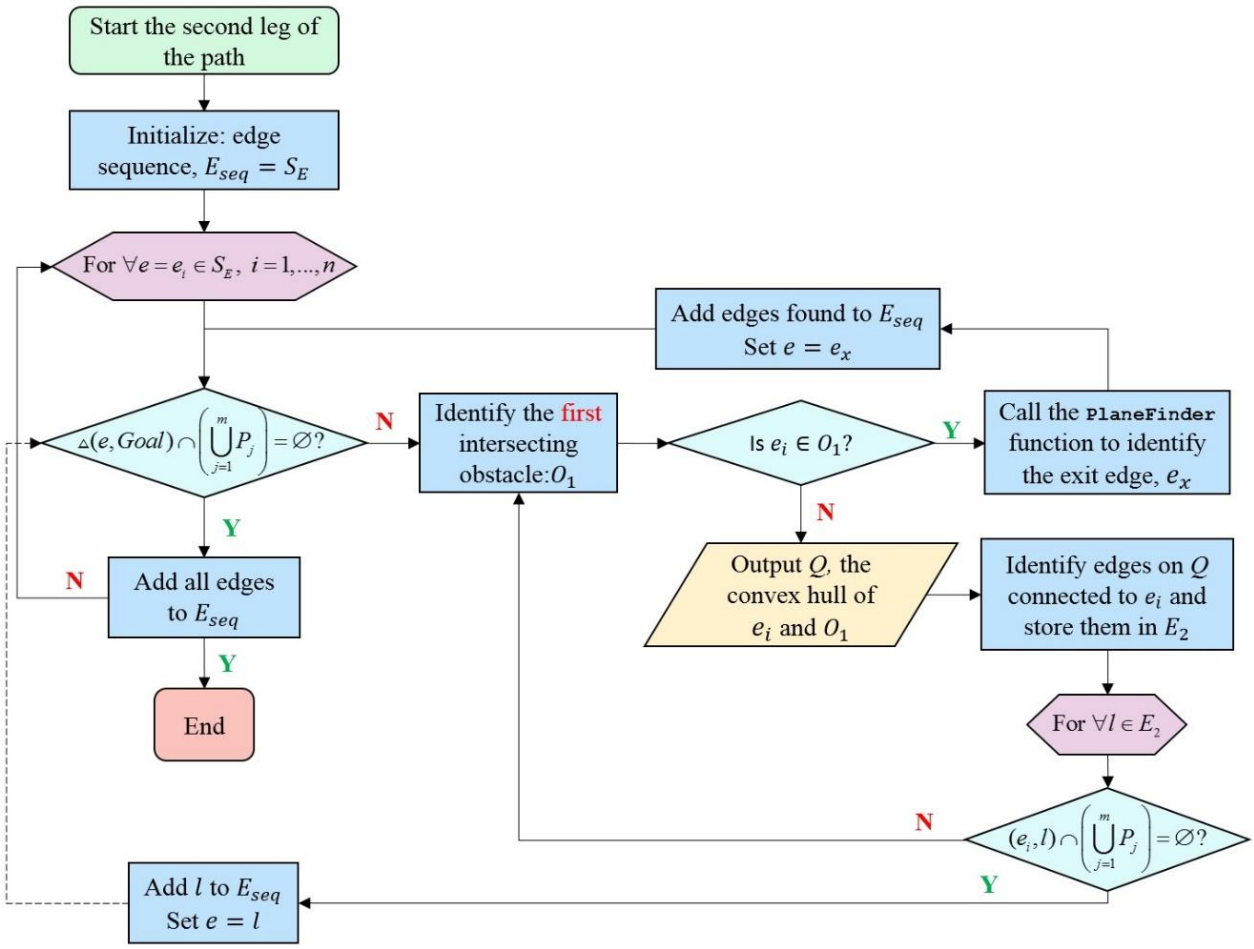


Figure 7.3 Flowchart for determining the second leg of the path

After the edges that could house the first waypoint are found, the second leg of the possible path, the segment between the first waypoint and the *Goal*, needs to be determined. For every edge in the set S_E at least one sequence of edges, following this edge, can be found to connect the *Start* to the *Goal* with a piecewise linear path.

Figure 7.3 lays out the flowchart for the algorithm developed in this research to find the sequences of edges to be traversed to reach the *Goal* from each of the identified first waypoints. Based on this algorithm, for each edge, e_i, in the set S_E, if the triangle created by this edge and the *Goal* is found collision-free (in other words, if the edge is visible to

the *Goal*) the edge sequence initiated by e_i is completed and e_i is the only edge that should be visited on the way from the *Start* to the *Goal*.

If, on the other hand, the triangle intersects with at least one obstacle, other edges must be identified on the way to the *Goal*. Similar to the algorithm for the first leg of the path, the first intersecting obstacle (here, the obstacle closest to the edge e_i) is considered at the initial step. The algorithm identifies the next set of edges to be added to the sequence, based upon two types of intersections between the edge-*Goal* triangle and the obstacle: (1) the triangle including the edge, e_i intersects the obstacle to which e_i belongs and (2) the intersecting obstacle does not contain e_i.

In the first case, a sequence of edges must be found to travel over the surface of the intersecting obstacle to avoid passing through its interior until an exit edge is achieved. The exit edge is the last edge identified on the obstacle from which the goal is visible, and from where the obstacle is left to reach the *Goal*. The "`PlaneFinder`" algorithm is developed that outputs the edge sequences on the surface of the intersecting obstacle.

The `PlaneFinder` algorithm benefits from the convex hull of the intersecting obstacle with the *Start* and *Goal* points to extract the subsequent edges over the surface of the obstacle. Since the convex hull is the smallest convex set that contains all the members of the set, it provides insight on the connected entities (vertices, in the 2D hull, and triangles in the 3D hull). In 2D, the convex hull function of MATLAB outputs a clockwise or counterclockwise ordered set of points that identify the vertices of the hull. Therefore, it is evident which two nodes are the neighbors of a known point. This fact played a crucial role in developing the 2D convex hull based routing algorithm. The 3D convex hull function,

in contrast, outputs a set of triangles that cover the outer surface of the 3D hull (see Figure 7.1). Thus, it may not be as clear as in the 2D hull case, which edges are directly connected to an edge of interest on this surface. To overcome this issue and facilitate finding the edges that follow the first edge on the obstacle, the `PlaneFinder` algorithm adopts the 2D convex hull to benefit from its potential to identify the neighboring entities.

For instance, in Figure 7.1, it is observable that the edges connected to the edge (1.0007-1.0008) on the obstacle are (1.0007-1.0005), (1.0007-1.0003), (1.0007-1.0001), (1.0007-1.0004), (1.0008-1.0004), (1.0008-1.0006), and (1.0008-1.0005) (on the bottom of the obstacle). Figure 7.4 includes other angles of view of the workspace in Figure 7.1 to better visualize the connections of the edges. Suppose the edge (1.0007-1.0008) houses the first waypoint; it can be seen from Figure 7.1 and Figure 7.4 that not all the edges connected to (1.0007-1.0008) are useful in determining the edge sequence over the surface of the convex hull to progress towards the *Goal*. Thus, the less useful edges need to be filtered out from the sequence.

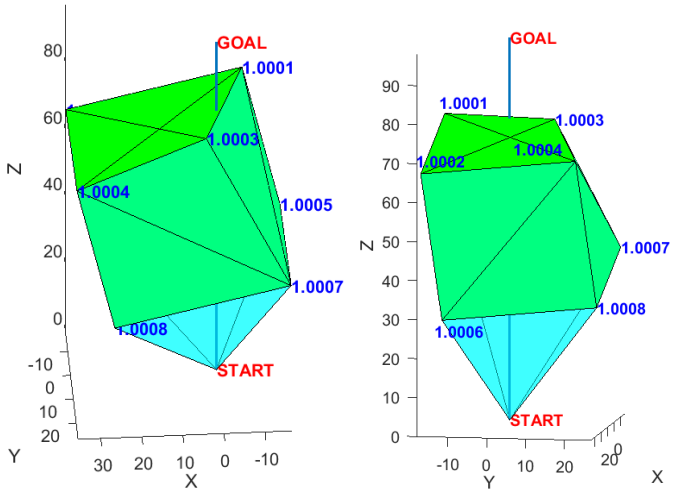


Figure 7.4 Connected edges on the convex hull

To ensure that the unnecessary edges are filtered out, the 3D convex hull is converted to a 2D hull by making a cut through the 3D convex hull generated between the intersecting obstacle and the *Start* and *Goal* points. The cutting plane contains the *Start-Goal* line and the line that is perpendicular to both the *Start-Goal* line and the originally identified edge. To find the perpendicular line, the `PlaneFinder` algorithm identifies the coordinates of two points, one on the *Start-Goal* line and the other on the discussed edge such that the line connecting the two points has the minimum length. This is a simple quadratic optimization problem that can be solved with MATLAB's `fmincon` nonlinear optimization solver. Figure 7.5 shows two views of an example of the line perpendicular to the *Start-Goal* line-segment and the edge (1.0007,1.0008). Shown in this figure, the points P_1 and P_2 are found on segments *Start-Goal* and (1.0007,1.0008), respectively.

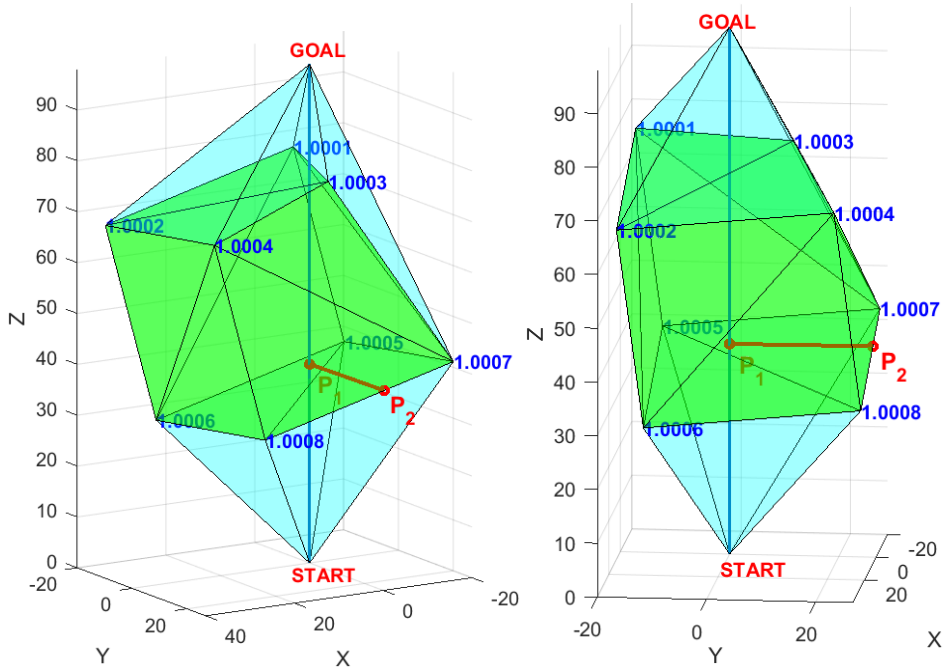


Figure 7.5 Example perpendicular line to the *Start-Goal* line and the edge

After the perpendicular line is determined, the cross product of this line and the *Start-Goal* line is calculated which yields the cutting plane's normal. Now that the normal is found, the plane is fully defined since the coordinates of at least one point in this plane are known (e.g. the *Start* or *Goal*). The final cutting plane for the example discussed in Figure 7.4 and Figure 7.5 is depicted in Figure 7.6.

Although the plane's normal along with the coordinates of one of its points can fully define a plane, since the purpose of the cutting plane is to determine the edges of the obstacle that intersect the plane, the plane's dimensions must be selected such that it spans the entire height, width, and depth of the convex hull. For example, in Figure 7.7, if the plane was only extended up to the (1.0006-1.0008) edge, it could not have covered the edge (1.0002-1.0004); thus, this edge would have not been included in the edge sequence though it should evidently be present in the edge sequence.

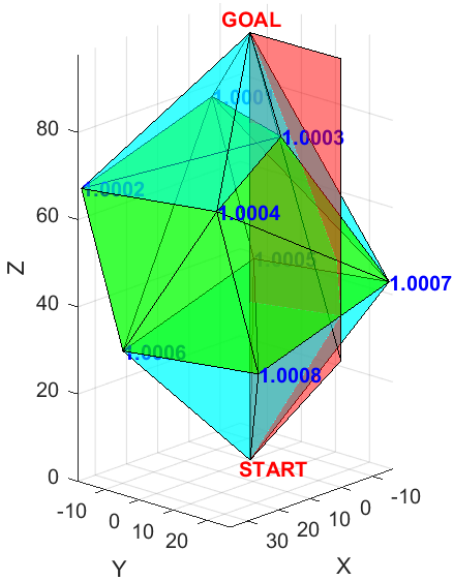


Figure 7.6 Example of cutting plane

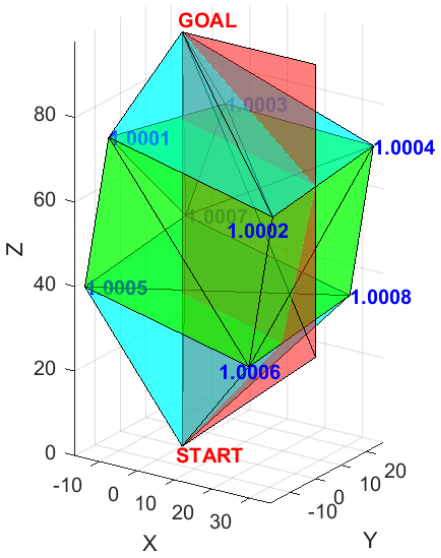


Figure 7.7 Dimensional limits of the cutting plane

As can be seen in Figure 7.6 and Figure 7.7, the cutting planes intersect the obstacle at edges that follow the initial edge with which the plane is created. Figure 7.8 further depicts the intersection of the cutting plane and the convex hull for the example in Figure 7.5 using two angles of view. The plane of intersection in this figure passes from these edges: (1.0007-1.0008), (1.0007-1.0004), and (1.0007-1.0003). Also, it can be observed that this plane connects the *Start* with the *Goal* by traveling to these edges. The example of Figure 7.8 is the edge sequence based on one initial edge and this process needs to be repeated on all other edges found following the algorithm for the path's first leg.

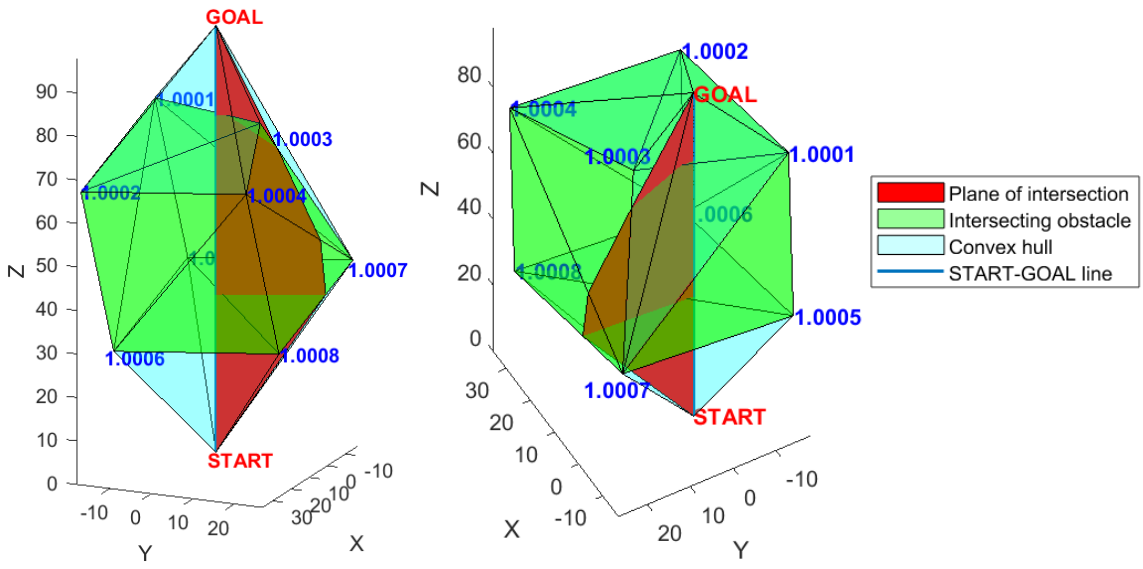


Figure 7.8 The intersection of the cutting plane and the convex hull

This procedure is sufficient to determine the edge sequence for cases with only one obstacle. If, however, the number of obstacles is greater than one, the same approach may not be able to output all possible edge sequences to the *Goal* and additional convex hulls may need to be generated with other intersecting objects.

It should be noted that since the obstacles are disjointed, there is no need to check for intersections with obstacles when moving from one edge on the surface of the same obstacle to another.

After the exit edge is found, the triangle between the exit edge and the *Goal* is checked for collisions with other obstacles. If no collision is reported between this triangle and any of the obstacles, the exit edge becomes the last edge in that sequence and the process goes on to check other edges from the set S_E. Otherwise, the algorithm is recursively iterated to find the collision-free edge sequence (see Figure 7.3) to reach the *Goal*. At this point, the triangle created by the exit edge and the *Goal* may intersect an

obstacle that does not contain the exit edge and thus a different approach must be followed to output the edge sequence.

The second type of intersection between the edge-*Goal* triangle and the first intersecting obstacle, as shown in the flowchart of Figure 7.3, occurs when the initiating edge does not belong to the intersecting obstacle. In this case, a convex hull is generated between the edge, the closest intersecting obstacle to the edge, and the *Goal*, analogous to the first leg of the path, except that instead of having a start point, this time there is a start edge. Hence, similar to the algorithm of the first leg of the path, after the convex hull is created, the edges that are connected to the start edge must be detected. To avoid unexpected (and often undesired) twists in the final optimal path, only edges that form a plane with the start edge are considered to be added to the sequence. See Figure 7.9 for example of the second type intersection and connected edges.

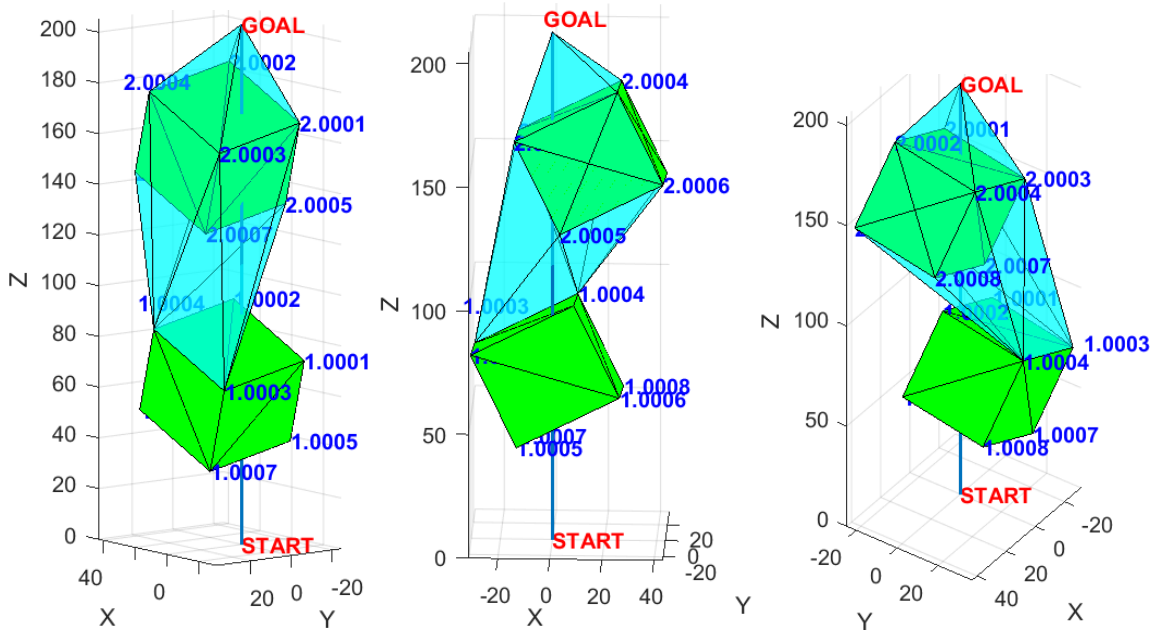


Figure 7.9 Illustration of type II intersection between an edge and an obstacle

The example shown in Figure 7.9 illustrates a type II intersection between the edge (1.0003-1.0004) and the second obstacle. As a result of this intersection, the convex hull is created which indicates the edges connected to (1.0003-1.0004). The connected edges are (2.0003-2.0004) and (2.0005-2.0006). Other edges could be misinterpreted as connected edges such as (2.0001-2.0003) which does not form a planar surface with (1.0003-1.0004) and therefore should not be included in the edge sequence.

After an edge in the convex hull is found connected to the start edge (e.g. (1.0003-1.0004) in Figure 7.9) and forms a plane with it, the planar surface made by the two edges needs to be checked for collisions with obstacles. If no collision is detected, the found edge is added to the sequence and the process moves on to the next connected edge. Otherwise, like the previous case, the algorithm is recursively iterated until the edge connected to the start edge forms a collision-free planar surface with the start edge. For example, both edges (2.0003-2.0004) and (2.0005-2.0006) found connected with (1.0003-1.0004) in Figure 7.9 form collision-free planar surfaces. Thus, they can be added to the edge sequence without further deliberation.

This procedure can be followed at each edge added to the sequence until an edge forms a collision-free triangle with the *Goal* which implies arriving at the final edge of the sequence. Example edge sequences are provided in Figure 7.10 and Figure 7.11 for workspaces of Figure 7.4 and Figure 7.9 respectively. While several sequences of edges are generated for each workspace, only two of them are shown per figure. For example, the two edge sequences in Figure 7.10 are: Sequence I = (1.0005-1.0006), (1.0006-1.0001), and (1.0001-1.0002) and Sequence II = (1.0005-1.0007) and (1.0007-1.0001).

