Optimization Approaches for a Dubins Vehicle in Coverage Planning Problem and Traveling Salesman Problems

by

Xin Yu

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 10, 2015

Keywords: Coverage Path Planning, Traveling Salesman Problem, Dubins Vehicles, Combination Optimization

Approved by

John Y. Hung, Chair, Professor of Electrical and Computer Engineering
David M. Bevly, Professor of Mechanical Engineering
Thaddeus A. Roppel, Associate Professor of Electrical and Computer Engineering
Bogdan M. Wilamowski, Professor of Electrical and Computer Engineering

Abstract

The motivation of this dissertation is a path planning task for an autonomous robot-trailer system in geophysical surveys. The path planning task includes two main stages. In the first stage, an efficient coverage path is required to obtain a fully sensor coverage of a site to provide a complete map of anomalies. After the locations of anomalies are determined, in the second stage, an efficient traversal path is required to visit these anomalies to mark or obtain more data for further identification. The first stage can be regarded as the coverage path planning problem and the second stage can be regarded as a special case of traveling salesman problem. The robot-trailer system is modeled as a Dubins vehicle that can only move forward and turn with upper bounded curvature. Motivated by this autonomous inspection task, the author makes several contributions to the solution of coverage path planning problem and the solution of traveling salesman problems.

In the coverage path planning, the author presents an optimization approach that takes the vehicle's characteristics into account to minimize the non-working travel of the vehicle. Since turns are often costly for Dubins vehicle, minimizing the cost of turns usually produces more working efficiency. Prior researches on coverage path planning tend to fall into two complementary categories: (1) minimize the number of turns, by finding the optimal decomposition of a complex field into subfields and the optimal driving directions; (2) minimize the cost on a fixed number of turns, by finding the optimal visiting sequence of subfields and the optimal traversal sequence of parallel tracks for each subfield. This dissertation firstly presents a new algorithm to find the optimal decomposition that belongs to the first category; then designs a novel traversal pattern of parallel field tracks that belongs to the second category; finally extends the proposed traversal pattern to connect with the decomposition approach in the first category, providing a complete coverage path planning method

for the mobile robot. Experiments show that the proposed method can provide feasible solutions and the total wasted distance can be greatly reduced, when compared against classical boustrophedon path or recent state-of-the-art.

In the traveling salesman problems, given a set of waypoints and the turning constraint on the vehicle, the addressed problem is to determine a visiting sequence of these waypoints, and to assign a configuration of the vehicle at each waypoint. The objective function is to minimize the total distances traveled by the vehicle. A genetic algorithm is designed to find the shortest path and the performance is evaluated in numerical study. The proposed genetic algorithm can perform very well in both low waypoint density and high waypoint density situations. The author then takes the sensor scope into consideration to further minimize the total travel distance. The problem can be regarded as a special case of the Traveling Salesman Problem with Neighborhoods (TSPN). The concept of a neighborhood is used to model the physical size of the sensor scope. The neighborhoods are represented by disks in this dissertation. The author uses a two-step approach to solve the problem: (1) design a new algorithm for the TSPN to search the optimal visiting sequence and entry positions; (2) design a new algorithm for the Dubins vehicle to determine the heading at each entry position. The theoretical and numerical studies show that the proposed approach can perform very well for both disjoint and overlapped disks cases. The practical experiment shows that the model is feasible for the robot-trailer application.

While the authors focus on a robot-trailer system in this dissertation, the proposed algorithm could be applied to any Dubins vehicle that has similar mission requirements.

Acknowledgments

The author would like to express thanks to the members of his committee Dr. John Y. Hung, Dr. David M. Bevly, Dr. Thaddeus A. Roppel, Dr. Bogdan M. Wilamowski for their valuable assistance and guidance. The author also thanks the university reader Dr. Andrew J. Sinclair for his valuable suggestions on this dissertation.

Special thanks are given to Dr. John Y. Hung for the many hours of guidance and encouragement he has provided during this research. His suggestions have aided in the design of algorithms and experiments, and his advice has improved the visualization and written presentation of this work.

Thanks are also expressed to the Siwei Wang, Aditya Singh, Michael L. Payne and William J. Woodall for their collaboration and the wealth of background knowledge they have provided. Particular thanks go to David W. Hodo for his extensive previous work for the basis of this research and his invaluable support while performing the experiments.

This work would not have been possible without the funding and support provided by the Environmental Technology Certification Program (ESTCP) through the Army Corp of Engineers Huntsville Center.

Finally, the author dedicate this dissertation to his family and Zhongyuan Jia. None of this would be possible without their tremendous love and enthusiasm.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces a real-world problem that has motivated this dissertation: the inspection of unexploded ordnances (UXO) by an autonomous mobile robot. This challenge has inspired the development of new path planning techniques that achieve sensor coverage of complex 2D fields or a collection of waypoints. The problem statement in Section 1.1 is followed by a statement of contributions of this dissertation in Section 1.2.

1.1 Motivation and Problem Statement

Detection and clearing sites of unexploded ordnances (UXO) are generally labor-intensive, slow and expensive. In a report [5] of Department of Defense (DoD), it is estimated that in excess of 10 million acres of land on around 1400 sites of DoD may be affected by UXO. The cost would be tens of billions of dollars to detect and clear all of the possibly affected land. And the DoD are currently spending more than 200 million dollars per year on the UXO problems.

To map, locate, identify and select anomalies for sampling and removal within areas containing UXO, the process is typically done by conducting what is known as a geophysical survey. A geophysical survey provides a complete map of any detectable geophysical anomalies on a site. Different geophysical mapping sensors are used to detect metal or ferrous objects on or below the ground. Once these anomalies are located, they are either excavated by an explosives disposal team or more data is taken at their locations to attempt to determine if the anomaly is a piece of ordnance before excavating them. [6] Traditional techniques for geophysical surveys involve the use of hand-held detectors operated by UXO technicians who must walk across a survey area, as illustrated in Fig. 1.2a. It is not only time consuming,

but also exposes the operators to risk of serous injury. The use of an autonomous unmanned vehicle can not only reduce the risk to UXO technicians by providing the precise location of suspicious objects, but also relieve the operators of the tedious search process in large areas.

Auburn University has developed an autonomous tow vehicle, as illustrated in Fig. 1.2b. The goal of the project is to increase safety, productivity, and accuracy of the geophysical survey process by using autonomous vehicle technologies. The platform is capable of towing an array of industry standard geophysical mapping sensors in either tele-operated or semi-autonomous modes. It has been used to collect geophysical data with Geonics EM61-MK2 time domain metal detectors and Geometrics G858 magnetometers, but is capable of towing most any sensor package. The towing robot is a modified Segway® Robotic Mobility Platform (RMP) 440. Position information is provided to centimeter accuracy by a commercial integrated differential Global Positioning System (GPS) / Inertial Navigation System (INS) solution, the Novatel® SPAN system with the high precision Honeywell® HG1700 AG58 gyro. The location of geophysical mapping sensor is determined by geometric calculations based on tow bar hitch angles and the fixed tow bar lengths and alternately by a second GPS placed on the trailer. [6]

The central focus of this dissertation is to plan an efficient inspection path for geophysical surveys. The path planning task includes two main stages. In the first stage, an efficient coverage path is required to obtain a fully sensor coverage of a site to provide a complete map of UXO. After the locations of anomalies are determined, in the second stage, an efficient traversal path is required to visit these anomalies to mark or obtain more data for further identification. It is assumed a priori knowledge of the environment to be surveyed. Motivated by the autonomous UXO inspection task, the author contributes several new algorithms to the solution of coverage path planning problem and the solution of traveling salesman problems in this dissertation.