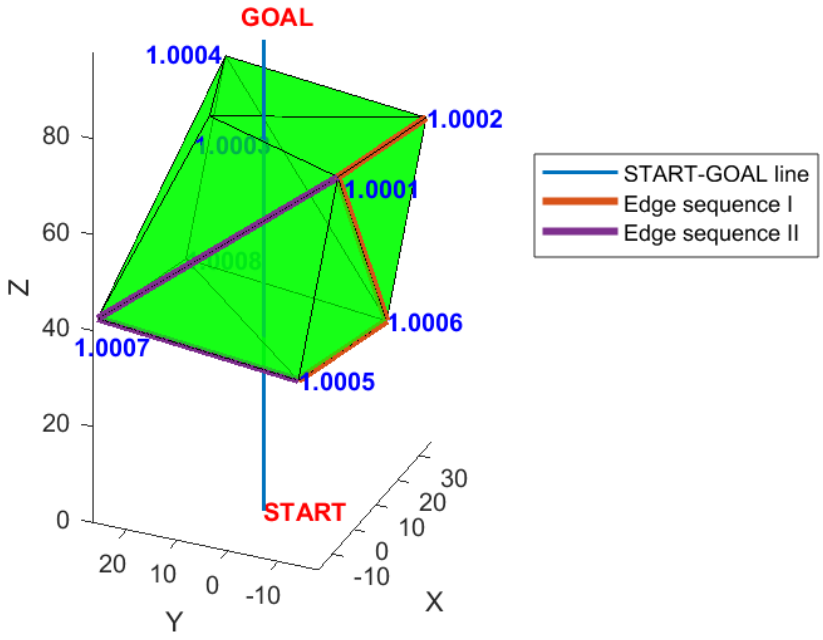


Figure 7.10 Sample edge sequences for Figure 7.4

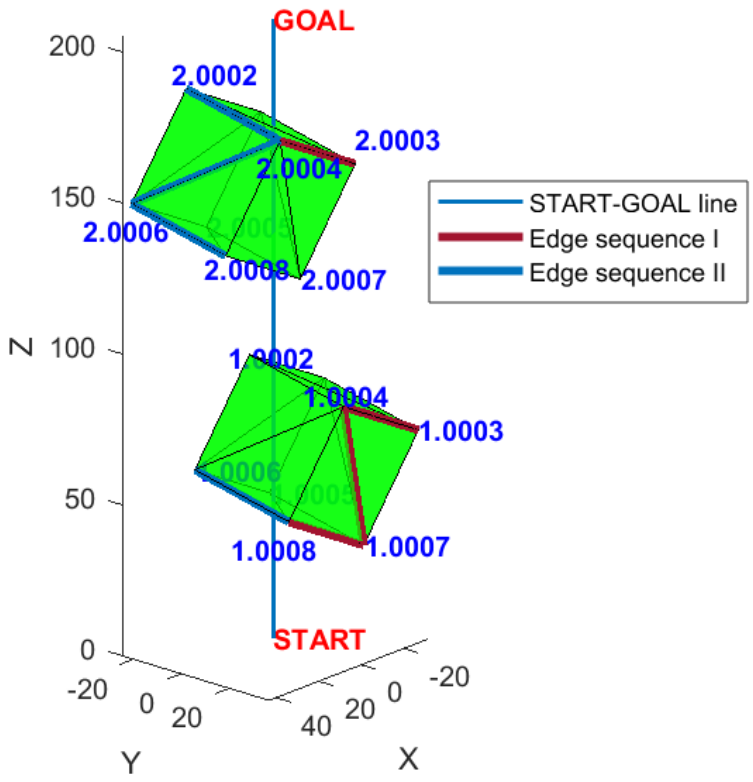


Figure 7.11 Sample edge sequences for Figure 7.9

7.1.2 Optimal locations of the turning points of a path

After all the edge sequences from the *Start* to the *Goal* are found, the exact locations of the paths' turning points must be decided. To find the optimal locations of these waypoints an optimization problem is solved per each edge sequence from the *Start* to the *Goal*. The formulation of this optimization problem is as in **Problem 7.1**.

In **Problem 7.1**, the decision variables are $x_i,\ i = 1, \dots, n$. For each edge in the sequence, an x_i is assigned which has a value between 0 and 1. The parametric definition of a line segment is used in finding the location of the point P_i on the i[th] edge. P_i can be anywhere between E_1^i and E_2^i. For example, if $x_i = 0$, $P_i = E_1^i$ and if $x_i = 1$, $P_i = E_2^i$. The objective function minimizes the total Euclidean distances between the waypoints, *Start,* and *Goal*. Since this is a constrained nonlinear optimization problem, MATLAB's `fmincon` solver is a suitable candidate to solve the problem. Figure 7.12 and Figure 7.13 present the optimal locations of waypoints for the edge sequences shown in Figure 7.10 and Figure 7.11, respectively.

Problem 7.1

$$\min_{x_i \in \mathbb{R}} Z = \|P_1 - S\| + \left(\sum_{i=1}^{n} \|P_{i+1} - P_i\| \right) + \|G - P_n\|$$

$$S.t.\quad 0 \le x_i \le 1$$

$$x_i \in \mathbb{R}$$

Where

$P_i = E_1^i + x_i(E_2^i - E_1^i)$

E_1^i and E_2^i: the first and second endpoints in the i[th] edge

$S, G \in \mathbb{R}^3$: Cartesian coordinates of the *Start* and *Goal* respectively

n : the number of edges in the edge sequence

$\|a - b\|$: Euclidean distance between a and b in $\mathbb{R}^3$

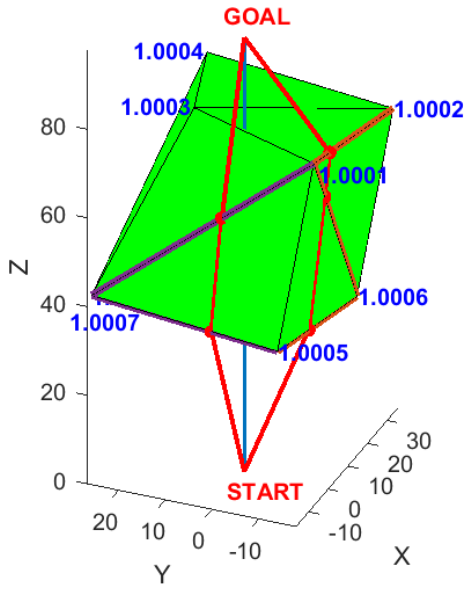


Figure 7.12 Optimal locations of waypoints for edge sequences of Figure 7.10

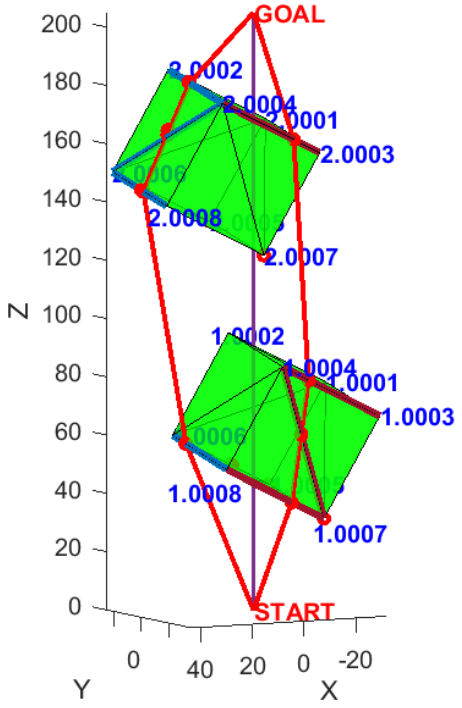


Figure 7.13 Optimal locations of waypoints for edge sequences of Figure 7.11

After all waypoints in an edge sequence are optimally located, the corresponding nodes and edges are added to the graph. The graph is completed by solving **Problem 7.1**

for every edge sequence and appending the generated nodes and edges to it. Final collision-free graphs of Figure 7.4 and Figure 7.9 are shown in Figure 7.14 and Figure 7.15 respectively.

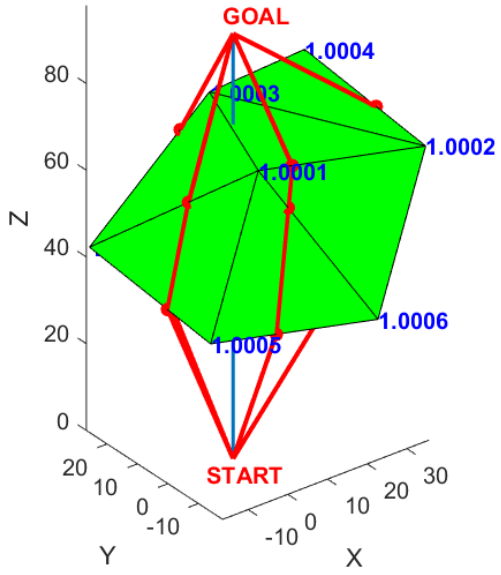


Figure 7.14 Final collision-free graph of Figure 7.4

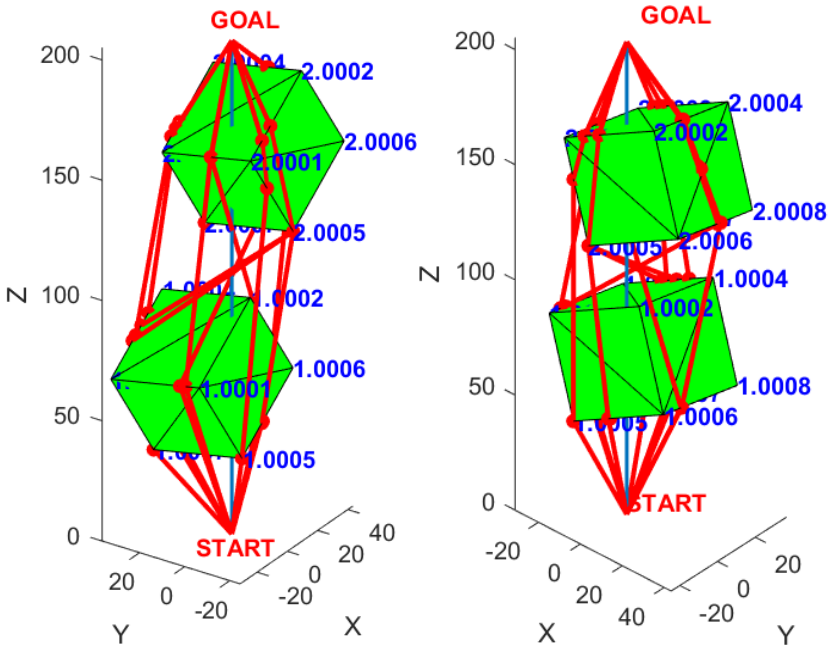


Figure 7.15 Final collision-free graph of Figure 7.9

7.2 Shortest path: 3D graph search

After the free space graph is constructed using the edges of the intersecting obstacles, a search algorithm needs to be applied to find the shortest path on the graph. Various search algorithms exist with different accuracies and time complexities. For simplicity and exactness, Dijkstra's search algorithm [5] is selected in this research. For the graphs of Figure 7.14 and Figure 7.15, the shortest route from the START to the GOAL is found and shown in Figure 7.16 and Figure 7.18 respectively. Additionally, the graphs and shortest paths are shown on the actual untransformed workspaces in Figure 7.17 and Figure 7.19.

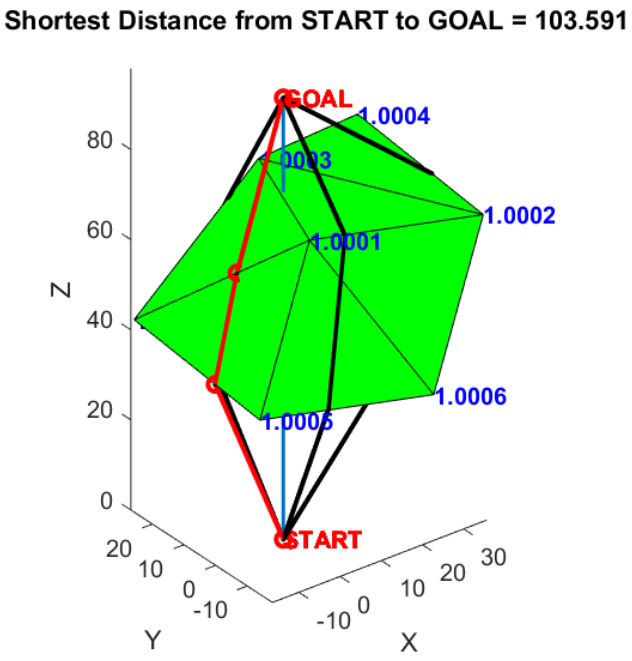


Figure 7.16 Shortest route on the graph of Figure 7.14 (after geometric transformation)

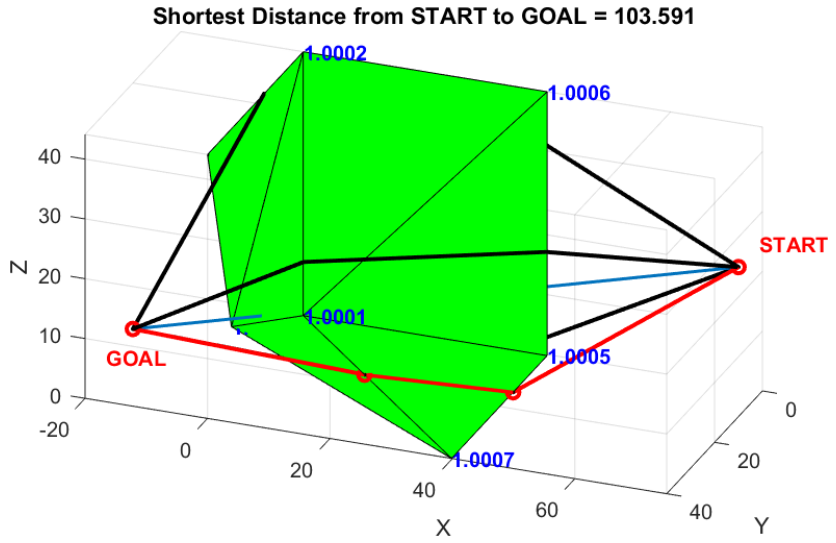


Figure 7.17 Shortest path on the untransformed workspace of Figure 7.4

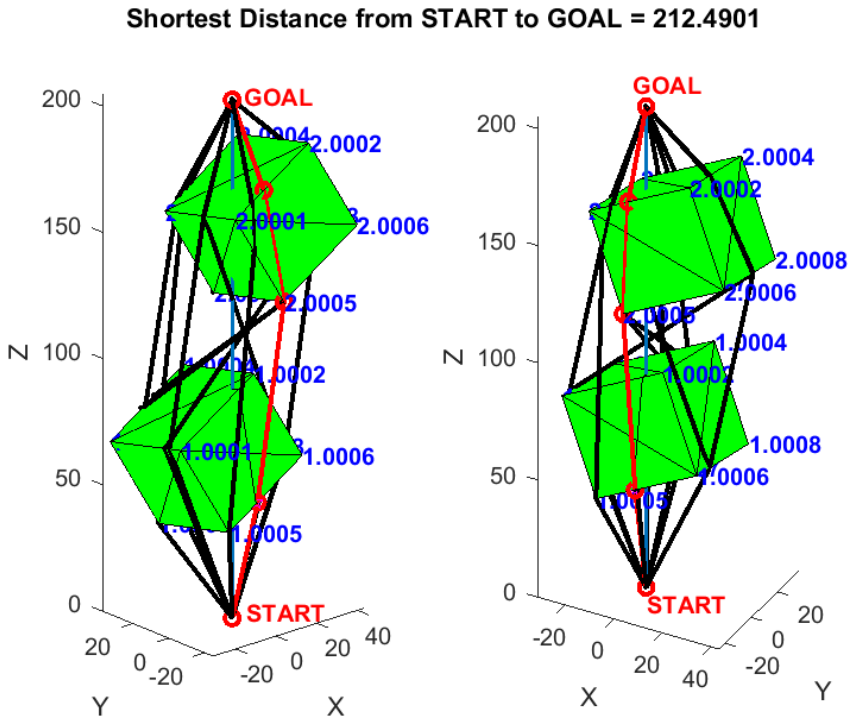


Figure 7.18 Shortest route on the graph of Figure 7.15 (after geometric transformation)

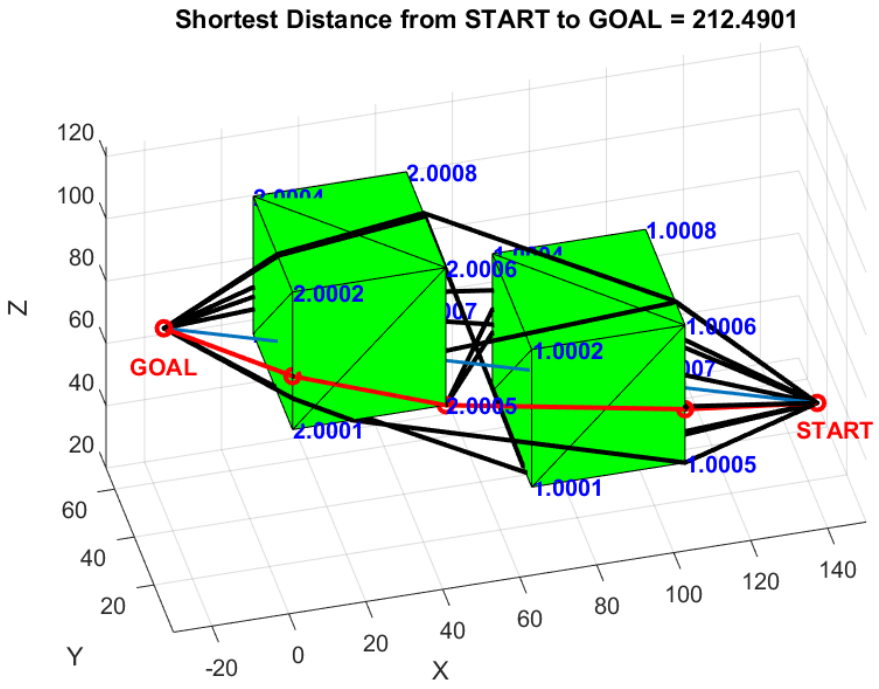


Figure 7.19 Shortest path on the untransformed workspace of Figure 7.9

7.3 Results and discussion

To evaluate the performance of the developed method in constructing the 3D free-space graph and exploring the graph for the shortest path using Dijkstra's method, several test cases are created which investigate the effects of the number of face/edges/vertices of the obstacles as well as the number of obstacles on the final optimal path and the computation time. This section presents the results of these tests followed by a discussion of their meaning from the computational perspective.

7.3.1 Effects of the number of faces/edges/vertices

The presented algorithm of constructing the collision-free graph relies substantially on identifying and manipulating some edges of the obstacles at each iteration. Hence, it can be predicted to observe an increase in the computation time when the number of

geometric primitives (faces/edges/vertices) of an obstacle also increases. In general, it can be predicted that increasing the number of any geometric primitive, defined as faces, edges, and vertices based on B-rep representation [211], will increase the computation time (regardless of the shape of the obstacle) and the path length (only if the obstacle has curved surfaces such as spheres where the curved surface is tessellated to be linearized and resemble a polyhedron). To prove (or disprove) this hypothesis, a test case is generated with one intersecting obstacle with a half-sphere shape. The number of faces in this obstacle is gradually increased (from 10 to 1000) and the final optimal path and the computation time are recorded for each model.

Since it is assumed that all obstacles are convex (free of any non-convexities including holes) and non-self-intersecting and their surfaces are closed, Euler's polyhedron law applies to these obstacles with F faces, E edges, and V vertices.

$$F - E + V = 2 \qquad \text{(Euler's law)}$$

Using Euler's law, it is evident that the number of faces is linearly proportionate to the number of edges as well as the number of vertices for the obstacles used in this research. Thus, increasing one geometric primitive (e.g. the number of faces) will consequently increase the other two (the number of edges and vertices) at the same rate. Hence, it is sufficient to test the effects of increasing one of the geometric primitives and extrapolate conclusions on the effects of the other two. In this study, the number of faces of the obstacle is increased. To do so, the original solid model of the obstacle is imported into Autodesk Meshmixer (software designed to work with triangular meshes) where the number of its triangular faces can be changed and a new solid model is generated. Figure 7.20 shows a

sample of three different solid models generated by Meshmixer with 10, 100, and 1000 triangles, respectively. Also, the *Start* and *Goal* points of the path are shown in this figure with respective locations at (15,15,20) and (10,-5,12).

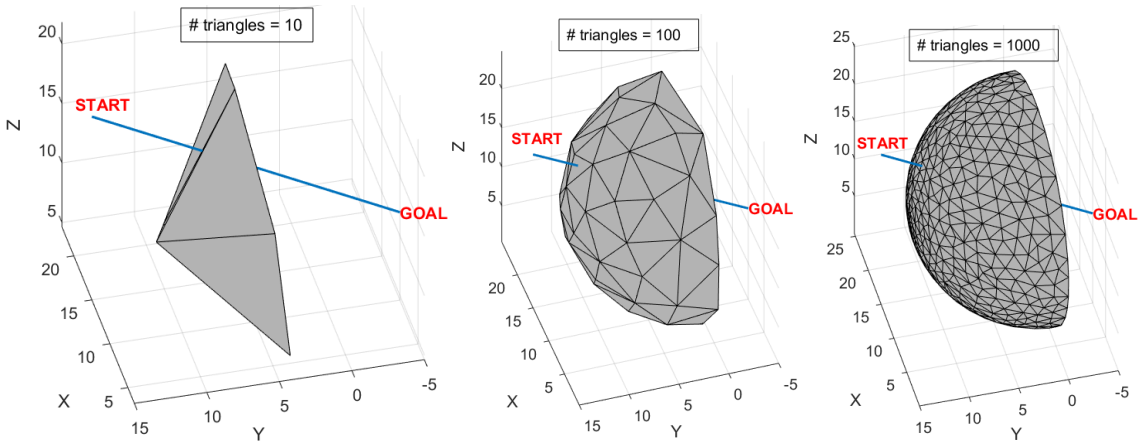


Figure 7.20 Tessellated models of half-sphere used for the test

Additionally, the final collision-free graphs with the shortest path on each of the three models in Figure 7.20 are shown in Figure 7.21.

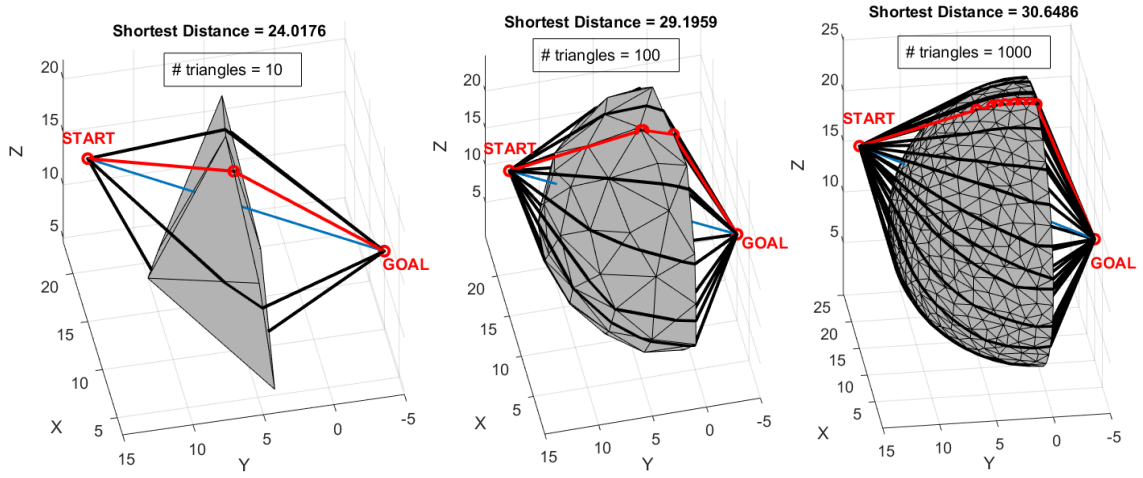


Figure 7.21 Collision-free graphs on tessellated models of half-sphere

The results of the tests for evaluating the effects of the number of faces on the computation time are plotted in graphs of Figure 7.22 and Figure 7.23 in semilog and loglog scales, respectively.