(a)

(b)

Figure 1.1: Unexploded ordnance. (a) Munitions Debris located during surface sweep and excavated anomalies. (b) An 81mm mortar. Image courtesy of ECC. Source: `http://www.earthexplorer.com/2009-07/uxo_lands_restoration_and_release.asp`.

1.2 Organization and Contributions of the Dissertation

The rest of this dissertation is organized as follows: In Chapter 2, the algorithms and applications of coverage path planning problem and traveling salesman problems are reviewed,

(a)

(b)

Figure 1.2: (a) Geophysical survey operated by an UXO technician. Image courtesy of David W. Hodo. Source: `http://www.auburn.edu/~hododav/projects/segway_project/DSCN3821.JPG`. (b) An autonomous robot-trailer system for geophysical survey. The towing robot is a modified Segway® RMP 440.

as well as some unresolved issues of these two path planning problems. The contributions of coverage path planning are mainly discussed in Chapter 3 and Chapter 4. The contributions of traveling salesman problems are mainly discussed in Chapter 5 and Chapter 6.

In Chapter 3, the author presents an optimization approach to minimize the number of turns of autonomous vehicles in coverage path planning. For complex polygonal fields, the problem is reduced to finding the optimal decomposition of the original field into simple subfields. The optimization criterion is minimization of the sum of widths of these decomposed subfields. A new algorithm is designed based on a multiple sweep line decomposition. The time complexity of the proposed algorithm is $O(n^2 \log n)$. Experiments show that the proposed algorithm can provide nearly optimal solutions very efficiently when compared against recent state-of-the-art. The proposed algorithm can be applied for both convex and non-convex fields. The work on this topic is also drafted in a paper for a conference.

In Chapter 4, the author presents an optimization approach that takes the vehicle's characteristics into account to minimize the non-working travel of the robots in coverage path planning. The aim is to minimize the cost on a fixed number of turns, by finding the optimal traversal sequence of parallel tracks for the surveyed field. The author firstly presents a novel traversal pattern of parallel tracks for a single convex field, then extends the proposed traversal pattern to connect with decomposition algorithms, providing a complete coverage path planning method for non-convex fields. Experiments show that the proposed method can provide feasible solutions and the total wasted distance can be greatly reduced for both single convex field and multiple decomposed fields, when compared against classical boustrophedon path or recent state-of-the-art. The work on this topic is also drafted in a paper for a journal.

In Chapter 5, the author studies the traveling salesman problems. Taken the vehicle's characteristics into account, the problem is modeled as a Traveling Salesman Problem for Dubins vehicles. A genetic algorithm is designed to find the shortest path and the performance is evaluated in numerical study. The experiments show that the proposed algorithm

can perform better than the well-known Alternating Algorithm and Random Headings Algorithm, in both low waypoint density and high waypoint density situations. The work on this topic is also documented in [7].

In Chapter 6, the author takes the physical size of the actual sensors into consideration when planning a path for the waypoints visiting problem. The trailer equipped with geophysical mapping sensors traverses among a collection of waypoint neighborhoods. The concept of a neighborhood is used to model the size of sensor scope. The problem is modeled as a Dubins Traveling Salesman Problem with Neighborhoods (DTSPN), where the neighborhoods are represented by disks. The authors firstly design a new algorithm for the Traveling Salesman Problem with Neighborhoods (TSPN), then extend this algorithm to find the shortest path for the DTSPN. The experiments show that the proposed algorithm can perform very well for both disjoint and overlapped disks cases. The work on this topic is also documented in [8] and drafted in a paper for a journal.

In Chapter 7, the author concludes the key results and discusses promising areas for future work.

Chapter 2

Relevant Literatures

Path planning is one of the fundamental problems in robotics. The most general case involves a robot finding a trajectory from one state to another automatically, while avoiding collisions with obstacles. Approaches for path planning include exact roadmap methods, such as cell decompositions [9,10], visibility graphs [11–13] and Voronoi diagrams [14,15]; sampling based methods [16], such as probabilistic roadmap method [17] and rapidly exploring random tree [18]. Methods such as evolutionary algorithms [19–22], neural networks [23], potential field methods [24] and optimal control theory [25] have also been applied for industrial applications in recent literatures.