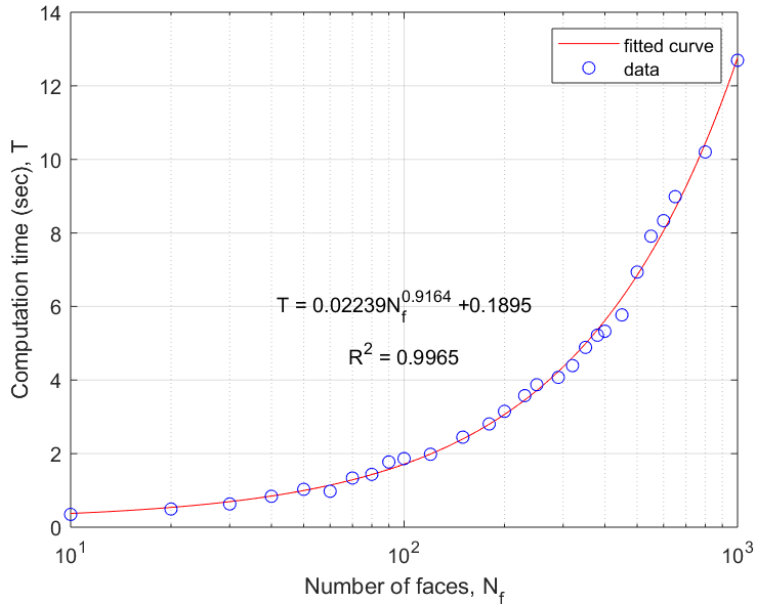


Figure 7.22 Computation time vs. the number of triangular faces (semilog scale)

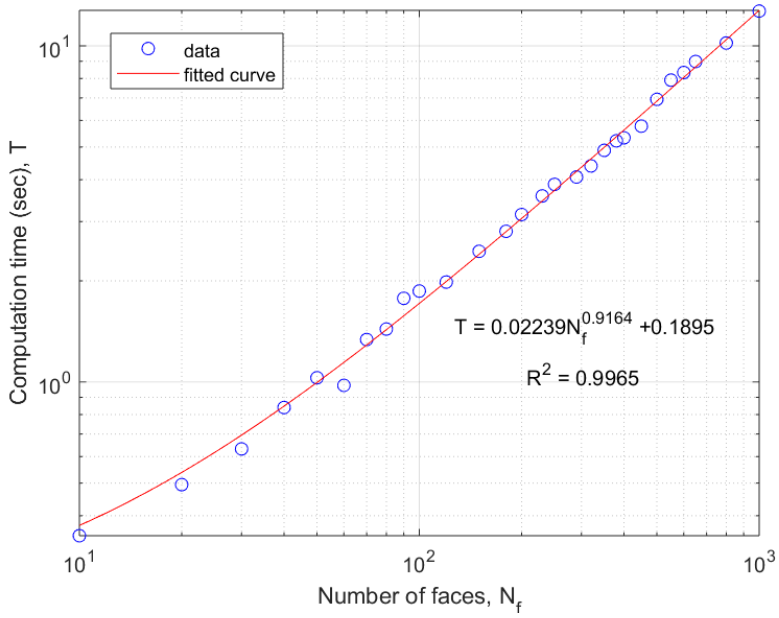


Figure 7.23 Computation time vs. the number of triangular faces (loglog scale)

From Figure 7.23, it can be concluded that for this obstacle, the computation time increases with the number of faces. A power curve fits the data of computation time vs. the number of triangles with $R^2 = 0.9965$.

Further, Figure 7.24 and Figure 7.25 indicate the effects of increasing the number of faces on the length of the final optimal path. From these figures, it is seen that the length of the optimal path increases logarithmically with increasing the number of triangles. The curve flattens at greater than 600 triangles and the optimal length eventually reaches the true length where it becomes independent of the number of triangles. A power curve with $R^2 = 0.9654$ is fitted to the data of path length vs. the number of faces.

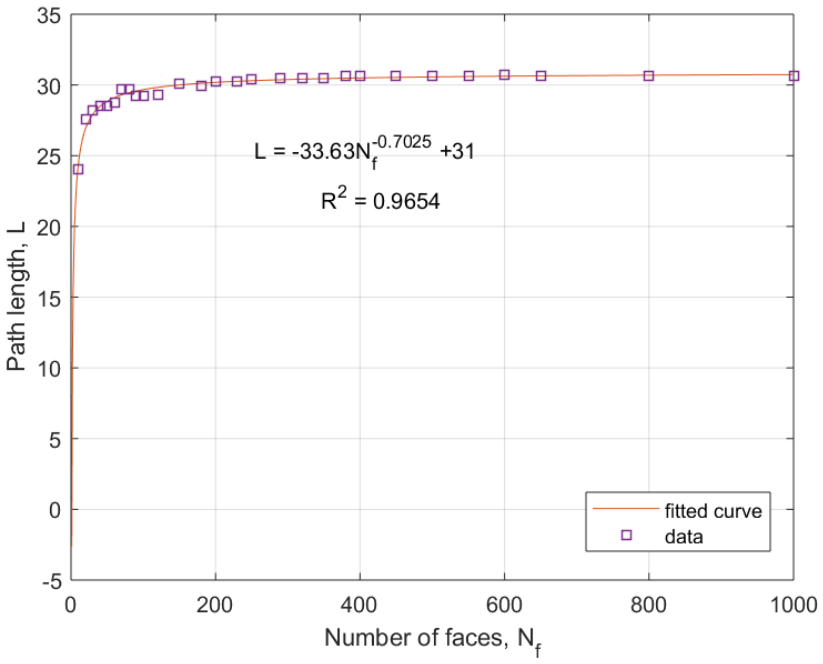


Figure 7.24 Path length vs. the number of triangular faces

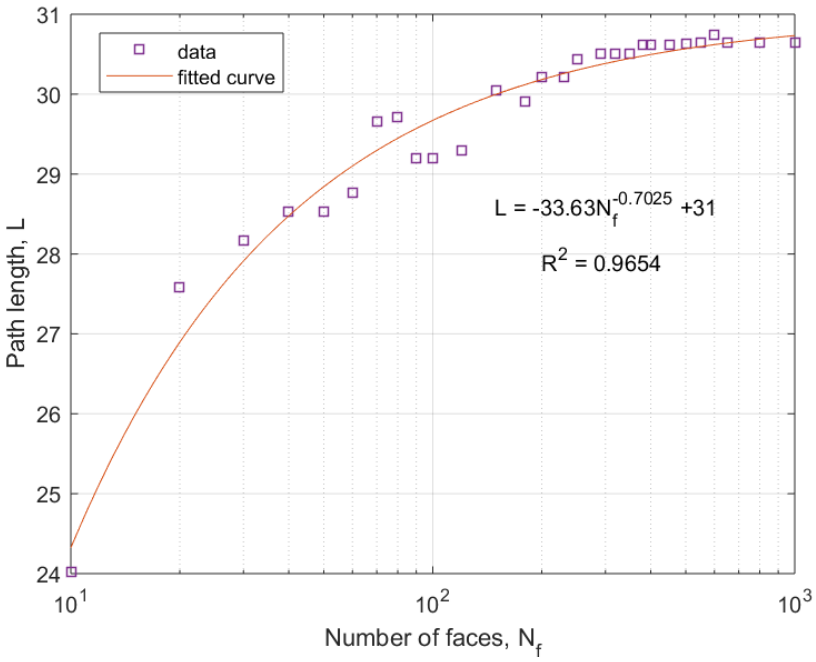


Figure 7.25 Path length vs. the number of triangular faces (semilog scale)

In addition to the examples of Figure 7.20, the half-sphere is oriented as in Figure 7.26 and a different shortest path is obtained by varying the number of triangular faces from 10 to 1000 as shown in the same figure.

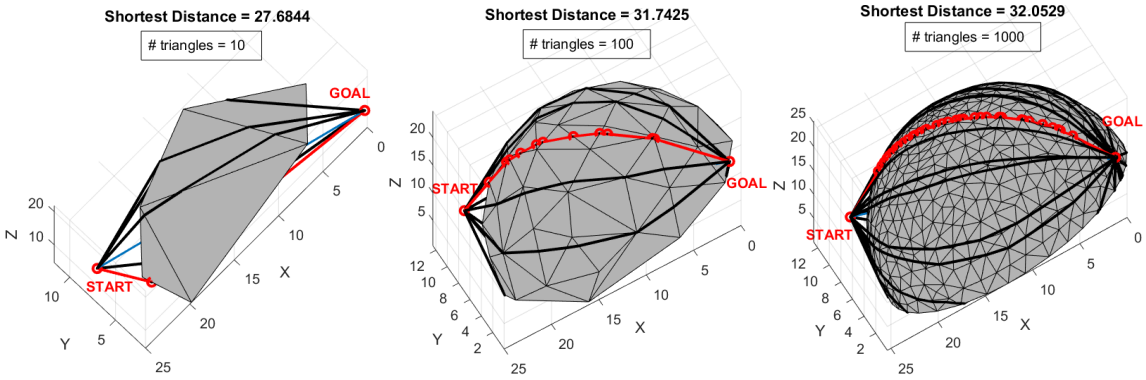


Figure 7.26 Collision-free graphs and shortest paths on oriented half-sphere of Figure 7.20

Similar to the example in Figure 7.20, the computation time and the optimal path length are plotted vs. the number of triangles as in Figure 7.27 through Figure 7.30.

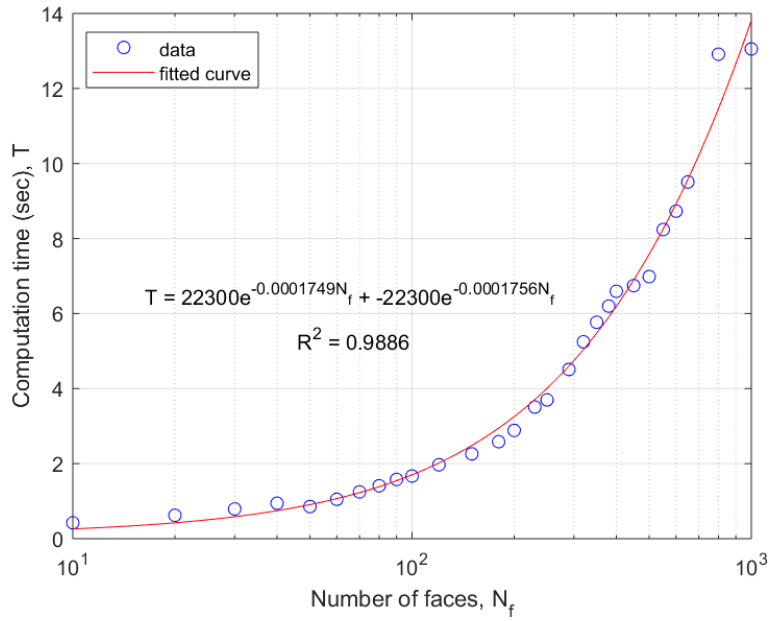


Figure 7.27 Computation time vs. the number of triangular faces for oriented half-sphere (semilog scale)

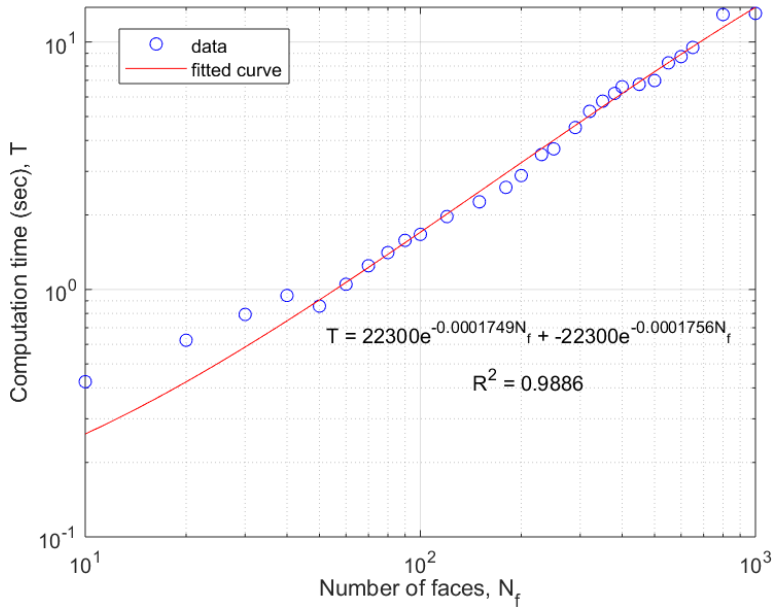


Figure 7.28 Computation time vs. the number of triangular faces for oriented half-sphere (loglog scale)

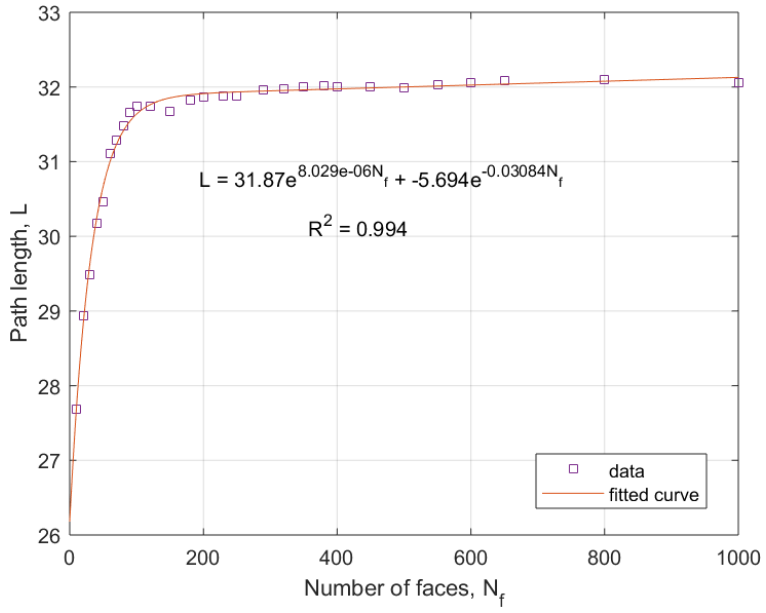


Figure 7.29 Path length vs. the number of triangular faces for oriented half-sphere

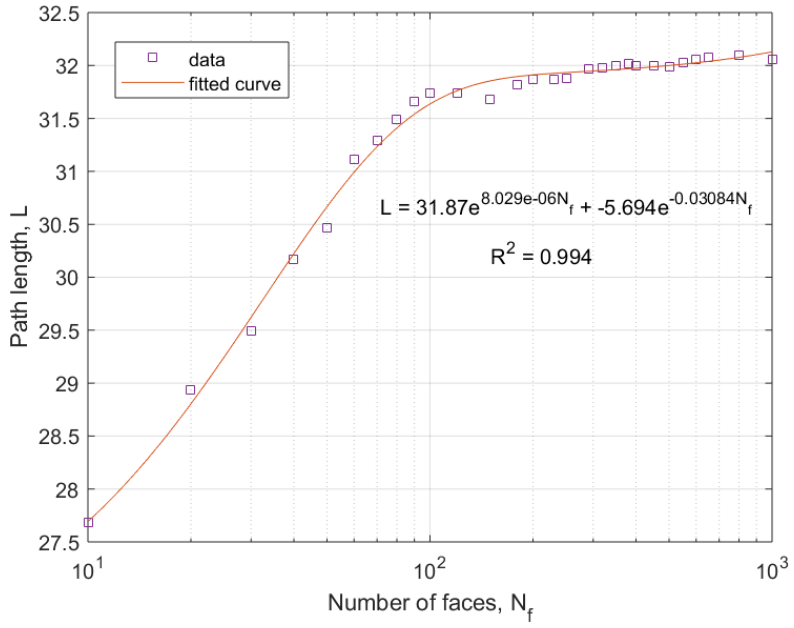


Figure 7.30 Path length vs. the number of triangular faces for oriented half-sphere (semilog)

Exponential functions with $R^2 = 0.9886$ and $R^2 = 0.994$ describe respectively the increase in computation time and path length as the number of triangular faces in the half-

sphere grows as indicated in Figure 7.27 through Figure 7.30. Therefore, with changing the orientation of the half-sphere, the same conclusions can be drawn that the computation time and path length increase when the intersecting object has more faces to pass over.

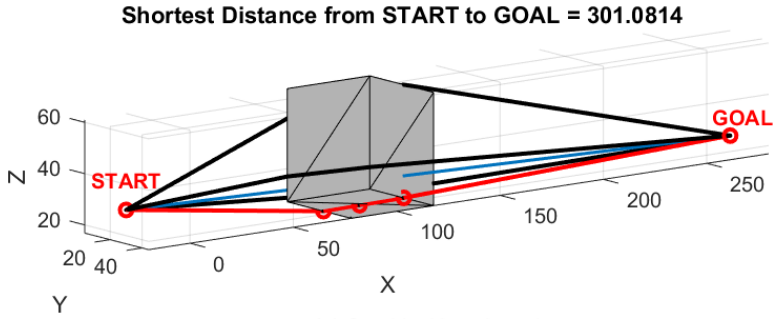
It should be noted that the outliers on the path length vs. the number of faces are resulted from adding the triangles with different sizes and orientations to the original model of the half-sphere with 10 triangles. These new faces can be added with an orientation that blocks the path, therefore increasing the length, or they may not interfere with the found shortest path.

Even though the results of Figure 7.22 to Figure 7.25 and Figure 7.27 to Figure 7.30 indicate that both computation time and path length increase with the number of triangular faces, this may not be extrapolated to the cases with more than one obstacle as the rate of increase could be steeper. One reason is that for the case with only one obstacle, it is sufficient to create one convex hull with the *Start* point and the obstacle. This convex hull alone can yield all possible edge sequences from the *Start* to the *Goal* without the need to create a second convex hull. This, however, is not true for more than one obstacle, as at each iteration, there might be a need to create a new convex hull which could increase the computation time more drastically.

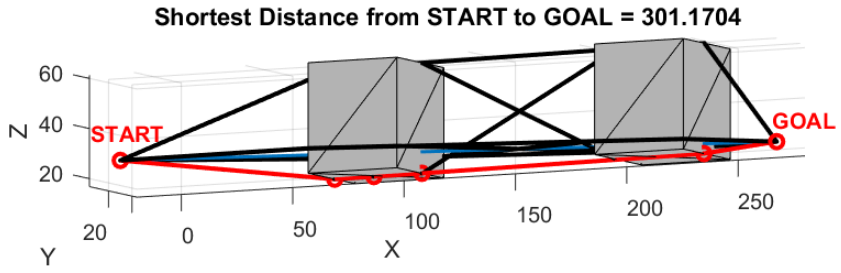
7.3.2 Effects of the number and shapes of the intersecting obstacles

Adding more obstacles that block the path to the *Goal* will lead to generating additional convex hulls to extract the edges on the new intersecting obstacles to be visited and detour the obstacle. Therefore, the addition of intersecting obstacles increases the overall computation time and may also affect the optimal path length.

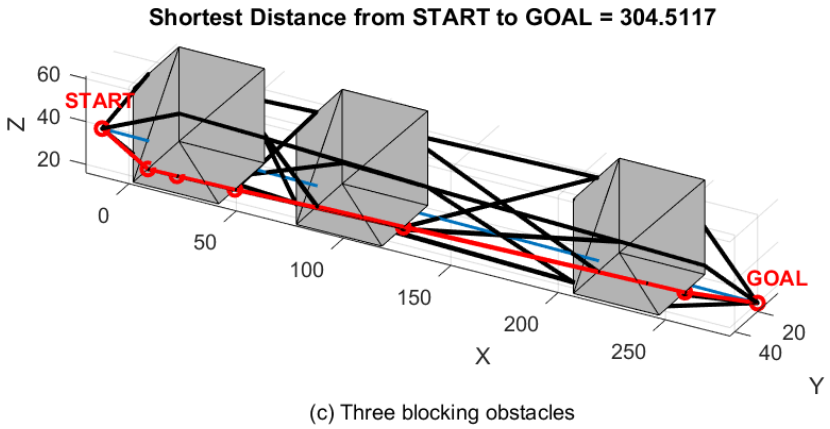
Consider the simple example of a cubic obstacle that blocks the path of the $Start = (-20,30,30)$ to the $Goal = (280,20,20)$ in Figure 7.31. The shortest path is found in 0.6 sec using the discussed method of section 7.2 and the solution is shown in Figure 7.31(a). Now, if a second obstacle with the same shape and orientation is added to the workspace with the fixed location of *Start* and *Goal* to further block the line of sight of the points, the graph and the respective shortest path will be changed as shown in Figure 7.31(b), which is found in 5.8 sec.



(a) One blocking obstacle



(b) Two blocking obstacles



(c) Three blocking obstacles

Figure 7.31 Effects of adding blocking obstacles

Adding a third obstacle with similar geometry (shape and orientation), an increase in the computation time (15.2 sec) and a change in the graph alongside the shortest path are observable as illustrated in Figure 7.31(c). It is also expected to observe the same trend of obtaining a new graph in higher computation time and possibly a new shortest path as more obstacles are added that block the line of sight between the two points.

It should be noted, however, that depending on the location and orientation of newly added obstacles, the shortest path may stay the same while certainly a new graph is generated. For example, if the second object in Figure 7.31(b) is added as in Figure 7.32, such that it does not interfere with the graph edge in the shortest path (shown with the arrow in Figure 7.32) that connects the first object to the *Goal*, the shortest path does not change on the new graph.

If, however, the location of the second obstacle remains unchanged and is the same as in Figure 7.32, but it is rotated around the *Start-Goal* line, a new graph, as well as the shortest path, are achieved as shown in Figure 7.33. It can be seen that the graph is changed and the shortest path is slightly longer than the one found in Figure 7.32. Therefore, both

the location and the orientation of the new blocking obstacle(s) affect the final graph and possibly the shortest path.