2.1 Coverage Path Planning

Coverage path planning is a special type of path planning, which requires the robot to determine a path that passes all points of an area. It is a common challenge in many industrial applications, and extensively studied in recent years. Examples include demining robots [3], autonomous lawn mowers [26], indoor service robots [27–31], exploration robots [32–35], autonomous underwater vehicles [36] and automated harvesters [37]. Several solutions to the coverage path planning have been reviewed and categorized in surveys [38, 39]. The surveys show that most existing algorithms adopt cellular decomposition of the given field to achieve the provable guarantee of complete coverage. A cellular decomposition finds efficient ways to subdivide the given field into cells that can be easily traversed by a coverage path.

Based on the main method they use, cellular decompositions can be subdivided into three types: approximate, semi-approximate and exact. Approximate cellular decomposition approximates the target field by using cells of same size and shape. Semi-approximate

decomposition uses cells with fixed width to approximate the target field, but the top and bottom of cells can have any shape. Exact cellular decomposition uses a set of non-intersecting cells without size and shape constraints, and the union of cells exactly fills the target field.

In this dissertation, only exact cellular decomposition is considered. The exact cellular decomposition algorithms usually include three procedures: (1) decomposition of the complex coverage field into subfields with parallel tracks; (2) selection of a traversal sequence of those subfields; and (3) generation of a boustrophedon path (straight parallel paths with alternate directions) that covers each subfield individually.

One popular exact cellular decomposition technique is the trapezoidal decomposition [40], in which the free space is decomposed into trapezoidal cells. Since each cell is a trapezoid, coverage in each cell can be easily achieved with the boustrophedon path. Coverage of the field is achieved by visiting each cell in the adjacency graph. The shortcoming of this method is that it requires too many redundant turns to guarantee complete coverage. Since turns are often costly and considered as non-working time, minimizing the cost of turns usually produces higher working efficiency. Two main categories of solution strategies are recently studied to reduce the cost of turns, which can be seen as improvements for procedure (1) and procedure (3) separately.

2.1.1 Optimal Decomposition and Track Layout

The first category of solution strategy is to find the optimal decomposition of a given field and the optimal layout of parallel tracks in each subfield.

Choset and Pignon [41] develop a boustrophedon decomposition to reduce the redundant turns. In this method, a line segment, termed a slice, is swept through the environment. Whenever there is a change in connectivity of the slice, a new cell is formed. When the connectivity increases, two new cells are spawned. Conversely, when connectivity decreases, two cells are merged into one cell. The tracks in each cell are parallel to the slice.

Huang [1] introduces a decomposition algorithm to minimize the total number of turns required to cover a field. The algorithm adopts multiple line sweeps to divide the coverage field into cells, then combines cells into larger subfields by dynamic programming, and finally assigns each subfield a sweep direction according to the minimum sum of altitudes. The parallel tracks in each subfield are perpendicular to the sweep direction of that subfield.

Oksanen and Visala [37] propose an algorithm that incrementally decomposes the field into subfields using trapezoidal decomposition, merges small subfields into larger ones, then searches for the merged subfield with the best cost and removes it from the original field. In searching for the subfield with best cost, the layout of parallel tracks is determined by a heuristic approach. The process is repeated for the remaining field until the whole field is computed.

Fang and Anstee [42] propose an iterative decomposition scheme based on the generalized Voronoi diagram [43]. They firstly compute an approximate generalized Voronoi diagram of the given field, then apply boustrophedon decomposition along the longest line segment of the approximate generalized Voronoi diagram, select the subfields that contain the longest line segment, and remove these subfields with well planned path. The algorithm is repeated for the remaining field until the whole field is fully covered.

Jin and Tang [44] adopt a divide-and-conquer strategy to the decomposition. The algorithm firstly searches the optimal layout of parallel tracks without any decomposition, then finds "all possible ways" to splitting the field into two and for each possible way sees if the field would be more efficient as two subfields instead of one. The algorithm is implemented recursively on each subfield until there is no valid decomposition that achieves a better solution.

Li et al. [2] propose a decomposition algorithm to minimize the number of turns based on a greedy recursive method. The process recursively decomposes the field into two subfields until no concave region remains. In each decomposition step, the criterion of optimization is

the minimum width sum of two subfields. The final tracks in each subfield are perpendicular to the width of that subfield. The algorithm is proven to have polynomial time complexity.

2.1.2 Optimal Traversal Sequence

The second category of solution strategy is toward computing optimal traversal sequence for the parallel field tracks, instead of using boustrophedon path.

Rankin et al. [45] propose a set pattern to determine the traversal sequence of field tracks, by taking into account the minimum turning radius of the vehicle to skip the adjacent tracks. Hodo et al. [3] extend this pattern in the application of Dubins vehicle and also consider avoidance of known obstacles on the straight tracks. Bochtis and Vougioukas [46] propose a method to model this problem as a Traveling Salesman Problem (TSP). They treat each field track as a node in the TSP. The travel cost between tracks is related to the degree of a maneuver. After the cost matrix is constructed, the problem can be solved by the existing TSP solvers. Bochtis et al. [4] further extend this problem to cases with capacity constraint, and model it as Vehicle Routing Problem (VRP). The VRP is a generalization of the TSP, and reduces to a TSP when the vehicle number is one and capacity is infinite. In their work, each field track is represented by two nodes, one for each endpoint of that track. Each node corresponds to a "customer" in the VRP. In order to ensure the track be covered, the cost between nodes in the same track is set to be zero and the connection between two nodes is avoided if they represent endpoints of different tracks at opposite side. After the cost matrix is constructed, the problem can be solved by existing VRP solvers.

2.1.3 Some Unresolved Issues

Upon deeper investigation, key issues still remain in finding the optimal traversal sequence. In [46], the travel costs from different endpoints of one track to the corresponding endpoints of the other track are assumed to be the same. But in many applications, the travel cost from one track to another may vary, as illustrated in Fig. 2.1a. On the other

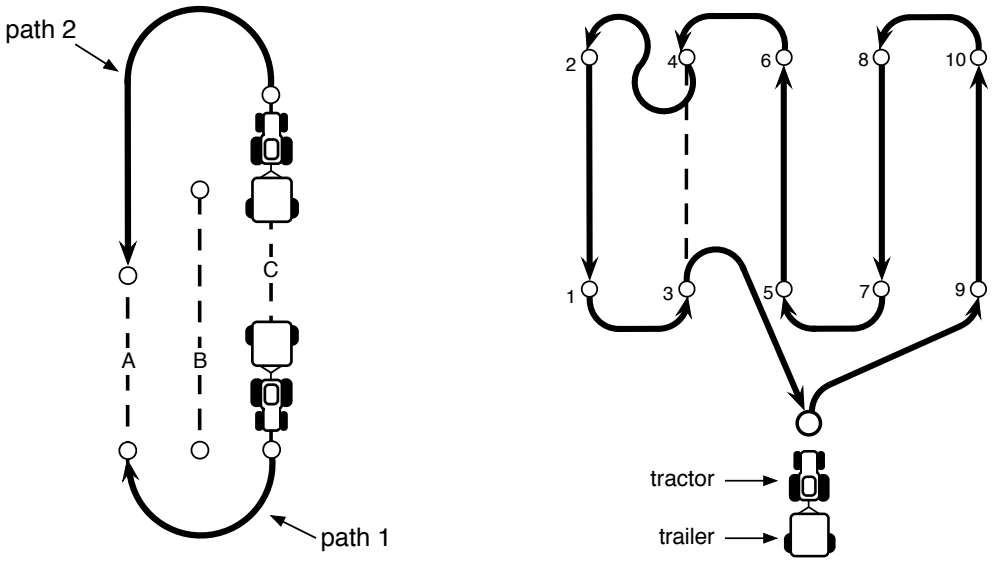


Figure 2.1: Remaining issues in finding optimal traversal sequence: (a) the non-working travel distances from track C to track A are different between path 1 and path 2, (b) optimal traversal of endpoints may skip a track (3-4).