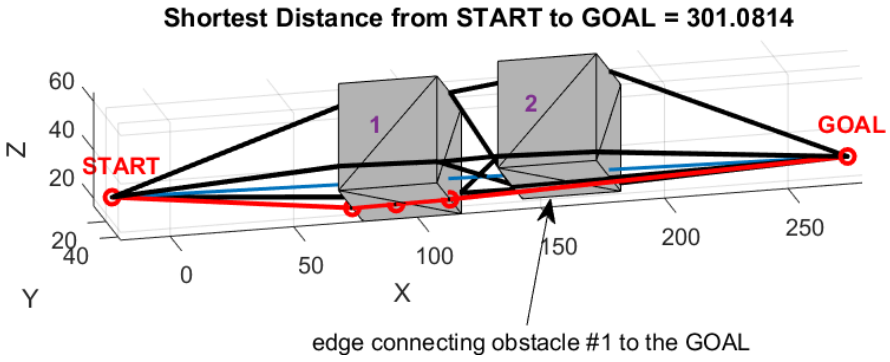


Figure 7.32 Effect of the location of the blocking obstacle

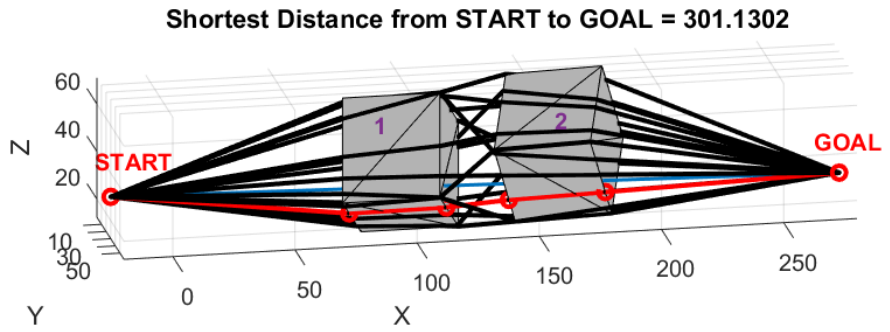


Figure 7.33 Effect of the orientation of the blocking obstacle

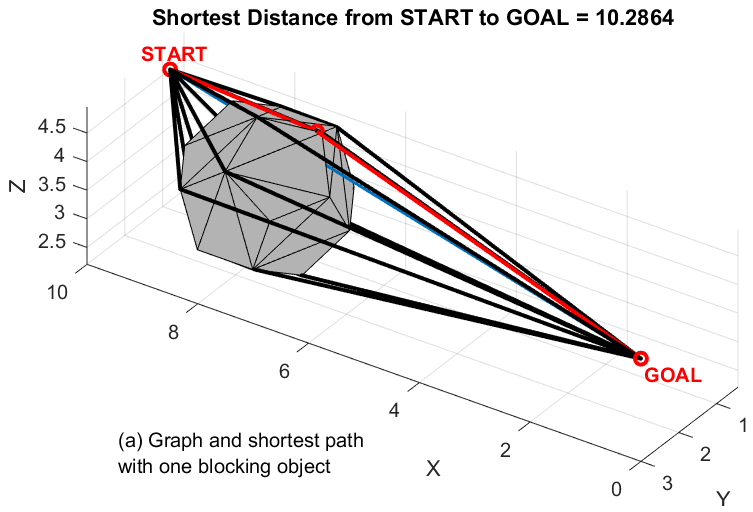
It is noteworthy that in generating the graph of Figure 7.33, the constraint of being co-planar for two consecutive obstacle edges (the second must belong to the next object), illustrated in Figure 7.9, is relaxed to allow having twists in the path. While this may not be desirable for some applications (e.g. pipe routing), it can be possible in the wire routing problem where the connectors are flexible.

From these observations, it could be concluded that adding more blocking obstacles increases the computation time and results in a new graph or even a longer path. Assuming the shapes and orientations of all blocking obstacles are the same if *n* of these obstacles are

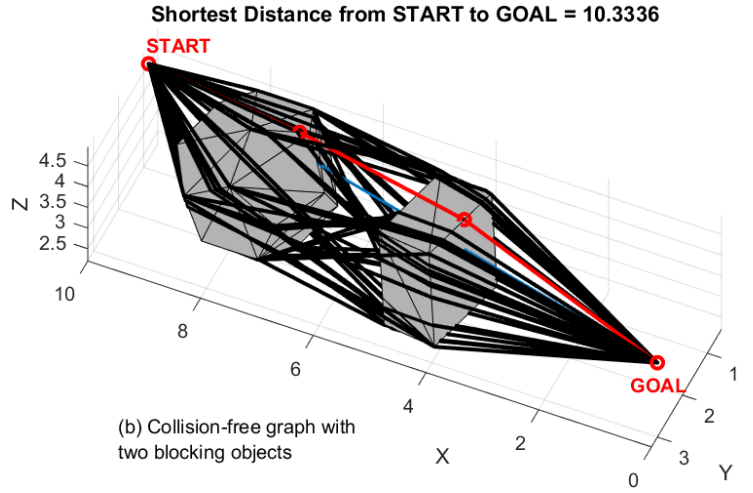
put in series to block the path and each has E number of edges at its largest cross-section (where it touches the convex hull), going from one obstacle to the next, at each iteration, E convex hulls are generated to extract the edges on the next object to be traversed. Thus, in total, $E^n + 1$ convex hulls are generated, one convex hull between the *Start* and the first intersecting object and E^n convex hulls between an edge of an obstacle and the next intersecting obstacle in the line. Therefore, the number of required convex hulls grows exponentially with the number of blocking objects which could directly influence the computation time. This conclusion may not, however, be generalized to cases that involve objects of different shapes/orientations.

In the examples shown so far, the shapes of the obstacles remained the same while other effects (location, orientation, and the number of objects) were discussed. To have a more realistic evaluation of the effect of adding blocking objects, examples of Figure 7.34 can be considered. In Figure 7.34(a), there is only one intersecting obstacle with 36 faces and the final graph and shortest path are found in 6.15 sec.

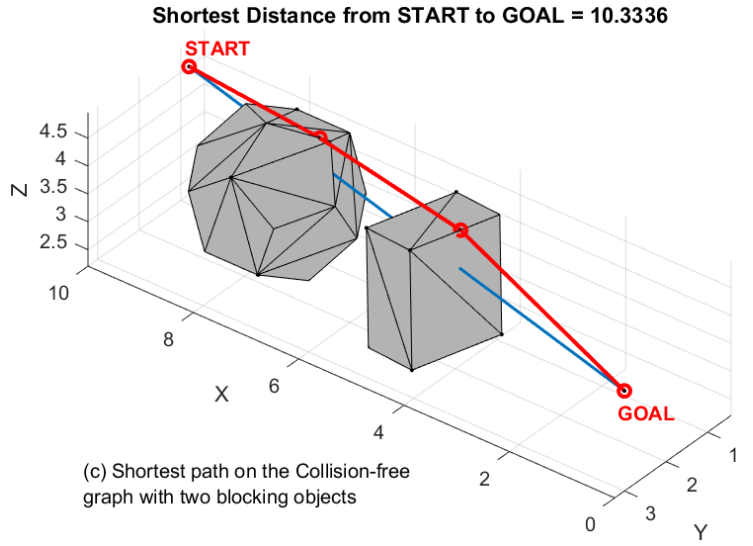
The second object with fewer faces (12), is added as in Figure 7.34(b) and the shortest path is found (Figure 7.34(c)) on the collision-free graph (Figure 7.34 (b)) in 26.06 sec. Finally, the third blocking object with even fewer faces (8) is added which still increases the computation time to 69.20 sec to generate the graph (Figure 7.34(d)) and search it for the shortest path (Figure 7.34(e)). This example shows that the addition of blocking objects to the path increases the computation time to generate the graph regardless of the shape and number of faces of the added object.



(a) Graph and shortest path with one blocking object



(b) Collision-free graph with two blocking objects



(c) Shortest path on the Collision-free graph with two blocking objects

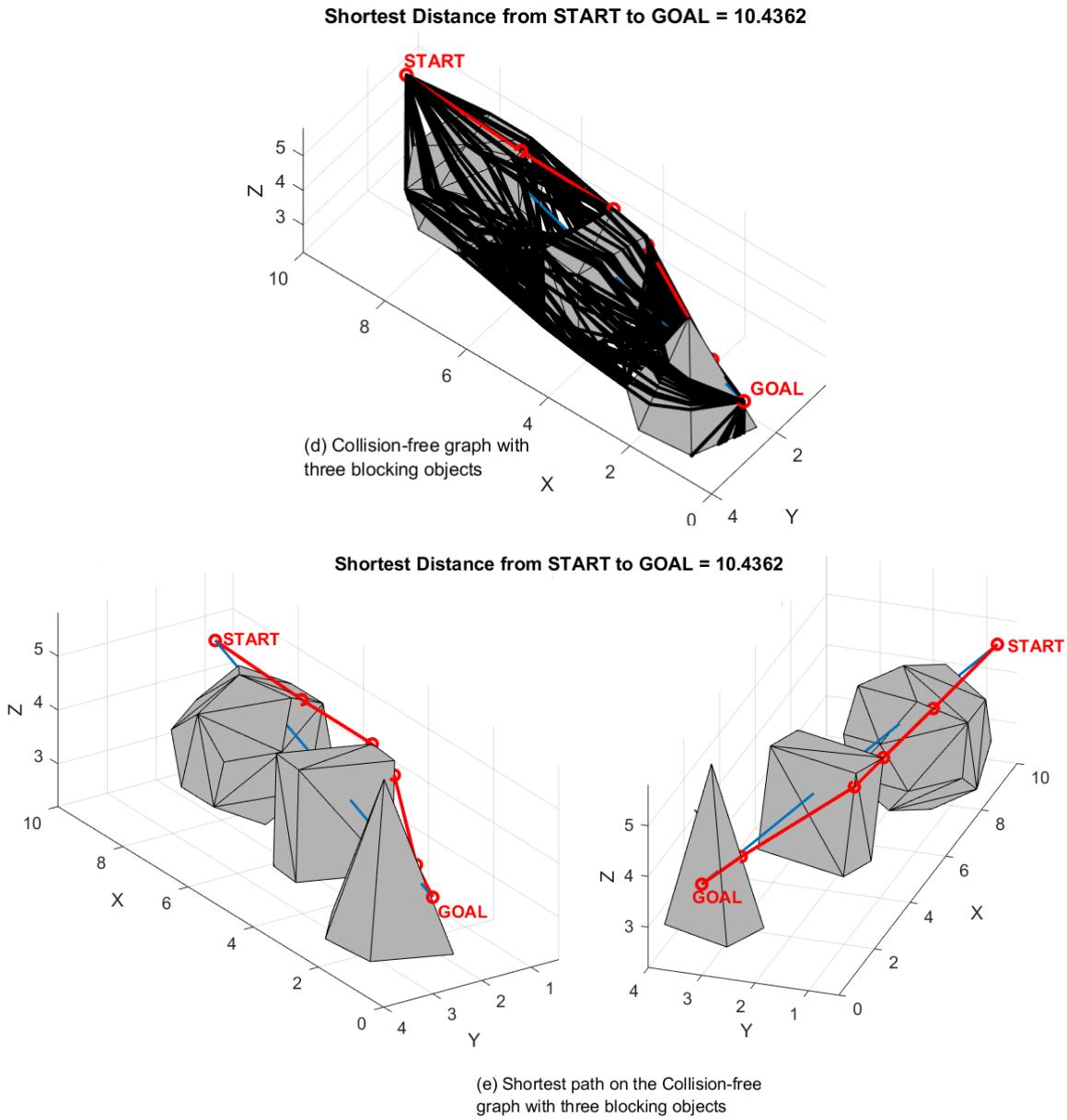


Figure 7.34 Effects of adding blocking objects of different shapes

Figure 7.35 demonstrates the exponential increase in the computation time by adding more blocking objects in Figure 7.34. Even with fewer faces, the addition of a blocking obstacle increases the computation time.

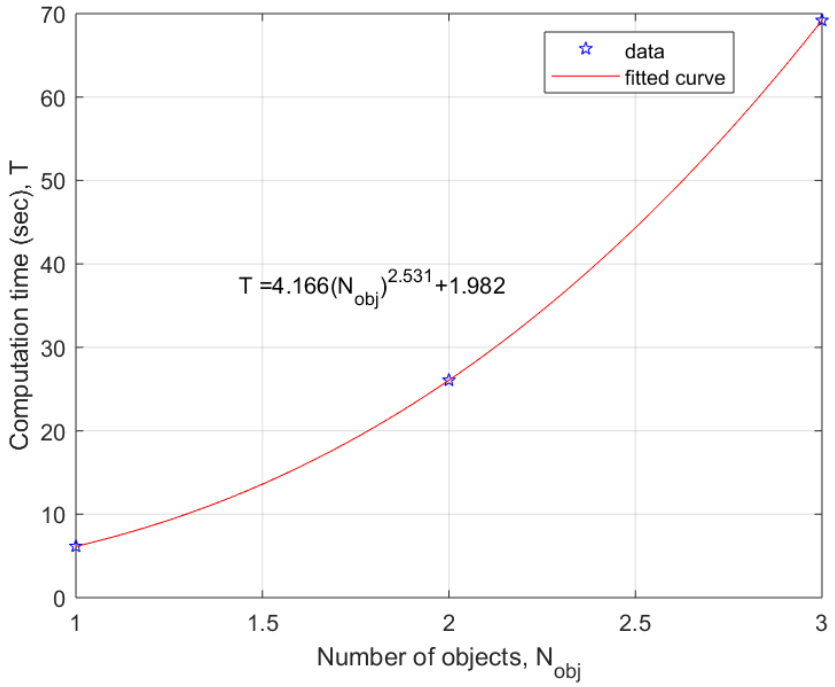


Figure 7.35 Effects of the number of objects on the computation time

While general conclusions cannot be made on the effects of the shape of an object, it is evident that the larger the blocking obstacle is, the more likely it is to be hit and the bigger detour needs to be made; thus a longer path can be anticipated. Figure 7.36 and Figure 7.37 show examples of path planning using the geometric-based method developed in this research on workspaces with different shapes of obstacles. As can be seen from these figures, the method is not limited to vertical objects; it, however, assumes the obstacles are convex polyhedra.

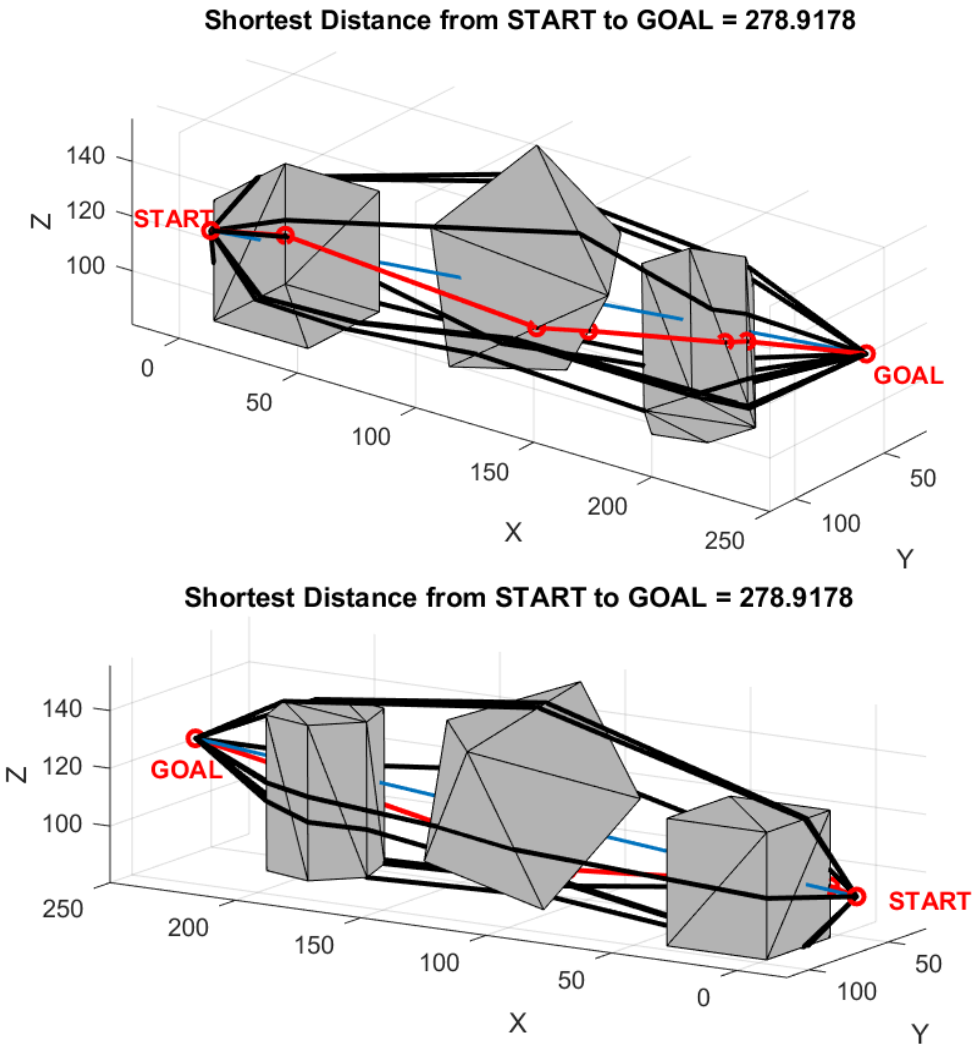


Figure 7.36 Example path planning in a workspace with three blocking objects

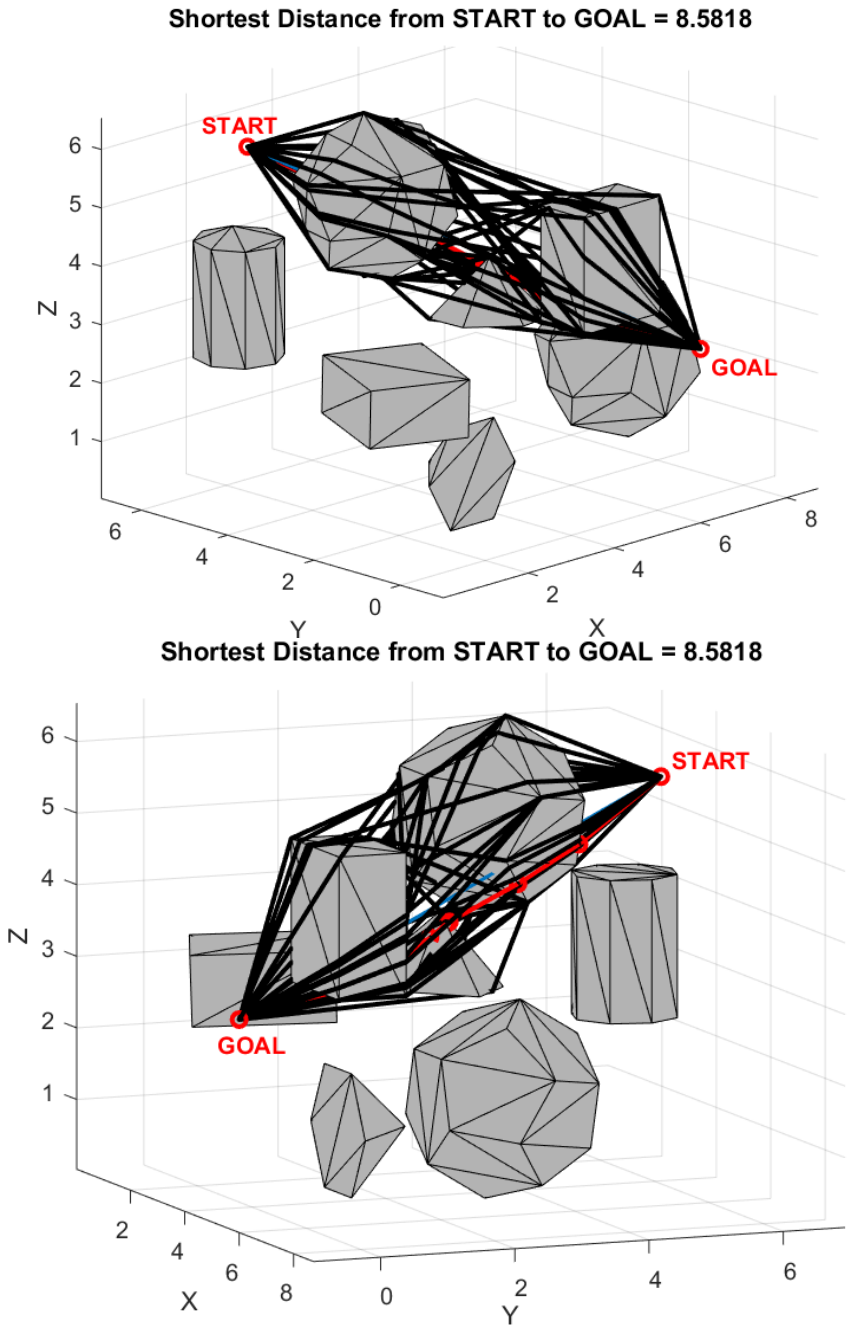


Figure 7.37 Collision-free graph of a workspace with multiple objects of random shapes

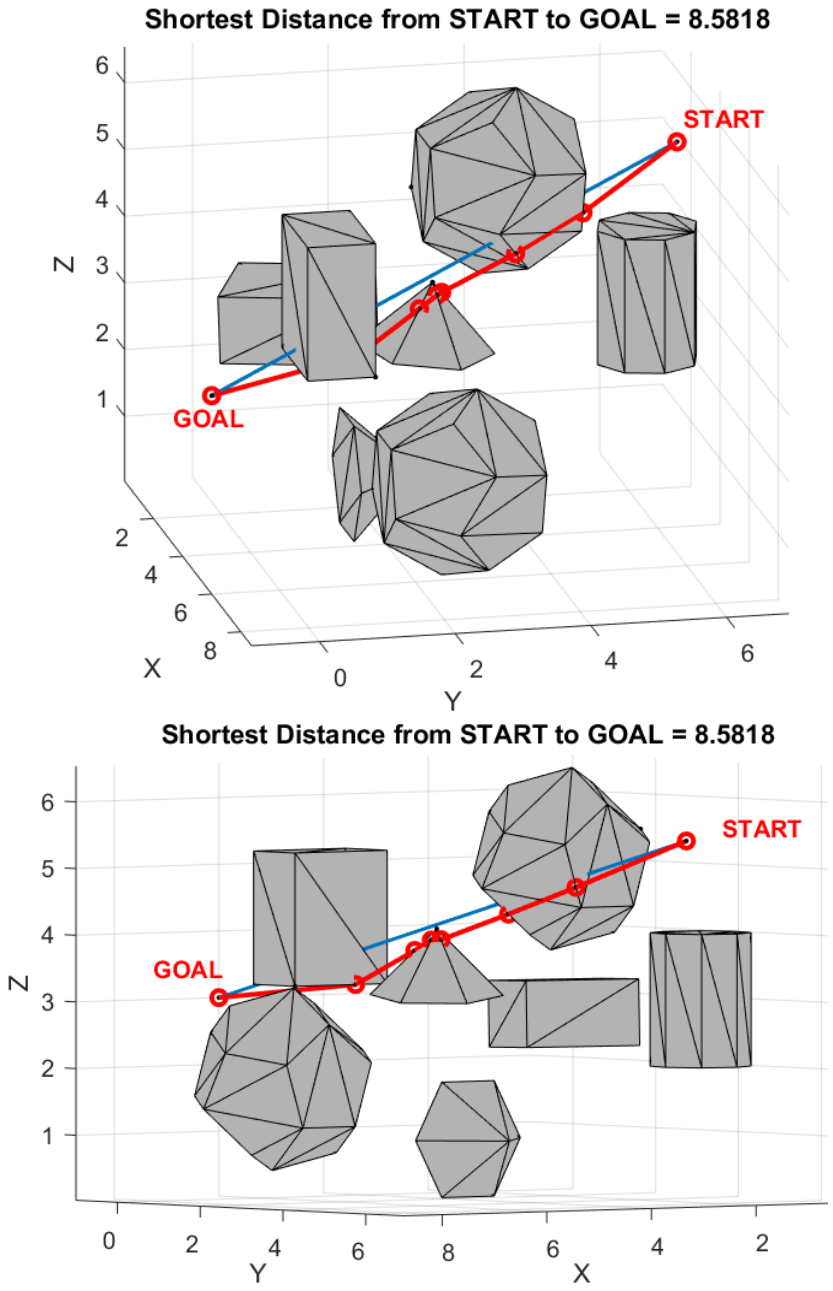


Figure 7.38 Shortest path on the graph of Figure 7.37 Collision-free graph of a workspace with multiple objects of random shapes

7.4 Final remarks

This chapter presented a geometric-based deterministic method for finding the collision-free (visibility) graph between two given points in a 3D cluttered environment.