hand, when dealing with some applications that require the vehicle to return to the starting point after traversal, the optimal solution may not be always feasible by using the method of representing the tracks with two endpoints, and traversing all endpoints to ensure covering each track. For an instance with odd number of tracks, the optimal path to traverse all endpoints may skip tracks and fail to cover the field entirely, as illustrated in Fig. 2.1b. Among the many methods of finding the optimal decomposition and track layout, the default path to traverse tracks in each subfield is the boustrophedon path, which can be improved by other kind of motions.

2.2 Traveling Salesman Problems

2.2.1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is one of the best known and most studied optimization problems. [47] In TSP, given a set of points, the task is to determine the shortest tour to visit each point only once and return to the starting point. In most cases,

the distance between two points in the TSP network is the same in both directions. When the distance between two points are not the same from different directions, the problem is called Asymmetric Traveling Salesman Problem (ATSP). If the distance between any two points is determined by Euclidean distance, it is called the Euclidean Traveling Salesman Problem (ETSP). [48] ETSP has been applied to many robot path planning problems.

2.2.2 Dubins Traveling Salesman Problem

However, when working with a car-like robot, researchers have to consider kinematic constraints such as minimum turning radius, i.e. the ETSP result may not provide an optimal solution. Typically, the car-like robot that can only move forward at a constant speed, with a minimum turning radius can be modeled as a Dubins vehicle [49]. The traveling salesman problem for the Dubins vehicles is usually called Dubins Traveling Salesman Problem (DTSP). Previous researches about DTSP may belong to two different categories, based on the main methods they used.

Category (1): Use existing TSP methods or other ordering methods to calculate the optimal visiting order of the given waypoints, and then design different algorithms to determine the heading of each waypoint based on that visiting order. Savla et al. [50] provide an Alternating Algorithm (AA) that connects the optimal ordered waypoints by straight lines, after which the odd-numbered edges along with respective headings are retained; the even-numbered edges are replaced by Dubins paths. Other researchers such as Ma and Castanon [51] firstly extend the two points Dubins path to three successive points Dubins path, then connect the ordered waypoints by the three points Dubins path, and use receding horizon theory to optimize the result. Tang et al. [52] design another algorithm by modifying the gradient descent method to determine the headings and offer some intuitive suggestions to improve the result. Medeiros and Urrutia [53] adopt an Angular-metric TSP [54] method to minimize the sum of direction changes in determining the visiting order, then design an algorithm based on heading discretizaiton and Dijkstra's Algorithm [55] to determine the

heading at each waypoint. Macharet et al. [56] also adopt the Angular-metric TSP method to determine the visiting order, then propose an improvement for the Alternating Algorithm to obtain the headings, finally apply a Greedy Randomized Adaptive Search Procedure (GRASP) [57] to further optimize the result.

Category (2): Determine heading for each waypoint first, and then transform DTSP into ATSP; finally use existing ATSP methods to solve the problem. Le Ny et al. [58] design a Randomized Headings Algorithm (RHA) in which they first assign each waypoint a random heading, and then calculate the distance between each pair of waypoints with Dubins path. In the following step, they use these distances to transform the DTSP to an ATSP. An improved version of this algorithm based on heading discretization [59] assigns a fixed number of discrete headings for each waypoint, and then treats the problem as a Generalized Traveling Salesman Problem (GTSP), finally transforms the GTSP into ATSP.