The presented method makes use of the convex hulls of the blocking objects and the cutting plane method discussed previously to determine a set of obstacle edges to be traversed in a sequence from the *Start* point to the *Goal* point. The found edges are next used in an optimization problem to find the optimal locations of the graph nodes.

An integral assumption in developing the algorithm is that all objects must be convex polyhedra and should be triangulated before being fed to the algorithm. Consequently, if a non-convex object exists in a workspace, two approaches can be taken to tackle the problem: (1) the object can be convexified by using its convex hull (or bounding box) which may introduce some levels of approximation to the final graph or (2) the object can be decomposed into a few smaller convex objects that are in contact with each other and the discussed approach can be used to obtain a collision-free graph. Further, curved surfaces can be approximated with planar surfaces using tessellation-based modeling with the desired resolution.

Unlike the methods developed for UAV routing in urban environments [178,180], there is no need for the edges of the obstacles to be parallel or perpendicular to the axes (like vertical objects) using this method. Also, the method does not subdivide the obstacle edges to locate the visibility nodes in contrast to some other visibility-based methods such as [15,24,30]. Subdivision of edges results in loss of information and limiting the location of the visibility nodes to only a sample of the edge while the entire edge may potentially house the waypoint. While Tran's method [9] also does not use subdivision of obstacle edges to locate the visibility graph nodes, a key assumption in his algorithm is that all

objects are vertical, and therefore cannot handle cases illustrated in Figure 7.36 and Figure 7.37.

The developed method does not use heuristics in determining the graphical representation of the free space, contrary to RRT and PRM, which means if the inputs of the algorithm do not change (geometry and locations of the objects as well as locations of the path endpoints), it outputs the same shortest path.

Last but not the least, even though the capability of the method in finding the shortest path in different cluttered environments with a number of blocking obstacles is shown in sample problems, it is demonstrated with examples that its computation time is substantial and will increase drastically with an increase in the complexity of the problem (e.g. with a higher number of obstacles or more complex shapes (more faces/edges)) demonstrating the NP-Hard aspect of this problem.

Chapter Eight
SUMMARY AND FUTURE WORK

The problems of finding an optimal layout for wire harnesses in two dimensions and defining a graphical representation of the free space in a 3D cluttered environment to enable searching for the shortest path are addressed in this research. This chapter presents a summary of the contributions made in this research and discusses potential research avenues that could be explored in the future.

8.1 Cable harness design problem in 2D

The design of cable harness assemblies requires the planning of optimal (or shortest) routes for the wiring connectors while avoiding collisions with the system components and satisfying physical constraints of the problem including keeping a distance from hot zones and sharp edges.

The final design solutions are often in the form of a graph that captures the topology of the connected system showing where the breakouts are placed and which components are connected to each of them. As discussed in this study, the design of cable harnesses is often left to the detail design stage where the remaining feasible space for the connectors is limited. Therefore, solution methods to address this problem must apply to densely populated environments with freeform objects.

In this work, a classification of the existing approaches to tackle the cable harness design (and similar multipath planning) problems is presented. According to this classification, the efforts belong to either the design category of solutions or the optimization category. While the efforts in the design category are focused on the actual

design process which may overlook the optimality of the solutions and require different levels of human intervention, the optimization methods attempt to provide the optimal solution to the problem. Among the optimization-based approaches, Steiner and spanning trees have gained popularity as they can find a minimum length tree that spans all the nodes of a graph (system components). Additionally, the approaches developed in the location theory are relevant in addressing the optimal locations of the breakouts. While offering exact solutions to the optimization problem, these deterministic methods fall short of addressing the collision avoidance constraint in a cluttered environment. Indeed, location problems in the presence of obstacles have been among the challenging problems in operations research which are considered to be NP-hard [152], and the proposed methods are limited to special cases with convex obstacles [150,151,167].

In this research, two solution methods are proposed and tested to address the problem of " *having a given number of start and goal points that connect different components in a cluttered 2D environment using flexible connectors (e.g. wires), a layout is to be found for the connectors to minimize the total lengths of needed connectors while maximizing their commonality such that the connectors do not cross any objects and the breakouts are not placed inside an occupied area.*" The two objectives of this problem are to first minimize the total lengths of wires and secondly, maximize the common length of wires to provide more accessibility and traceability for maintenance of wiring connections and/or system components.

The first solution method focuses on the mathematical modeling of the problem and uses the visibility information of the *Start* and *Goal* nodes to explicitly define the

optimization objectives in terms of Euclidean distances between the nodes. The idea of a *visibility map* is introduced that subdivides the feasible domain into several regions depending upon the visibility of the points inside a region with respect to the existing nodes. Then, binary variables are used to reflect the decisions on the chosen route to take to reach a node and the chosen region to place a breakout. Examples are shown and solved using the MOGA solver in MATLAB, in the absence of linear equality constraints and using the ε-constraint method when linear equality constraints exist. The set of final non-dominated solutions is generated which may not match the true Pareto set since the solver is heuristic-based and cannot guarantee the attainment of the global solution. The wire lengths around the obstacles are however computed using deterministic approaches and are therefore exact. Despite the capability of this method in explicitly defining the objective functions using Euclidean norms, the complexity of the problem formulation (which indicates the complexity of the solution) highly relies on the problem structure. It is shown, for example, that adding more obstacles, changing the shape of an obstacle, or changing the locations of the existing nodes can drastically increase the nonlinearity of the objectives and/or constraints which has a direct impact on the performance of the solver. Therefore, this method is most efficient for workspaces with few simple obstacles. Further, the obstacles must be polygonal and without any curved edges as having a curvature increases the nonlinearity of the constraints.

For this reason, a second solution method is proposed with the aim of solving the same problems with less computational effort. To achieve this, the convex hull based routing method, proven to be efficient in finding the shortest path in a planar densely

populated environment [83], is deployed to calculate the shortest collision-free distance between the two nodes when they are invisible to each other. The constraints and criteria of the optimization problem are the same and MATLAB's MOGA solver is selected to solve different examples with two breakouts. Final non-dominated sets of solutions are generated. Any member of this set is associated with an optimal layout for the cable harnesses and designers can use this information in their decision making at different stages of design.

The efficiency of the method in dealing with workspaces of different densities is also evaluated. The results show that while increasing the density of an environment certainly increases the computation time and the total lengths of wires, general conclusions cannot be made on its impact on the common length of wires. Other factors that affect the final solution are the number and locations of nodes and the number of breakouts. The two methods are then applied to a sample problem. While both generate the same sets of non-dominated solutions, the computational efficiency of the convex-hull based method is superior.

8.1.1 Future work

In the future, the following research questions can be addressed to further the capabilities of the developed methods to address the cable harness layout design and optimization problem.

(1) Given all the existing nodes, is it possible to develop an algorithm that outputs the visibility map of a cluttered workspace as well as the constraints and criteria of the optimization problem?

(2) How does adding other constraints and criteria (e.g. minimizing the number of turns and the number of breakouts, constraining the bend radii of wires) affect the problem formulation and consequently the choice of the optimization solver?

(3) How can the convex-hull based method be extended to also include rectilinear shortest path? For this purpose, the distance function needs to be modified to use Manhattan instead of Euclidean distance. The outcome of this can be applied to address pipe routing problem instances.

(4) Where in the design process does the wire routing problem need to be considered and addressed? And do the requirements of the problem change depending on the stage of the design process wire harnesses are considered? When the wire harness design is considered in the detail design stage, the feasible domain of the problem becomes limited which reduces the design solutions. Therefore, it may be more reasonable to address this problem at the early conceptual design stage and update the solution at each iteration of the design process to reflect the decisions made and their effects on the optimal layout of the system.

8.2 Graphical representation of the free space in 3D planning

Planning the shortest collision-free path among multiple scattered obstacles is claimed to be an NP-complete problem [223]. Owing to the intrinsic complexities of this problem, researchers mainly resort to heuristic and stochastic methods that can output an acceptable (but not necessarily optimal) solution in reasonable computation time.

The NP-completeness assumption of the problem has not, however, discouraged researchers from developing deterministic solutions that provide approximations to the shortest path. These methods are, in general, *complete* meaning if a solution exists, they can find it, and if no solution exists they stop. Among the deterministic methods, visibility-based approaches are the most popular. Since it might be impossible to generate the complete visibility graph for 3D cluttered environments, approaches are proposed to generate approximate graphs. According to studies [24], the turning points of the path (the nodes of the visibility graph) lie on the edges of the obstacles. These methods, hence, attempt to find possible edge sequences that house the turning points. The efforts were mainly centered around subdividing the edges of the obstacles and using a sampling of the obstacle edges to find the locations of the graph nodes (aka waypoints or turning points) [15,30], therefore, limiting the potential locations of the graph nodes to only a sample of an obstacle edge while the entire edge could potentially house the node. Other methods that do not use sampling on the obstacle edges are limited to special cases, e.g. vertical objects [9,178,180], specific shapes (only handling cubes and cones)[182], or geodesic paths on the surface of one convex polyhedron[24].

To overcome these limitations and focus on the optimality of the solution, a deterministic geometric-based approach is developed in this research that yields possible sequences of obstacle edges to be visited to reach the *Goal*. These edges could be used to form the graphical representation of the free space between the *Start* and the *Goal* which is later searched by a search algorithm (here, Dijkstra), to find the exact shortest path on the graph.

The convex hulls were shown to be effective in determining the graph edges in planar routing problems [83]. Therefore, their applicability was considered in this research. The developed method divides the search for edge sequences into two segments: (1) finding the sequences of edges to reach the first set of waypoints and (2) finding the sequences of edges to reach the *Goal* from each of the first waypoints in the sequence.

Cutting planes are used to find an edge sequence to be traveled on the surface of an obstacle when the waypoint is on an edge that is obscured by the object it belongs to. When traveling from one obstacle to the next is required, the convex hull between the last traveling edge on the previous obstacle is created with the next intersecting obstacle. This convex hull contains the information of edges of the next obstacle that are connected to the edge on the previous obstacle. If the surface connecting the edge on the previous obstacle to an edge on the next obstacle intersects any obstacles in the environment, the colliding obstacle needs to be considered and a new convex hull must be created to find edges on the colliding obstacle that needs to be traveled on before the edges on the next obstacle and the process is iterated until the surface connecting two edges of different obstacles is found collision-free. This is similar to the 2D convex hull based approach where convex hulls are recursively generated to avoid collisions with random objects that were not initially hit.

After all possible sequences of edges are determined, the algorithm solves an optimization problem to find the exact optimal locations of each waypoint on its respective edge. Hence, the method does not require any subdivision of obstacle edges and benefits from the full capacity of an edge to house a waypoint. As a result, final locations of the waypoints are globally optimal.

The capability of the method in deterministically finding shortest collision-free paths in environments cluttered with different objects of convex shapes is shown in examples of Figure 7.36 and Figure 7.37. As seen in these figures, the method is not limited to specific shapes of obstacles (e.g. vertical only or a combination of cubes and cones) as long as the shape is a convex polyhedron.

Since the objects are assumed to be convex disjoint polyhedral, a path always exists between two given points. Also, due to the presence of disjoint obstacles, the cutting plane method can always find an edge sequence on the surface on an obstacle and the convex hull can always generate the connected edges between two obstacles. Thus, it can be concluded that the method is *complete* and can always output a solution.

8.2.1 Limitations of the method

Even though the use of convex hulls in the 2D path planning is proven sufficient in determining all necessary edges of the free space graph [83], the generation of 3D convex hulls may not be sufficient in capturing all the necessary information that could be used to form the graph when non-convex obstacles are involved. For example, in Figure 8.1, a shorter collision-free route is available through the non-convex intersecting obstacle which cannot be obtained using the method developed in this research. Due to using convex hulls to extract the nodes of the collision-free graph, the method can only yield obstacle edges that are on the surfaces of these convex hulls, as seen in Figure 8.1. Therefore, the developed method may not be effective in finding an optimal solution for cases with non-convex objects. As suggested in Chapter 7, in cases where a non-convex object is present, to avoid introducing additional approximations, the non-convex object can be split into

several convex objects that are in contact with one another and the proposed method can be similarly applied to the newly defined problem.

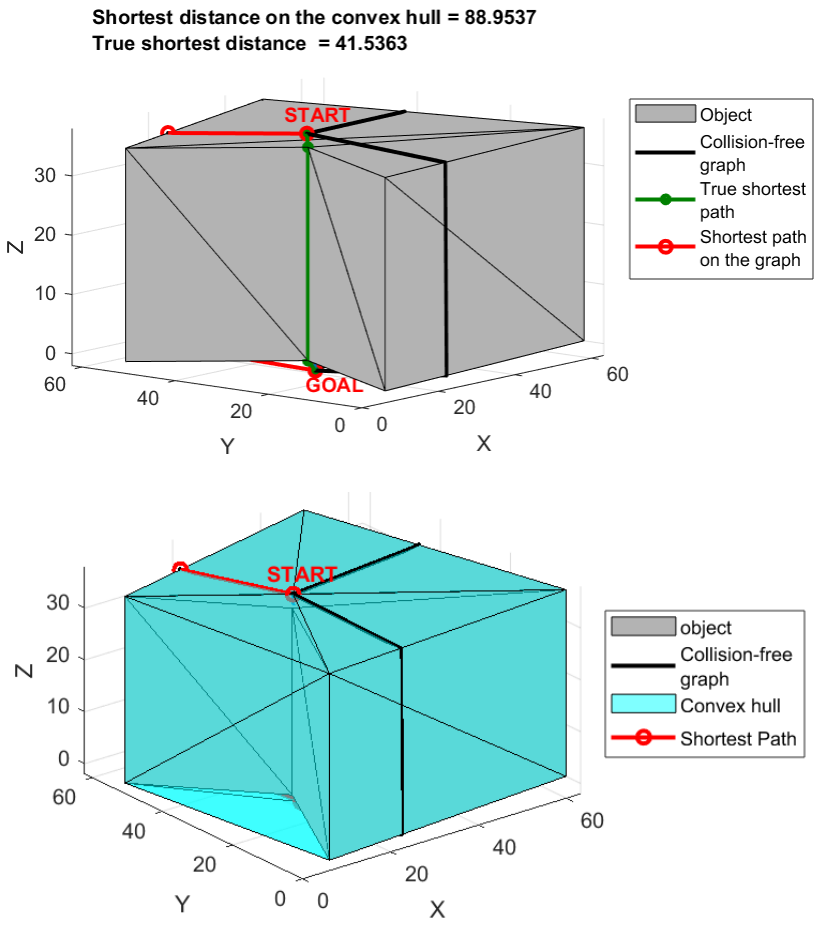


Figure 8.1 Example of path planning in the presence of a non-convex object

Additionally, the objects are modeled using tessellations that linearize curved features. Therefore, the final solution is an approximation to the shortest path. On this account, while STL-based data exchange format is used in this research, any other tessellation based formats can be handled with slight modifications to the program.

Last but not the least, it should be reminded that the deterministic method developed in this research targets an NP-complete problem, the complexity of which grows

exponentially with the increasing complexity of the objects. Thus, it is expected to observe a significant increase in computation time as the problem complexity grows. As an example, the effect of increasing the number of edges of the intersecting obstacle on the computation time is shown in Figure 7.22 and Figure 7.23 which verify the growing complexity of the problem and its characterization as an NP-Hard problem.

8.2.2 2D vs. 3D convex hull based routing

In this study, two graph construction methods are presented for 2D and 3D routing problems. Despite the use of the convex hull geometric structure in both approaches, the two solution methods have fundamental differences in the ways they generate the collision-free graphs. For 2D problems, it is proven that the use of the convex hull created by the intersecting object(s) is necessary and sufficient in extracting the graph edges [83]. This is due to the fact that non-convex vertices need to be excluded from the graph and therefore the corresponding edges connected to such vertices must also be removed from the graph to avoid lengthening the path. This property makes the approach equally applicable to convex and non-convex objects whereas the 3D convex hull falls short of this property. The exclusion of non-convexities in 3D objects (e.g. Figure 8.1) results in overlooking the optimality of the solution found on the constructed graph. Thus, it may not be sufficient to extract the edges of the graph from the convex hulls generated with non-convex objects unless the object is decomposed into multiple convex shapes.

Moreover, at each step in the 2D routing method when a new convex hull is generated, exactly two extreme points are identified as the next traveling (turning) points of the path. This, however, is not the case for 3D routing. In the 3D routing method, when

a new convex hull is generated, the number of next possible traveling points (or edges) varies depending on the number of edges in the cross-section of the convex hull.

In addition to the extreme points, when an intersection exists between a line segment connecting an extreme point and the *Goal* in 2D routing, the information of the convex hull can be used to identify the set of vertices to be traversed on the perimeter of the polygonal object to avoid intersecting its interior. This information, however, may not be helpful in determining the set of edges to be traversed over the surface of a polyhedral object when the triangle generated by an edge and the *Goal* intersects the interior of this object. Hence, in the 3D routing approach, the cutting plane is used as a guide to determine the necessary set of edges to be visited to avoid such an intersection.

8.2.3 Future work

The following research questions can be addressed as potential extensions of this research on the graphical representation of a cluttered workspace.

(1) How can the method be modified to take non-convex shapes of the obstacles and provide the shortest paths? Two approaches are proposed in this study that are convexification and decomposition of the non-convex shapes. The effects of these operations can be evaluated on the final optimal solution.

(2) How can the method be adapted to the changes in the environment? The changes can be on the shapes (configurable objects), locations, and orientations of the obstacles. If the method can be adapted, it can solve real-time routing, routing in dynamic environments, or routing under uncertainty. A possible approach to undertake this problem is to model the changes in the environment

as uncertainties or parameters whose values are to be determined. Following this approach, stochastic or parametric optimization methods can be utilized to solve dynamic routing problems.

(3) Can the method of generating the 3D graphical representation be adopted in solving the 3D layout problem for cable harnesses? The capability of the 3D convex hull based routing method to address planning in the presence of a few convex objects is demonstrated. This approach can be adapted to the mathematical framework proposed for harness layout problems to yield optimal layouts for cable harness assemblies in 3D spaces.

(4) What modifications can be made to the algorithm to be able to generate rectilinear shortest paths? A start point to tackle this problem could be the use of bounding boxed instead of convex hulls and Manhattan instead of Euclidean distances between points in 3D.

(5) The 3D routing problem considered in this research is at the component level. In the actual design process, which is a multidisciplinary problem, the system components interact and affect the system-level and component-level decisions. For example, considering the wire routing problem in airplanes, the routing may not be addressed independently of the constraints and criteria of other disciplines such as propulsion and aerodynamic disciplines. Thus, the effects of and interrelationships among other disciplines need to be factored in when planning the optimal routes for wires. This research proposed a solution to finding an optimal graphical representation that captures the connectivity

information of a system at the component level. One research question toward addressing the discussed challenge is: can the results of this research be used to capture the interrelationships among different disciplines within an interconnected system? And if so, what information can be used, depending on the system under the study, to model the interconnections among different disciplines?

(6) The graph-based optimization method developed in this research aimed at solving the routing problem in 3D. What other problems in interconnected systems can be modeled and solved using a graphical representation of the environment?

8.3 Broader Impact

This research proposed geometric-based optimization methods for routing and laying out cable harnesses in cluttered environments. The mathematical framework proposed for the optimization of cable harness layouts is applicable to network optimization problems where multiple components of the network need to be connected using one-dimensional connectors (e.g. wires, cables, hoses, or pipes). Examples of such networks are electromechanical systems such as computing devices, automobiles, and aircraft. Optimization of the length of the connectors in such systems results in minimizing the weight of the system and reducing its energy consumption. In addition to electromechanical systems, the layout problem is omnipresent in wind farm design. The decision of allocating wind turbines to feasible locations in the farm is a mixed-integer

optimization problem that could be modeled and solved using the proposed method in Chapter 3 of this dissertation.

In this study, also a graphical representation of the 3D cluttered environment of path planning problems is proposed. The generated graph is a representative set of design solutions for the routes of 3D flexible connectors. If the shortest route is desired, Dijkstra's search algorithm is applied to the graph that finds the minimum cost (here cost is the length of the path) path. Otherwise, depending on other physical constraints of the problem, such as accessibility of wires/components, maintenance, bend radius of the connector, or thermal loading, other routes could be selected from the graph. The advantage of graph-based over reasoning-based methods is the generation of a solution set rather than a single solution. This is especially advantageous in design problems where alternative solutions are sought for the designers to have flexibility in their decision making. The additional constraints can either be considered later in the decision making or be introduced as penalty functions in the objective of the optimization problem.

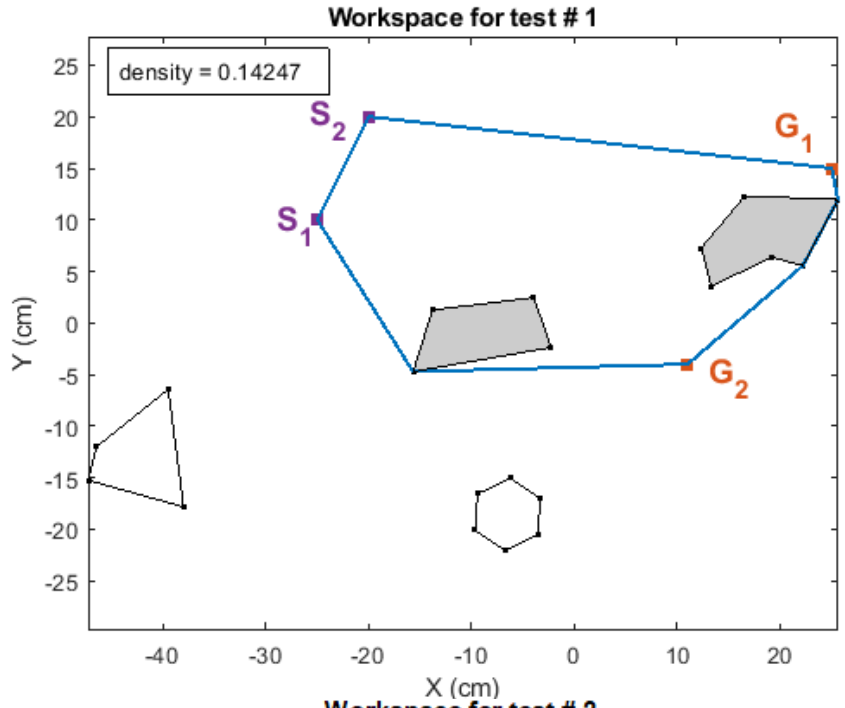
While the graph-based routing method can apply to workspaces with any convex shapes of obstacles, the fewer faces an obstacle has, the shorter it will take for the algorithm to determine the graph. Therefore, for problems with complex shapes of obstacles where an acceptable solution needs to be found in a reasonable amount of time, for example in automotive systems, the obstacles may be approximated by their bounding boxes to reduce the number of their associated faces. Other areas where the method can be applied to find routes of wiring connectors without major modifications to the algorithm include computers that contain components of regular convex shapes (e.g. rectangular blocks).

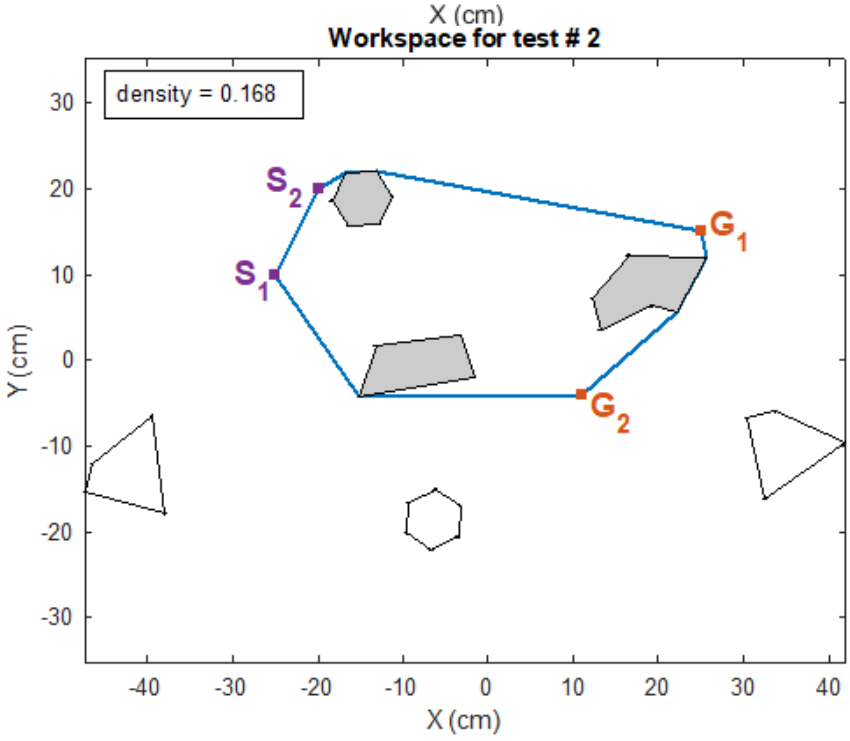
This dissertation shows that it may be possible to have optimization algorithms to deterministically identify the shortest routes in cluttered environments. As computing power keeps enhancing, larger and larger problems can be tackled and ultimately provide the designers with better tools to address such problems and eventually be confident in their design decisions.

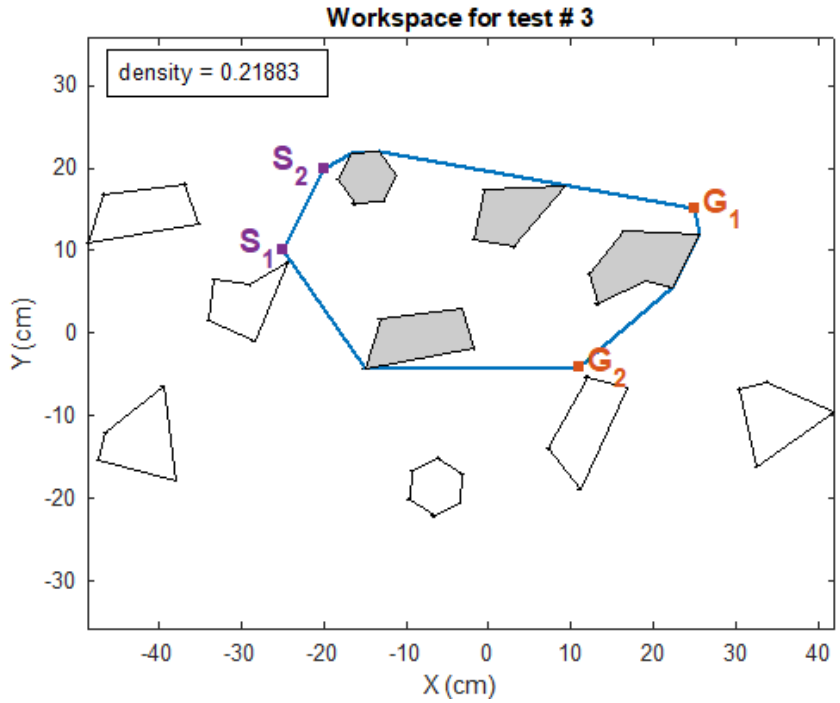
APPENDICES

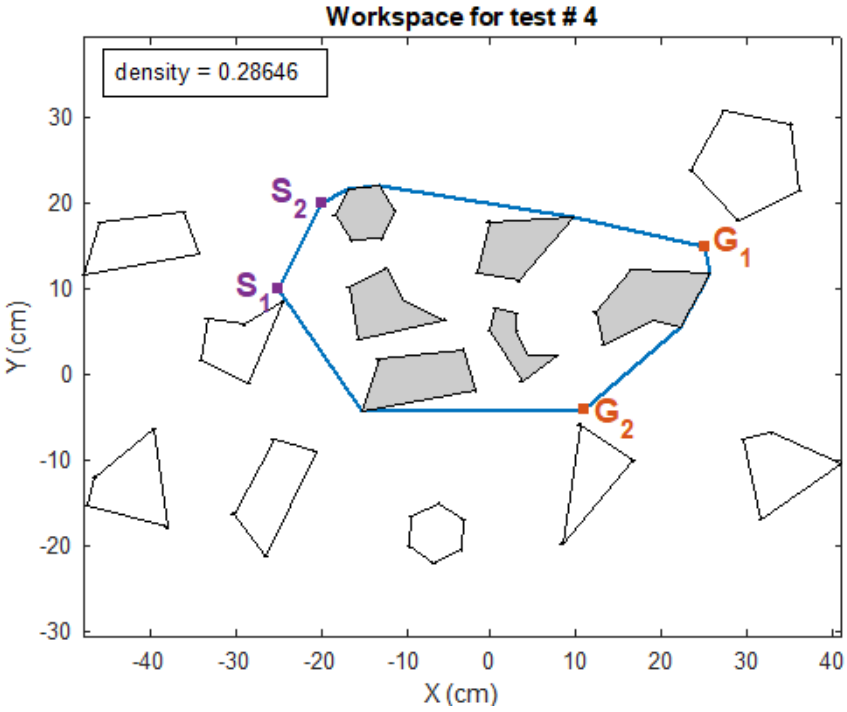
APPENDIX A: ADDITIONAL RESULTS FOR LAYOUT OPTIMIZATION PROBLEM

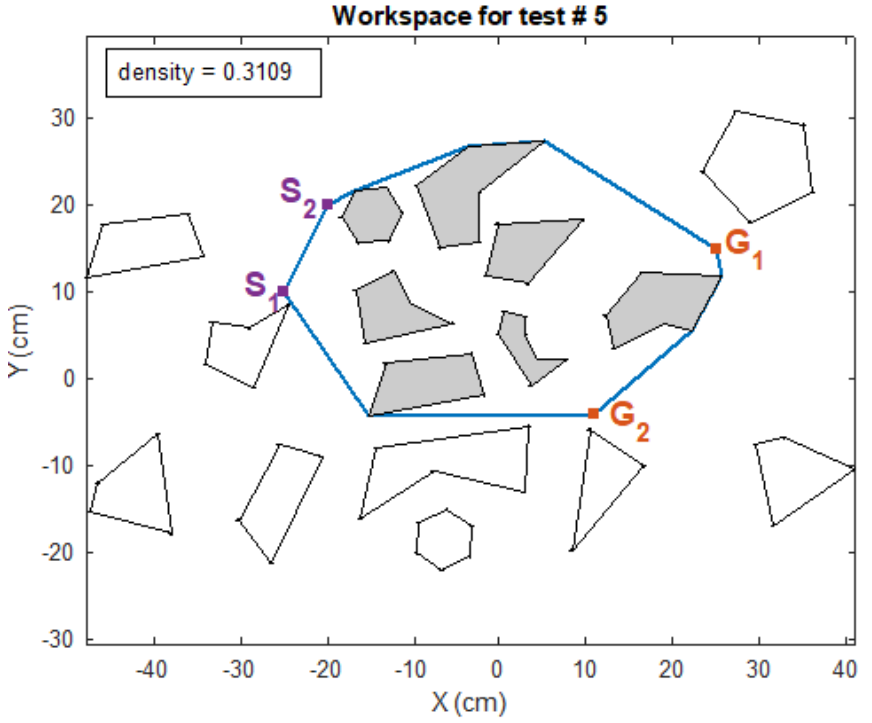
Workspaces of the test cases used to evaluate the effects of density

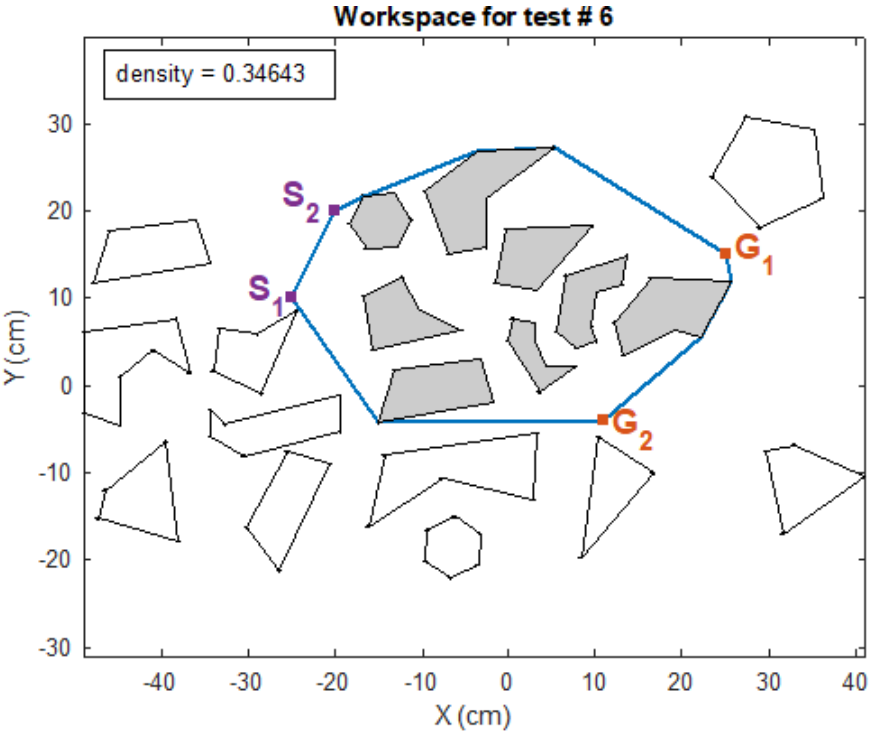


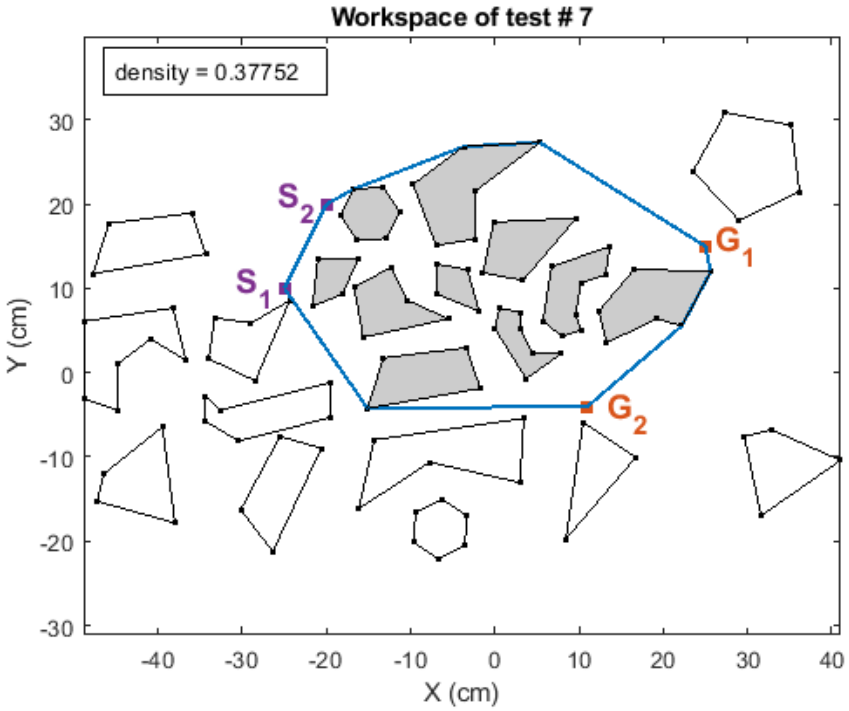


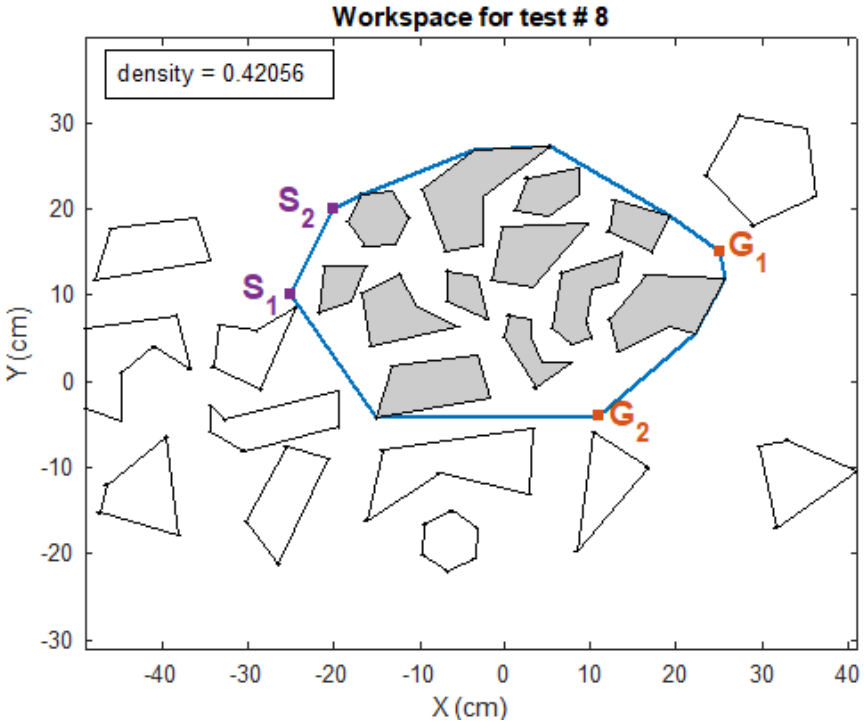

Workspace for test # 3
density = 0.21883
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40

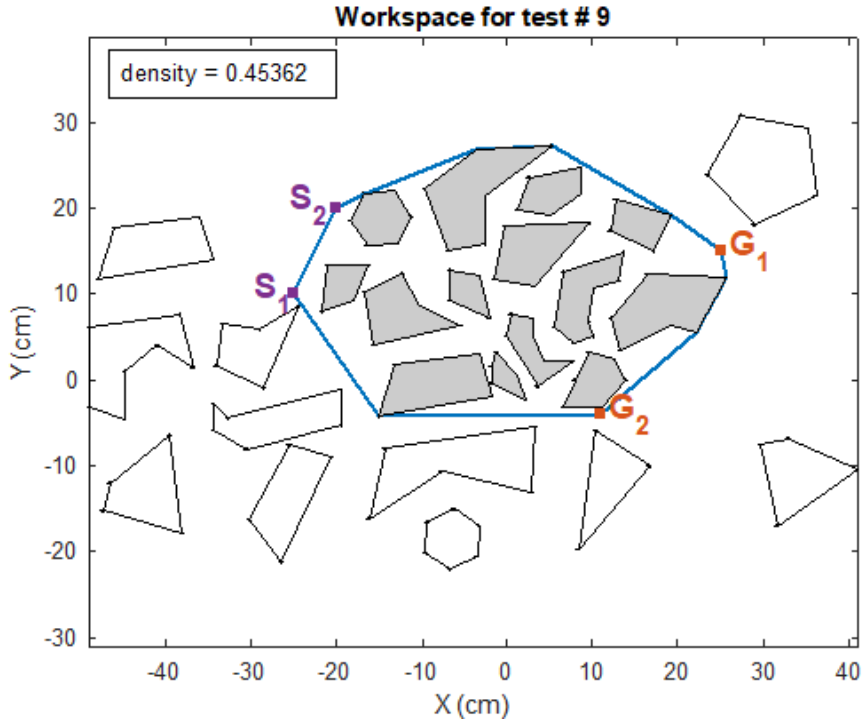

Workspace for test # 4
density = 0.28646
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40

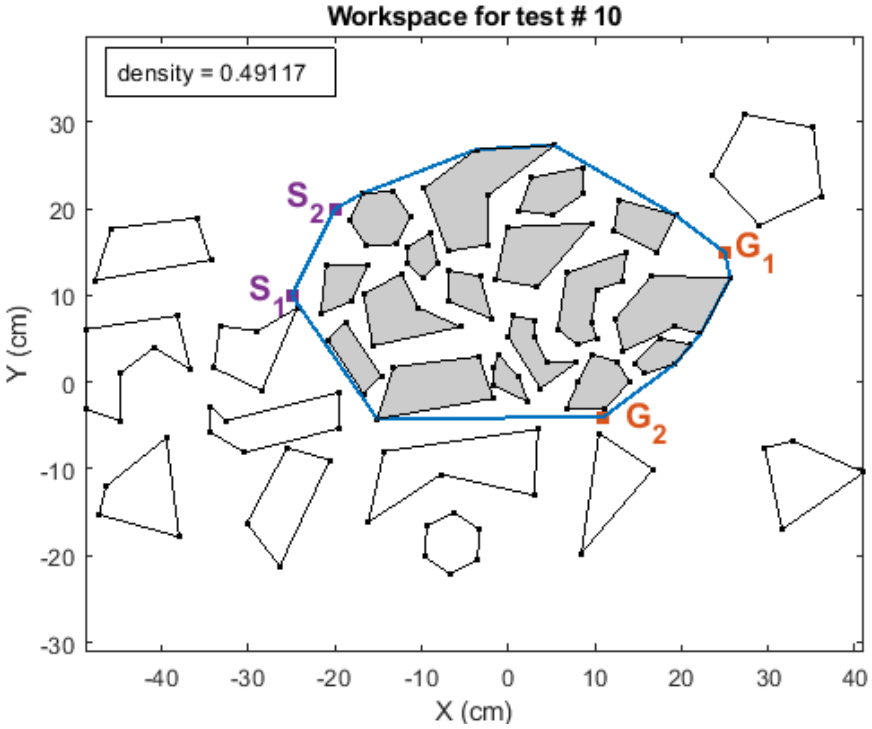

Workspace for test # 5
density = 0.3109
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40

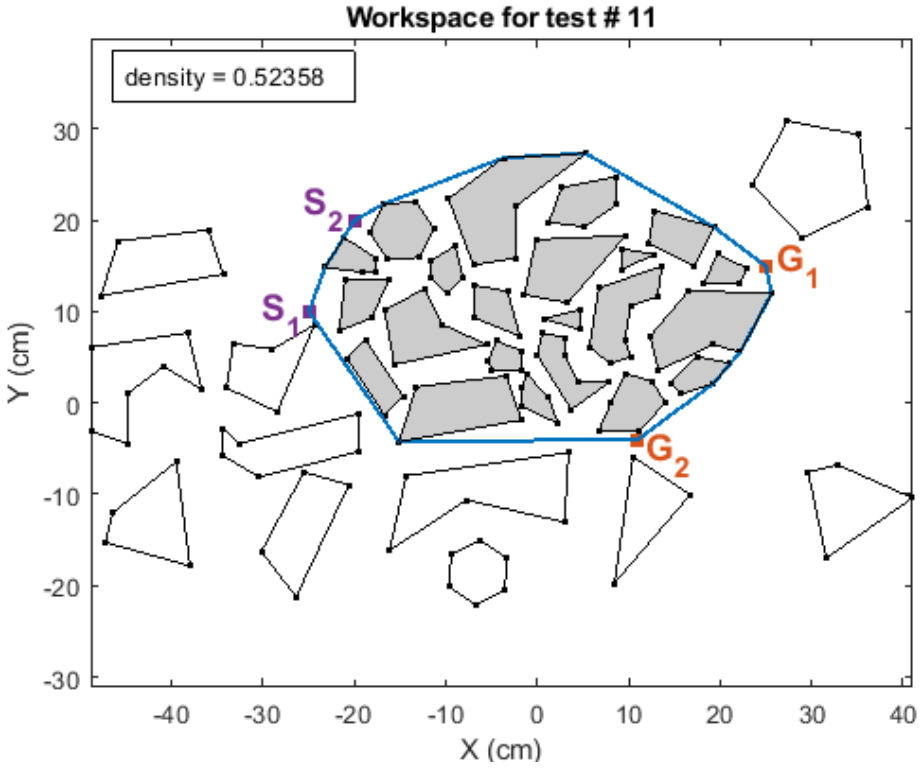

Workspace for test # 6
density = 0.34643
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40


Workspace of test # 7
density = 0.37752
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40


Workspace for test # 8
density = 0.42056
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40


Workspace for test # 9
density = 0.45362
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40


Workspace for test # 10
density = 0.49117
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40


Workspace for test # 11
density = 0.52358
S2
S1
G1
G2
Y (cm)
X (cm)
30
20
10
0
-10
-20
-30
-40
-30
-20
-10
0
10
20
30
40

BIBLIOGRAPHY

[1] Matheus, K., and Konigseder, T., 2015, Automotive Ethernet, Cambridge University Press.