2.2.3 Traveling Salesman Problem with Neighborhoods

A generalization of ETSP is the Traveling Salesman Problem with Neighborhoods (TSPN). In TSPN, given a collection of n regions in the plane, called neighborhoods, the task is to find a shortest tour that visits all neighborhoods. The researchers need to determine not only the visiting sequence of the neighborhoods, but also the entry points. Many researchers have addressed TSPN with various neighborhoods [60] [61]. Specially, for the case that the neighborhoods are represented by disks, Dumitrescu and Mitchell [62] provide a Polynomial Time Approximation Scheme (PTAS) for disjoint unit disks. de Berg et al. [63] provide a constant factor algorithm for disjoint convex fat neighborhoods of varying size, where the approximation factor is $12,000\alpha^3$ ($\alpha = 4$ for disks). Elbassioni et al. [64] improve this approximation factor to $(9.1\alpha + 1)$ by an approximation algorithm for disjoint α fat objects with possibly varying size ($\alpha = 4$ for disks). Later, Yuan et al. [65] provide an approach that firstly obtains the visiting sequence of the disjoint disks by an external TSP algorithm, then adopts Evolutionary Algorithm to search the entry points. Recently, He et

al. [66] propose a Combine-Skip-Substitute (CSS) scheme for both disjoint and overlapped disks cases.

In the case when the neighborhoods are disconnected vertex sets, then the problem is called Generalized Traveling Salesman Problem (GTSP), also known as the Set TSP, Group TSP, One-of-a-Set TSP. The tour is required to visit at least one point from each set. The GTSP can be solved by many algorithms. Among them include exact algorithms [67, 68], heuristic algorithms [69, 70], and methods of transforming GTSP into ATSP [71–73].

2.2.4 Dubins Traveling Salesman Problem with Neighborhoods

The Dubins Traveling Salesman Problem with Neighborhoods (DTSPN) can be seen as a combination of the well known TSPN and DTSP. When the turning radius is zero, DTSPN reduces into the TSPN case; when the neighborhood size is zero, then the DTSPN reduces into the DTSP case. Since both TSPN and DTSP are NP-hard problems, the DTSPN is also NP-hard.

Obermeyer [74] firstly addresses the DTSPN by using a genetic algorithm, then in [75] designs a sampling based algorithm to transform the DTSPN into a Generalized Traveling Salesman Problem (GTSP) and then into an Asymmetric Traveling Salesman Problem (ATSP) via the Noon and Bean transformation [71]. Isaacs et al. [76] further improve the sampling based algorithm by adopting a more general version of Noon and Bean transformation [71].

2.2.5 Some Unresolved Issues

Although each algorithm has its advantages, there are also disadvantages revealed by deeper investigation. For the TSPN, the previous approximation algorithms mostly deal with the disjoint disks case and have large approximation factors. Although the CSS scheme can perform very well in both disjoint and overlapped disks cases, which is the best result so far, there is still room for improvement in the disjoint disks case. For the DTSP, the *Category*

(1) algorithms can perform very well when waypoints are spaced far apart, but may perform worse when the distances between waypoints are very small relative to the turning radius. *Category (2)* algorithms can perform well when the distances between waypoints are small relative to the turning radius, but need much more computing effort. For the DTSPN, the sampling based algorithms may obtain better solutions when the number of samples are very large. However, it significantly increases the total number of nodes that need to be solved for the ATSP. For large scale instances, the ATSP solver may perform worse and the computing time will increase significantly, as the total number of nodes increases.