[2] Lindfors, N., Pesonen, A., Franck, C., and Kuosmanen, P., 2003, "CABLING DESIGN UTILIZING 3D CAD IN PRODUCT DEVELOPMENT OF AN ELECTRIC DEVICE," INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN ICED 03, A. Folkeson, K. Gralen, M. Norell, and U. Sellgren, eds., Stockholm, pp. 1–8.

[3] Park, H., Lee, S. H., and Cutkosky, M. R., 1992, "Computational Support for Concurrent Engineering of Cable Harnesses," Computers in Engineering Conference, San Francisco, CA, USA, USA.

[4] Ng, F. M., Ritchie, J. M., and Simmons, J. E. L., 2000, "The design and planning of cable harness assemblies," Proc. Inst. Mech. Eng. Part B J. Eng. Manuf., **214**(10), pp. 881–890.

[5] Dijkstra, E. W., 1959, "A note on two problems in connexion with graphs.pdf," Numer. Math., pp. 269–271.

[6] Hart, P. E., Nilsson, N. J., and Raphael, B., 1968, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. Syst. Sci. Cybern., **4**(2), pp. 100–107.

[7] Canny, J., and Reif, J., 1987, "New Lower Bound Techniques for Robot Motion Planning Problems.," Annu. Symp. Found. Comput. Sci., pp. 49–60.

[8] Latombe, J.-C., 1991, Robot Motion Planning, Kluwer Academic Publishers, Boston.

[9] Tran, N., Dinneen, M. J., and Linz, S., 2019, Multi-criteria Shortest Paths in 3D among Vertical Obstacles, Department of Computer Science, The University of Auckland, New Zealand.

[10] O'Rourke, J., and Mallinckrodt, A. J., 1995, Computational Geometry in C, Computers in Physics.

[11] Nilsson, N. J., 1969, "A Mobius Automation: An Application of Artificial Intelligence Techniques," Proceedings of the 1st international joint conference on Artificial intelligence, Morgan Kaufmann Publishers Inc., pp. 509–520.

[12] Wangdahl, G. E., Pollock, S. M., and Woodward, J. B., 1974, "Minimum-Trajectory Pipe Routing," J. Sh. Res., **18**(1), pp. 46–49.

[13] Lozano-Pérez, T., and Wesley, M. a., 1979, “An algorithm for planning collision-free paths among polyhedral obstacles,” Commun. ACM, **22**(10), pp. 560–570.

[14] Ma, Y., Zheng, G., and Perruquetti, W., 2013, “Cooperative path planning for mobile robots based on visibility graph,” Control Conference (CCC), 2013 32nd Chinese. IEEE, pp. 4915–4920.

[15] Papadimitriou, C. H., 1985, “An algorithm for shortest-path motion in three dimensions,” Inf. Process. Lett., **20**(5), pp. 259–263.

[16] Gao, B., Xu, D., Zhang, F., and Yao, Y., 2009, “Constructing Visibility Graph and Planning Optimal Path for Inspection of 2D workspace,” Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on, IEEE, Shanghai, China, pp. 693–698.

[17] Udupa, S. M., 1977, “Collision detection and avoidance in computer controlled manipulators,” California Institute of Technology.

[18] Hahmann, S., Miksch, J., Resch, B., Lauer, J., and Zipf, A., 2017, “Routing through open spaces – a performance comparison of algorithms,” Geo-spatial Inf. Sci., pp. 1–10.

[19] Souissi, O., Benatitallah, R., Duvivier, D., and Artiba, A., 2013, “Path Planning : A 2013 Survey,” Industrial Engineering and Systems Management (IESM), Proceedings of 2013 International Conference on, IEEE, Rabat, Morocco, pp. 1–8.

[20] Lee, D.-T., 1978, “Proximity and Reachability in The Palne,” ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB.

[21] Welzl, E., 1985, “Constructing the visibility graph for n-line segments in O (n2) time,” Inf. Process. Lett., **20**(4), pp. 167–171.

[22] Asano, T., Asano, T., Guibas, L., Hershberger, J., and Imai, H., 1985, “Visibility-Polygon Search and Euclidean Shortest Paths,” Found. Comput. Sci., pp. 155–164.

[23] Asano, T., Asano, T., Guibas, L., Hershberger, J., and Imai, H., 1986, “Visibility of Disjoint Polygons,” Algorithmica, **1**(1–4), pp. 49–63.

[24] Sharir, M., and Schorr, A., 1986, “On shortest paths in polyhedral Spaces,” SIAM J. Comput., **15**(1), pp. 144–153.

[25] Ghosh, S. K., and Mount, D. M., 1991, “An Output Sensitive Algorithm for Computing Visibility

Graphs," SIAM J. Comput., **20**(5), pp. 888–910.

[26] Chazelle, B., 1982, "A Theorem on Polygon Cutting with Applications," Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on, IEEE, Chicago, IL, USA, USA, pp. 339–349.

[27] Kapoor, S., and Maheshwari, S. N., 2000, "Efficiently constructing the visibility graph of a simple polygon with obstacles," SIAM J. Comput., **30**(3), pp. 847–871.

[28] Storer, J. a, and Reif, J. H., 1994, "Shortest paths in the plane with polygonal obstacles," Inf. Process. Lett., **23**(5), pp. 982–1012.

[29] Wein, R., Jur P., V. den B., and Halperin, D., 2007, "The visibility – Voronoi complex and its applications ☆," Comput. Geom., **36**(1), pp. 66–87.

[30] Clarkson, K., 1987, "Approximation algorithms for shortest path motion planning," Proc. Ninet. Annu. ACM Symp. Theory Comput., pp. 56–65.

[31] Fredman, M. L., and Tarjan, R. E., 1987, "Fibonacci heaps and their uses in improved network optimization algorithms.pdf," J. ACM, **34.3**, pp. 596–615.

[32] Hershberger, J., and Guibas, L. J., 1988, "An O(n 2) shortest path algorithm for a non-rotating convex body," J. Algorithms, **9**(1), pp. 18–46.

[33] Rohnert, H., 1986, "Shortest Paths in the Plane with Convex Polygonal Obstacles," Inf. Process. Lett., **23**, pp. 71–76.

[34] Priya, T. K., and Sridharan, K., 2004, "An Efficient Algorithm to Construct Reduced Visibility Graph and its FPGA Implementation," VLSI Design, 2004. Proceedings. 17th International Conference on. IEEE, IEEE, Mumbai, India, pp. 1057–1062.

[35] Jan, G. E., Sun, C., Tsai, W. C., and Lin, T., 2014, "An O (n log n) Shortest Path Algorithm Based on Delaunay Triangulation," 660 IEEE/ASME Trans. MECHATRONICS, **19**(2), pp. 660–666.

[36] Pillai, A. C., Chick, J., Johanning, L., Khorasanchi, M., and Laleu, V. De, 2015, "Offshore wind farm electrical cable layout optimization," Eng. Optim., **47**(12), pp. 1689–1708.

[37] Qureshi, A. H., and Ayaz, Y., 2015, "Intelligent bidirectional rapidly-exploring random trees for

optimal motion planning in complex cluttered environments," Rob. Auton. Syst., **68**, pp. 1–11.

[38] Jafarzadeh, H., and Fleming, C. H., 2018, "An Exact Geometry-Based Algorithm for Path Planning," Int. J. Appl. Math. Comput. Sci., **28**(3), pp. 493–504.

[39] Huang, H.-P., and Chung, S.-Y., 2004, "Dynamic Visibility Graph for Path Planning," IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, Sendai, Japan, pp. 2813–2818.

[40] Toan, T. Q., Sorokin, A. A., Thi, V., and Trang, H., 2017, "Using modification of visibility-graph in solving the problem of finding shortest path for robot," International Siberian Conference on Control and Communications (SIBCON), pp. 1–6.

[41] Gasilov, N., Doğan, M., and Arici, V., 2011, "Two-stage Shortest Path Algorithm for Solving Optimal Obstacle Avoidance Problem Two-stage Shortest Path Algorithm for Solving Optimal Obstacle Avoidance Problem," IETE J. Res., **57**(3), pp. 278–285.

[42] Badariyah, N., Latip, A., Omar, R., and Debnath, S. K., 2017, "Optimal Path Planning using Equilateral Spaces Oriented Visibility Graph Method," Int. J. Electr. Comput. Eng., **7**(6), pp. 3046–3051.

[43] Graser, A., 2016, "Integrating Open Spaces into OpenStreetMap Routing Graphs for Realistic Crossing Behaviour in Pedestrian Navigation," GI_Forum 2016, pp. 217–230.

[44] Garrido, S., Moreno, L., Abderrahim, M., and Martin, F., 2006, "Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching," Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, Beijing, China, pp. 2376–2381.

[45] Ó'Dúnlaing, C., and Yap, C. K., 1985, "A ' Retraction ' Method for Planning of a Disc the Motion," J. Algorithms, **6**(1), pp. 104–111.

[46] Brooks, R. A., 1983, "Solving the Find-Path Problem by Good Representation of Free Space," IEEE Trans. Syst. Man. Cybern., **13**(3), pp. 190–197.

[47] Wein, R., and Halperin, D., 2005, "The Visibility – Voronoi Complex and Its Applications," Proc. twenty-first Annu. Symp. Comput. Geom., pp. 63–72.

[48] Bhattacharya, P., and Gavrilova, M. L., 2007, "Geometric Algorithms for Clearance Based Optimal Path Computation," Proceedings of the 15th International Symposium on Advances in Geographic

Information Systems ACM GIS 2007, pp. 1–4.

[49] Leven, D., and Sharir, M., 1987, "Planning a Purely Translational Motion for a Convex Object in Two-Dimensional Space Using Generalized Voronoi Diagrams," Discrete Comput. Geom., **2**(1), pp. 9–31.

[50] Zhang, L., and Manocha, D., 2008, "An Efficient Retraction-based RRT Planner," IEEE International Conference on Robotics and Automation, Pasadena, CA, pp. 3743–3750.

[51] Takahashi, O., and Schilling, R. J., 1989, "Motion Planning in a Plane Using Generalized Voronoi Diagrams," IEEE Trans. Robot. Autom., **5**(2), pp. 143–150.

[52] Alt, H., and Yap., C. K., 1989, "Algorithmic Aspect of Motion Planning: A Tutorial," Algorithms Review, pp. 173–196.

[53] Schwartz, J. T., Sharir, M., and Hopcroft, J. E., 1987, Planning, geometry, and complexity of robot motion, Intellect Books.

[54] Schwartz, J. T., and Sharir, M., 1983, "On the 'piano movers'" problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers,'" Commun. pure Appl. Math., **36**(3), pp. 345–398.

[55] Brooks, R. A., and Lozano-Perez, T., 1985, "A subdivision algorithm in configuration space for findpath with rotation.," IEEE Trans. Syst. Man. Cybern., **2**, pp. 224–233.

[56] Faverjon, B., 1984, "Obstacle avoidance using an octree in the configuration space of a manipulator.," Robotics and Automation. Proceedings. 1984 IEEE International Conference on, IEEE, Atlanta, GA, USA, pp. 504–512.

[57] Asmara, A., and Nienhuis, U., 2006, "Automatic piping system in ship," The 5th International Conference on Computer and IT Applications in the Maritime Industries, pp. 269–280.

[58] Bhattacharya, B. Y. P., and Gavrilova, M. L., 2008, "Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path," IEEE Robot. Autom. Mag., **15**(2).

[59] Khatib, O., 1986, "Real-time obstacle avoidance for manipulators and mobile robots," Auton. Robot Veh., pp. 396–404.

[60] Overmars, M. H., 1992, A Random Approach to Motion Planning, Utrecht, the Netherlands.

[61] Elbanhawi, M., and Simic, M., 2014, “Sampling-Based Robot Motion Planning : A Review,” IEEE Access, **2**, pp. 56–77.

[62] Warren, C. W., 1989, “Global path planning using artificial potential fields,” Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on, IEEE, pp. 316–321.

[63] Ge, S. S., and Cui, Y. J., 2000, “New potential functions for mobile robot path planning,” IEEE Trans. Robot. Autom., **16**(5), pp. 615–620.

[64] Luh, G., and Liu, W., 2008, “An immunological approach to mobile robot reactive navigation,” Appl. Soft Comput., **8**(1), pp. 30–45.

[65] Li, G., Yamashita, A., Asama, H., and Tamura, Y., 2012, “An Efficient Improved Artificial Potential Field Based Regression Search Method for Robot Path Planning,” Mechatronics and Automation (ICMA), 2012 International Conference on, Chengdu, China, pp. 1227–1232.

[66] Vadakkepat, P., Lee, T. H., and Xin, L., 2001, “Application of Evolutionary Artificial Potential Field in Robot Soccer System,” IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th, IEEE, Vancouver, BC, Canada, Canada, pp. 2781–2785.

[67] Bohlin, R., and Kavraki, L. E., 2000, “Path Planning Using Lazy PRM,” Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on, IEEE, San Francisco, CA, USA, USA, pp. 521–528.

[68] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H., 1996, “Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces,” IEEE Trans. Robot. Autom., **12**(4), pp. 566–580.

[69] Kavraki, L., and Lat, J.-C., 1994, “Randomized Preprocessing of Configuration Space for Fast Path Planning,” Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, IEEE, San Diego, CA, USA, pp. 2138–2145.

[70] Geraerts, R., and Overmars, M. H., 2004, “A Comparative Study of Probabilistic Roadmap Planners,” Algorithmic Found. Robot. V, **7**, pp. 43–57.

[71] Saha, M., Latombe, J.-C., Chang, Y.-C., and Prinz, F., 2005, “Finding Narrow Passages with Probabilistic Roadmaps : The Small-Step,” Auton. Robots, **19**(3), pp. 301–319.

[72] Kuffner, J. J., and LaValle, S. M., 2000, "RRT-connect: An efficient approach to single-query path planning," Proc. 2000 ICRA. Millenn. Conf. IEEE Int. Conf. Robot. Autom. Symp. Proc. (Cat. No.00CH37065), **2**(April), pp. 995–1001.

[73] Lu, J., Diaz-Mercado, Y., Egerstedt, M., Zhou, H., and Chow, S. N., 2014, "Shortest paths through 3-dimensional cluttered environments," Proceedings - IEEE International Conference on Robotics and Automation, IEEE, pp. 6579–6585.

[74] Chow, S., 2013, "Global Optimizations by Intermittent Diffusion," Chaos, CNN, Memristors Beyond, pp. 466–479.

[75] Bhattacharya, B. Y. P., and Gavrilova, M. L., 2008, "Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path," IEEE Robot. Autom. Mag., **15**(2), pp. 58–66.

[76] Sandurkar, S., and Chen, W., 1999, "GAPRUS - genetic algorithms based pipe routing using tessellated objects," Comput. Ind., **38**(3), pp. 209–223.

[77] Masehian, E., and Sedighizadeh, D., 2007, "Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review," World Acad. Sci. Eng. Technol., **29**(1), pp. 101–106.

[78] Zachariadis, E. E., Tarantilis, C. D., and Kiranoudis, C. T., 2009, "A Guided Tabu Search for the Vehicle Routing Problem with two-dimensional loading constraints," Eur. J. Oper. Res., **195**(3), pp. 729–743.

[79] Gao, R., and Setup, A. C., 2016, "Complex Housing: Modelling and Optimization Using an Improved Multi-Objective Simulated Annealing Algorithm," ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, American Society of Mechanical Engineers, Charlotte, North Carolina, USA, pp. 1–12.

[80] Ahn, C. W., and R., R., 2002, "A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations," IEEE Trans. Evol. Comput., **6**(6), pp. 566–579.

[81] Liu, Q., and Wang, C., 2011, "A discrete particle swarm optimization algorithm for rectilinear branch pipe routing," Assem. Autom., **31**(4), pp. 363–368.

[82] Thantulage, G., Kalganova, T., and Wilson, M., 2006, "Grid Based and Random Based Ant Colony Algorithms for Automatic Hose Routing in 3D Space," Trans. Eng. Comput. Technol., **14**, pp. 144–

150.

[83] Masoudi, N., Fadel, G., and Wiecek, M. M., 2019, “Planning the Shortest Path in Cluttered Environments: A Review and a Planar Convex Hull-Based Approach,” J. Comput. Inf. Sci. Eng., **19**(4).

[84] Sharir, M., and Schorr, A., 1986, “On shortest paths in polyhedral Spaces,” SIAM J. Comput., **15.1**(1), pp. 193–215.

[85] Rohnert, H., 1986, “Shortest Paths in the Plane with Convex Polygonal Obstacles,” Information, **23**, pp. 71–76.

[86] Yan, J., Zuo, D. W., Jiao, G. M., and Li, J. P., 2008, “Survey on the Design and Planning of Cable Harness Assemblies in Electromechanical Products,” Appl. Mech. Mater. Vols., **10**, pp. 889–893.

[87] Sheridan, H. C., 1976, “AN OVERVIEW OF A CASDAC SUBSYSTEM-COMPUTER-AIDED PIPING DESIGN AND CONSTRUCTION (CAPDAC),” Nav. Eng. J., **88**(5), pp. 87–98.

[88] Kang, S., Sehyun, M., and Han, S., 1999, “A Design expert system for auto-routing of ship pipes,” J. Sh. Prod., **15**(1), pp. 1–9.

[89] Roh, M. Il, Lee, K. Y., and Choi, W. Y., 2007, “Rapid generation of the piping model having the relationship with a hull structure in shipbuilding,” Adv. Eng. Softw., **38**(4), pp. 215–228.

[90] Billsdon, R., and Wallington, K., 1998, “Wiring harness design can a computer help?,” Comput. Control Eng. J., **9**(4), pp. 163–167.

[91] Mingji, H., and Dong, X., 2010, “Research on Flexible Cable Geometric Modeling Technology in Virtual Maintenance Based on VRML,” Mechanic Automation and Control Engineering (MACE), 2010 International Conference on, IEEE, Wuhan, China, pp. 772–775.

[92] Han, N., and Guo, L., 2017, “THE RESEARCH ON ROUTING OPTIMIZATION METHOD USING UNDIRECTED-GRAPH IN CABLE HARNESS DESIGN,” Proceedings of the ASME 2017 International Mechanical Engineering Congress and Exposition IMECE2017, Tampa, Florida, USA, pp. 1–6.

[93] Lin, C., Rao, L., Ambrosio, J. D., and Sangiovanni-vincentelli, A., 2014, “Electrical Architecture Optimization and Selection - Cost Minimization via Wire Routing and Wire Sizing,” SAE Int. J.

Passeng. Cars-Electronic Electr. Syst., **7**(2), pp. 502–509.

[94] Valentini, P. P., 2011, "Interactive cable harnessing in augmented reality," Int J Interact DesManuf, **5**, pp. 45–53.

[95] Ritchie, J., Simmons, J., Holt, P., and Russell, G., 2002, "Immersive virtual reality as an interactive tool for cable harness design," Proceedings of PRASIC (2002), pp. 249–255.

[96] Simmons, J., Ritchie, J., and Holt, P. O. B., 2002, "Immersing the Human in the Design : Design-for-Manufacture of Cable Harnesses," Proceedings of SCI/ISAS 2002 Volume XXII.

[97] Ritchie, J. M., Robinson, G., Day, P. N., Dewar, R. G., Sung, R. C. W., and Simmons, J. E. L., 2007, "Cable harness design, assembly and installation planning using immersive virtual reality," Virtual Real., **11**(4), pp. 261–273.

[98] O'B Holt, P., Ritchie, J. M., Day, P. N., Simmons, J. E. L., Robinson, G., Russell, G. T., and Ng, F. M., 2004, "Immersive Virtual Reality In Cable and Pipe Routing: Design Metaphors and Cognitive Ergonomics," J. Comput. Inf. Sci. Eng., **4**(3), pp. 161–170.

[99] Park, H., Cutkosky, M. R., Conru, A. B., and Lee, S., 1994, "An Agent-Based Approach to Concurrent Cable Harness Design," AI EDAM, **8**(1), pp. 45–61.

[100] Wu, Y., Champaneri, R., and Mehta, P., 1992, "Use of the expert system as a design tool for the cable harness design.," APPL ARTIF INTELL ENG., COMPUTATIONAL MECHANICS PUBL, SOUTHAMPTON(ENGL), 1992, pp. 357–372.

[101] Shang, W., Liu, J., Ning, R., and Liu, J., 2012, "A Computational Framework for Cable Layout Design in Complex Products," Phys. Procedia, **33**, pp. 1879–1885.

[102] Van Der Velden, C., Bil, C., Yu, X., and Smith, A., 2007, "An intelligent system for automatic layout routing in aerospace design," Innov. Syst. Softw. Eng., **3**(2), pp. 117–128.

[103] Cerezuela, C., Cauvin, A., Boucher, X., and Kieffer, J.-P., 1998, "A Decision Support System for a Concurrent Design of Cable Harnesses: Conceptual Approach and Implementation," Concurr. Eng., **6**(1), pp. 43–52.

[104] Pemarathne, W. P. J., and Fernando, T. G. I., 2016, "Wire and cable routings and harness designing systems with AI , a Review," Information and Automation for Sustainability (ICIAfS), 2016 IEEE

International Conference on, IEEE, Galle, Sri Lanka, pp. 1–6.

[105] Teegavarapu, S., Summers, J. D., and Mocko, G. M., 2008, "Case Study Method for Design Research: A Justification," Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference Proceedings of IDETC/DTM 2008 IDETC/CIE 2008 ASME 2008 International Design Engineering Technical Conferences August , Brooklyn, New York, USA, pp. 1–9.

[106] Ng, F. M., Ritchic, J. M., and Simmons, J. E. L., 2000, "The design and planning of cable harness assemblies," Proc. Inst. Mech. Eng. Part B J. Eng. Manuf., **214**(10), pp. 881–890.

[107] Lin, M. H., Tsai, J. F., and Yu, C. S., 2012, "A review of deterministic optimization methods in engineering and management," Math. Probl. Eng., **2012**.

[108] Prim, R. C., 1957, "Shortest Connection Networks And Some Generalizations," Bell Syst. Tech. J., **36**(6), pp. 1389–1401.

[109] Kruskal, J. B., 1955, "On the shortest spanning subtree of a graph and the traveling salesman problem.," Proceedings of the American Mathematical society, pp. 48–50.

[110] Gilbert, E. N., and Pollak, H. O., 1968, "Steiner Minimal Trees," SIAM J. Appl. Math., **16**(1), pp. 1–29.

[111] Sankoff, D., and Rousseau, P., 1975, "Locating the vertices of a steiner tree in an arbitrary metric space," Math. Program., **9**(1), pp. 240–246.

[112] Garey, M. R., and Johnson, D. S., 1977, "The rectilinear Steiner tree problem," SIAM J. Appl. Math., **32**(4), pp. 826–834.

[113] Fonseca, R., Brazil, M., Winter, P., and Zachariasen, M., 2014, "Faster exact algorithms for computing Steiner trees in higher dimensional Euclidean spaces," 11th DIMACS Implementation Challenge, Brown University.

[114] Suchý, O., 2017, "Extending the Kernel for Planar Steiner Tree to the Number of Steiner Vertices," Algorithmica, **79**(1), pp. 189–210.

[115] Juhl, D., Warme, D. M., Winter, P., and Zachariasen, M., 2018, "The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study," Math. Program.

Comput., **10**(4), pp. 487–532.

[116] Kou, L., Markowsky, G., and Berman, L., 1981, "A fast algorithm for Steiner trees," Acta Inform., **15**(2), pp. 141–145.

[117] Imase, M., and Waxman, B. M., 1991, "Dynamic Steiner Tree Problem," SIAM J. Discret. Math., **4**(3), pp. 369–384.

[118] Kapsalis, A., Rayward-Smith, V. J., and Smith, G. D., 1993, "Solving the Graphical Steiner Tree Problem Using Genetic Algorithms," J. Oper. Res. Soc., **44**(4), pp. 397–406.

[119] Koch, T., and Martin, A., 1998, "Solving Steiner tree problems in graphs to optimality," Networks An Int. J., **32**(3), pp. 207–232.

[120] Warme, D. M., Winter, P., and Zachariasen, M., 2000, "Exact Algorithms for Plane Steiner Tree Problems: A Computational Study," Adv. Steiner trees. Comb. Optim., **6**, pp. 81–116.

[121] Robins, G., and Zelikovsky, A., 2008, Minimum Steiner Tree Construction*.

[122] Byrka, J., Grandoni, F., Rothvo, T., and Sanità, L., 2010, "An Improved LP-based Approximation for Steiner Tree," Proceedings of the forty-second ACM symposium on Theory of computing, pp. 583–592.

[123] Kumar, S., 2014, "A Simple Algorithm for Steiner Tree Problem in Networks," Int. J. Appl. or Innov. Eng. Manag., **3**(7), pp. 232–236.

[124] Lin, C. W., Rao, L., Giusto, P., D'Ambrosio, J., and Sangiovanni-Vincentelli, A. L., 2015, "Efficient Wire Routing and Wire Sizing for Weight Minimization of Automotive Systems," IEEE Trans. Comput. Des. Integr. Circuits Syst., **34**(11), pp. 1730–1741.

[125] Sommer, J., Doumith, E. A., and Duva, Q., 2009, "On Link Harness Optimization of Embedded Ethernet Networks," Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on, IEEE, Lausanne, Switzerland, pp. 191–200.

[126] Vasko, F. J., Barbieri, R. S., Rieksts, B. Q., Reitmeyer, K. L., and Stott, K. L., 2002, "The cable trench problem: Combining the shortest path and minimum spanning tree problems," Comput. Oper. Res., **29**(5), pp. 441–458.

[127] Jeng, D. J. F., and Watada, J., 2007, "Non-deterministic algorithm for routing optimization: A case

study," Second International Conference on Innovative Computing, Information and Control, ICICIC 2007.

[128] Marianov, V., Gutiérrez-Jarpa, G., Obreque, C., and Cornejo, O., 2012, "Lagrangean relaxation heuristics for the p-cable-trench problem," Comput. Oper. Res., **39**(3), pp. 620–628.

[129] Schwarze, S., 2015, "The multi-commodity cable trench problem," 23rd European Conference on Information Systems, ECIS 2015, Münster, Germany, pp. 1–14.

[130] Vasko, F. J., Landquist, E., Kresge, G., Tal, A., Jiang, Y., and Papademetris, X., 2016, "A Simple and Efficient Strategy for Solving Very Large-Scale Generalized Cable-Trench Problems," Networks, **67**(3), pp. 199–208.

[131] Calik, H., Leitner, M., and Luipersbeck, M., 2017, "A Benders decomposition based framework for solving cable trench problems," Comput. Oper. Res., **81**, pp. 128–140.

[132] Zyma, K., Girard, J. N., Landquist, E., Schaper, G., and Vasko, F. J., 2017, "Formulating and solving a radio astronomy antenna connection problem as a generalized cable-trench problem: an empirical study," Int. Trans. Oper. Res., **24**(5), pp. 943–957.

[133] Winter, P., 1993, "Euclidean Steiner minimal trees with obstacles and Steiner visibility graphs," Discret. Appl. Math., **47**(2), pp. 187–206.

[134] Zachariasen, M., and Winter, P., 2002, "Obstacle-Avoiding Euclidean Steiner Trees in the Plane: An Exact Algorithm," Goodrich M.T., McGeoch C.C. Algorithm Eng. Exp. ALENEX 1999. Lect. Notes Comput. Sci., **1619**, pp. 286–299.

[135] Müller-Hannemann, M., and Tazari, S., 2010, "A near linear time approximation scheme for Steiner tree among obstacles in the plane," Comput. Geom. Theory Appl., **43**(4), pp. 395–409.

[136] Parque, V., and Miyashita, T., 2018, "Obstacle-avoiding euclidean steiner trees by n-star bundles," Proc. - Int. Conf. Tools with Artif. Intell. ICTAI, **2018**-**Novem**, pp. 315–319.

[137] Chiang, C., Wong, C. K., and Sarrafzadeh, M., 1994, "A Weighted Steiner Tree-Based Global Router with Simultaneous Length and Density Minimization," IEEE Trans. COMPLITER-AIDED Des. lNTEGRATED CIRCUITS SYSTRMS, **13**(12), pp. 1461–1469.

[138] Ganley, J. L., and Cohoon, J. P., 1994, "Routing a multi-terminal critical net: Steiner tree

construction in the presence of obstacles," Proceedings - IEEE International Symposium on Circuits and Systems, pp. 113–116.

[139] Hu, Y., Feng, Z., Jing, T., Hong, X., Yang, Y., Yu, G., Hu, X., and Yan, G., 2004, "FORst: A 3-step heuristic for obstacle-avoiding rectilinear Steiner minimal tree construction," J. Inf. Comput. Sci., **1**(3), pp. 107–116.

[140] Wu, P. C., Gao, J. R., and Wang, T. C., 2007, "A fast and stable algorithm for obstacle-avoiding rectilinear steiner minimal tree construction," Proceedings of the 2007 Asia and South Pacific Design Automation Conference, ASP-DAC, Yokohama, pp. 262–267.

[141] Lin, C.-W., Chen, S.-Y., Li, C.-F., Chang, Y.-W., and Yang, C.-L., 2008, "Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs," IEEE Trans. Comput. Des. Integr. Circuits Syst., **27**(4), pp. 643–653.

[142] Ajwani, G., Chu, C., and Mak, W. K., 2011, "FOARS: FLUTE based obstacle-avoiding rectilinear steiner tree construction," IEEE Trans. Comput. Des. Integr. Circuits Syst., **30**(2), pp. 194–204.

[143] Wędzik, A., 2014, "The Optimization of Cable Layout Design in Wind Farm Internal Networks," Acta Energ., **3**(20), pp. 144–149.

[144] Fischetti, M., Leth, J., and Borchersen, A. B., 2015, "A Mixed-Integer Linear Programming approach to wind farm layout and inter-array cable routing," Proceedings of the American Control Conference, American Automatic Control Council, pp. 5907–5912.

[145] Fischetti, M., and Pisinger, D., 2018, "Optimizing wind farm cable routing considering power losses," Eur. J. Oper. Res., **270**(3), pp. 917–930.

[146] Wędzik, A., Siewierski, T., and Szypowski, M., 2016, "A new method for simultaneous optimizing of wind farm's network layout and cable cross-sections by MILP optimization," Appl. Energy, **182**, pp. 525–538.

[147] Weber, A., 1929, Theory of the Location of Industries, University of Chicago Press.

[148] Hamacher, H. W., and Nickel, S., 1998, "Classification of location models," Locat. Sci., **6**, pp. 229–242.

[149] Nickel, S., 1993, Bicriterial and Restrictive Planar 2-Median Problems.

[150] Katz, I. N., and Cooper, L., 1981, "Facility location in the presence of forbidden regions, I Formulation and the case of Euclidean distance with one forbidden circle," Eur. J. Oper. Res., **6**(2), pp. 166–173.

[151] Aneja, Y. P., and Parlar, M., 1994, "Algorithms for Weber Facility Location in the Presence of Forbidden Regions and / or Barriers to Travel," Transp. Sci., **28**(1), pp. 70–76.

[152] Hamacher, H. W., and Klamroth, K., 2000, "Planar location problems with barriers under polyhedral gauges," Ann. Oper. Res., **96**, pp. 191–208.

[153] Klamroth, K., 2001, "A reduction result for location problems with polyhedral barriers," Eur. J. Oper. Res., **130**(3), pp. 486–497.

[154] Bischoff, M., and Klamroth, K., 2007, "An efficient solution method for Weber problems with barriers based on genetic algorithms," Eur. J. Oper. Res., **177**(1), pp. 22–41.

[155] Mcgarvey, R. G., and Cavalier, T. M., 2003, "A global optimal approach to facility location in the presence of forbidden regions," Comput. Ind. Eng., **45**, pp. 1–15.

[156] Hansen, P., Peeters, D., and Thisse, J.-F., 1981, "On the location of an obnoxious facility," Sist. Urbani, **3**(1), pp. 299–317.

[157] Kuhn, H. W., 1973, "A note on Fermat's problem," Math. Program., **4**(1), pp. 98–107.

[158] Klamroth, K., 2006, Single-facility location problems with barriers, Springer Science & Business Media.

[159] Butt, S. E., 1995, "Facility location in the presence of forbidden regions and congested regions."

[160] Bischoff, M., Fleischmann, T., and Klamroth, K., 2009, "The multi-facility location-allocation problem with polyhedral barriers," Comput. Oper. Res., **36**(5), pp. 1376–1392.

[161] Conru, A. B., 1994, "A genetic approach to the cable harness routing problem," Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, IEEE, Orlando, FL, USA, pp. 200–205.

[162] Conru, A. B., and Cutkosky, M. R., 1993, "Computational Support for Interactive Cable Harness Routing and Design," Adv. Des. Autom., **65–1**, pp. 551–558.

[163] Kimura, H., 2011, "Automatic designing system for piping and instruments arrangement including

branches of pipes," International Conference on Computer Applications in Shipbuilding (ICCAS), p. 93—99.

[164] Zhu, Z., Rocca, G. La, and Tooren, M. J. L. van, 2017, "A methodology to enable automatic 3D routing of aircraft Electrical Wiring Interconnection System," CEAS Aeronaut. J., **8**(2), pp. 287–302.

[165] Kabul, I., Gayle, R., and Lin, M. C., 2007, "Cable route planning in complex environments using constrained sampling," Proc. 2007 ACM Symp. Solid Phys. Model. - SPM '07, **1**(212), p. 395.

[166] Hermansson, T., Bohlin, R., Carlson, J. S., and Söderberg, R., 2016, "Automatic routing of flexible 1D components with functional and manufacturing constraints," Comput. Des., **79**, pp. 27–35.

[167] Klamroth, K., 2004, "Algebraic properties of location problems with one circular barrier," Eur. J. Oper. Res., **154**(1), pp. 20–35.

[168] Ng, F. M., Ritchie, J. M., Simmons, J. E. L., and Dewar, R. G., 2000, "Designing cable harness assemblies in virtual environments," J. Mater. Process. Technol., **107**(1–3), pp. 37–43.

[169] Lin, C., Rao, L., Giusto, P., Ambrosio, J. D., and Sangiovanni-vincentelli, A., 2014, "An Efficient Wire Routing and Wire Sizing Algorithm for Weight Minimization of Automotive Systems," Proceedings of the 51st Annual Design Automation Conference. ACM, pp. 1–6.

[170] Conru, A. B., and Cutkosky, M. R., 1993, "Computational Support for Interactive Cable Harness Routing and Design," Adv. Des. Autom., **1**, pp. 551–558.

[171] 2020, "paretosearch Algorithm," MathWorks [Online]. Available: https://www.mathworks.com/help/gads/paretosearch-algorithm.html.

[172] Deb, K., 2001, Multi-Objective Optimization using Evolutionary Algorithms, Wiley.

[173] 2020, "gamultiobj," Mathworks [Online]. Available: https://www.mathworks.com/help/gads/gamultiobj.html#bvf79ug-options.

[174] Haimes, Y.Y.; Lasdon, L.S.; Wismer, D. A., 1971, "On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization," IEEE Trans. Syst. Man. Cybern., **SMC-1**(3), pp. 296–297.

[175] Redish, A. D., and Jacquenot, G., 2008, "Fast InPolygon detection MEX. Retrieved May 31, 2020."

[176] Har-Peled, S., 1998, “Constructing approximate shortest path maps in three dimensions,” Proceedings of the fourteenth annual symposium on Computational geometry, pp. 383–391.

[177] Choi, J., Sellen, J., and Yap, C. K., 1997, “Approximate euclidean shortest paths in 3-space,” Int. J. Comput. Geom. Appl., **7**(4), pp. 271–295.

[178] Gewali, L. P., Ntafos, S., and Tollis, I. G., 1990, “Path planning in the presence of vertical obstacles,” IEEE Trans. Robot. Autom., **6**(3), pp. 331–341.

[179] Jiang, K., Seneviratne, L. D., and Earles, S. W. E., 1993, “Finding the 3D shortest path with visibility graph and minimum potential energy,” Proceedings of the 1993 IEEWSJ International Conference on Intelligent Robots and Systems, IEEE, Yokohama, Japan, pp. 679–684.

[180] Frontera, G., Martín, D. J., Besada, J. A., and Gu, D. W., 2017, “Approximate 3D Euclidean Shortest Paths for Unmanned Aircraft in Urban Environments,” J. Intell. Robot. Syst. Theory Appl., **85**(2), pp. 353–368.

[181] Omar, R., and Gu, D., 2010, “3D path planning for unmanned aerial vehicles using visibility line based method,” ICINCO 2010 - Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics, pp. 80–85.

[182] Liang, X., Meng, G., Xu, Y., and Luo, H., 2018, “A geometrical path planning method for unmanned aerial vehicle in 2D/3D complex environment,” Intell. Serv. Robot., **11**(3), pp. 301–312.

[183] Szykman, S., and Cagan, J., 1996, “Synthesis of Optimal Nonorthogonal Routes.pdf,” J. Mech. Des., **118**(3), pp. 419–424.

[184] Fan, X., Lin, Y., and Ji, Z., 2006, “The ant colony optimization for ship pipe route design in 3D space,” Proc. World Congr. Intell. Control Autom., **1**, pp. 3103–3108.

[185] Thantulage, G., Kalganova, T., and Fernando, W. a. C., 2006, “A Grid-based Ant Colony Algorithm for Automatic 3D Hose Routing,” 2006 IEEE Int. Conf. Evol. Comput., pp. 48–55.

[186] Fernando, T. G. I., and Kalganova, T., 2012, “Multi-Colony Ant Systems for Multi-Hose Routing,” Int. J. Comput. Appl., **59**(2), pp. 1–14.

[187] Liu, Q., and Wang, C., 2010, “Pipe-assembly approach for aero-engines by modified particle swarm optimization,” Assem. Autom., **30**(4), pp. 365–377.

[188] LIU, Q. WANG, C., 2012, “Multi-terminal pipe routing by steiner minimal tree and particle swarm optimisation.,” Enterp. Inf. Syst., **6**(3), pp. 315–327.

[189] Liu, Q., and Wang, C., 2008, “A modified particle swarm optimizer for pipe route design,” Proc. 11th IEEE Int. Conf. Comput. Sci. Eng. CSE Work. 2008, pp. 157–161.

[190] Gong, D. W., Zhang, J. H., and Zhang, Y., 2011, “Multi-objective particle swarm optimization for robot path planning in environment with danger sources,” J. Comput., **6**(8), pp. 1554–1561.

[191] Redon, S., and Lin, M. C., 2005, “Practical local planning in the contact space,” Proc. - IEEE Int. Conf. Robot. Autom., **2005**(April), pp. 4200–4205.

[192] Barraquand, J., Kavraki, L., Latombe, J. C., Motwani, R., Li, T. Y., and Raghavan, P., 1997, “A random sampling scheme for path planning,” Int. J. Rob. Res., **16**(6), pp. 759–774.

[193] Ichter, B., and Pavone, M., 2019, “Robot Motion Planning in Learned Latent Spaces,” IEEE Robot. Autom. Lett., **4**(3), pp. 2407–2414.

[194] Yoshida, E., Esteves, C., Belousov, I., Laumond, J. P., Sakaguchi, T., and Yokoi, K., 2008, “Planning 3-D collision-free dynamic robotic motion through iterative reshaping,” IEEE Trans. Robot., **24**(5), pp. 1186–1198.

[195] Yang, K., and Sukkarieh, S., 2008, “3D smooth path planning for a UAV in cluttered natural environments,” 2008 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS, (October 2008), pp. 794–800.

[196] Hong, R., and DeSouza, G. N., 2010, “A real-time path planner for a smart wheelchair using harmonic potentials and a rubber band model,” IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc., pp. 3282–3287.

[197] Azzabi, A., and Nouri, K., 2019, “An advanced potential field method proposed for mobile robot path planning,” Trans. Inst. Meas. Control, **41**(11), pp. 3132–3144.

[198] Chen, X., and Zhang, J., 2013, “The three-dimension path planning of UAV based on improved artificial potential field in dynamic environment,” Proc. - 2013 5th Int. Conf. Intell. Human-Machine Syst. Cybern. IHMSC 2013, **2**, pp. 144–147.

[199] Masehian, E., and Naseri, A., 2010, “Mobile Robot Online Motion Planning Using Generalized Voronoi Graphs,” J. Ind. Eng., **5**, pp. 1–15.

[200] Liu, L., and Zhang, S., 2009, "Voronoi diagram and GIS-based 3D path planning," 2009 17th International Conference on Geoinformatics, Geoinformatics 2009, IEEE, Fairfax, VA, pp. 1–5.

[201] Pehlivanoglu, Y. V., 2012, "A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV," Aerosp. Sci. Technol., **16**(1), pp. 47–55.

[202] Wu, L., and Hori, Y., 2006, "Real-time collision-free path plannng for robot manipulator based on octree model," 9th IEEE International Workshop on Advanced Motion Control, Istanbul, pp. 284–288.

[203] Hamada, K., and Hori, Y., 1996, "Octree-based Approach to Real-time Collision-free Path Planning for R . obot Manipulator .," Proceedings of 4th IEEE International Workshop on Advanced Motion Control - AMC '96 - MIE, Japan, pp. 705–710.

[204] Zhang, G., and Jia, H., 2013, "3D path planning of AUV based on improved ant colony optimization," Chinese Control Conference, CCC, TCCT, CAA, pp. 5017–5022.

[205] Dubins, L. E., 1957, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," Am. J. Math., **79**(3), pp. 497–516.

[206] Hota, S., and Ghose, D., 2010, "Optimal geometrical path in 3D with curvature constraint," IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings, IEEE, Taipei, Taiwan, pp. 113–118.

[207] Shanmugavel, M., Tsourdos, A., White, B., and Zbikowski, R., 2010, "Co-operative path planning of multiple UAVs using Dubins paths with clothoid arcs," Control Eng. Pract., **18**(9), pp. 1084–1092.

[208] Hota, S., and Ghose, D., 2010, "Optimal path planning for an aerial vehicle in 3D space," Proceedings of the IEEE Conference on Decision and Control, IEEE, Atlanta, pp. 4902–4907.

[209] Kuwata, Y., and How, J., 2004, "Three dimensional receding horizon control for UAVs," AIAA Guidance, Navigation, and Control Conference and Exhibit, Providence, RI, pp. 5144–5157.

[210] Huang, S., and Teo, R. S. H., 2019, "Computationally efficient visibility graph-based generation of 3D shortest collision-free path among polyhedral obstacles for unmanned aerial vehicles," 2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019, IEEE, Atlanta, pp. 1218–

1223.

[211] Zeid, I., 2005, Mastering CAD/CAM, Mc Graw Hill.

[212] Owen, J., and Bloor, M. S., 1987, “Neutral formats for product data exchange: The current situation,” Comput. Des., **19**(8), pp. 436–443.

[213] Dibya Chakravorty, 2019, “The Most Common 3D File Formats” [Online]. Available: https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/.

[214] Fadel, G. M., Kirschman, C., and Kirschman, C., 1996, “Accuracy issues in CAD to RP translations,” Rapid Prototyp. J.

[215] “Multidimensional Arrays” [Online]. Available: https://www.mathworks.com/help/matlab/math/multidimensional-arrays.html#f1-87418.

[216] “struct” [Online]. Available: https://www.mathworks.com/help/matlab/ref/struct.html.

[217] “Linked List Data Structure” [Online]. Available: https://www.geeksforgeeks.org/data-structures/linked-list/.

[218] Micó, P., 2020, “stlTools (https://www.mathworks.com/matlabcentral/fileexchange/51200-stltools), MATLAB Central File Exchange. Retrieved March 27, 2020.”

[219] Möller, T., and Trumbore, B., 1997, “Fast, minimum storage ray/triangle intersection,” J. Graph. tools, **2**(1), pp. 21–28.

[220] Tuszynski, J., 2018, “Triangle/Ray Intersection (https://www.mathworks.com/matlabcentral/fileexchange/33073-triangle-ray-intersection), MATLAB Central File Exchange. Retrieved March 28, 2020.”

[221] Moller, T., 1997, “Fast triangle-triangle intersection test,” J. Graph. Tools, **2**(2), pp. 25–30.

[222] Tuszynski, J., 2020, “Surface Intersection (https://www.mathworks.com/matlabcentral/fileexchange/48613-surface-intersection), MATLAB Central File Exchange. Retrieved May 12, 2020.”

[223] Canny, J., 1987, “A new algebraic method for robot motion planning and real geometry,” 28th Annu. Symp. Found. Comput. Sci. (sfcs 1987).