Chapter 3

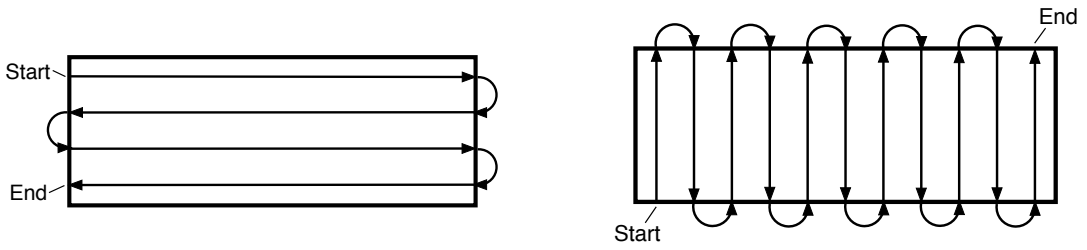
Coverage Path Planning: Optimal Decomposition and Track Layout

3.1 Introduction

Coverage path planning determines a path that guides a robot to pass every part of a workspace completely and efficiently. Since turns are often costly for autonomous vehicles, minimizing the cost of turns usually produces more working efficiency. In this chapter, the authors propose a polynomial time algorithm to minimize the number of turns in coverage path planning, and the time complexity is greatly improved comparing to the existing algorithms. The remainder of this chapter is organized as follows. In Section 3.2, the problem of coverage path planning is transformed into width calculation of the coverage field and the problem statement is formally introduced. A linear time algorithm for convex fields is described in Section 3.3. A polynomial time algorithm for non-convex fields is designed in Section 3.4. In Section 3.5, the proposed algorithm is compared with existing algorithms and a practical experiment with real field data is also conducted. Section 3.6 summarizes the key results in this chapter.

3.2 Problem Statement

The goal of this dissertation is to find a path to completely cover a field by a vehicle and the travel distance of the vehicle must be minimized. Since boustrophedon method is the most common method to cover a simple field, this chapter will adopt parallel tracks that can be used by boustrophedon method (or other similar methods) to cover the field. Based on this assumption, the travel distance of the vehicle consists of the distance along tracks and the distance to turn at the end of tracks.



(a) (b)

Figure 3.1: Different track directions for convex fields. [1]

Before giving further problem statement, the authors firstly introduce the concept of altitude and width of convex polygons as follows:

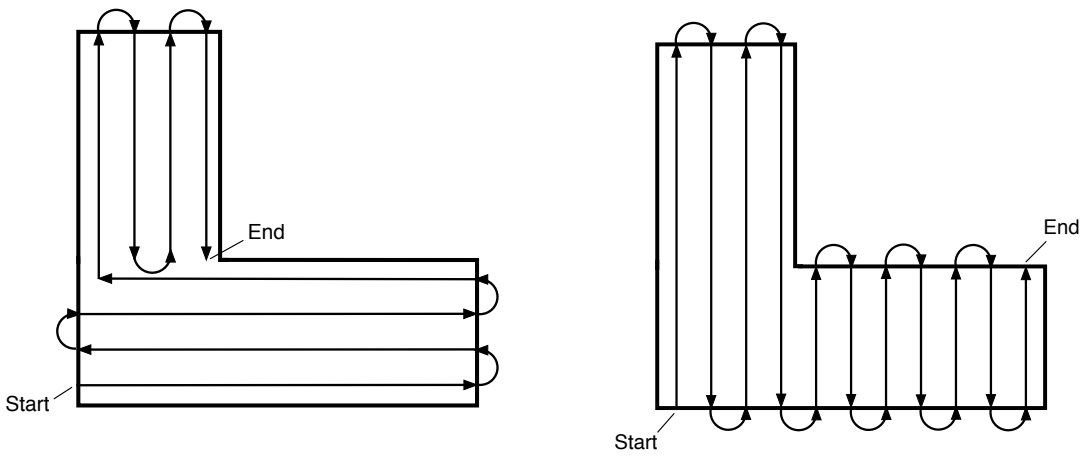
Definition 3.1. *Given a convex polygons P, a line of support L is a line intersecting P and such that the interior of P lies to one side of L.*

Definition 3.2. *The altitude A of a convex polygon P is the shortest distance between a pair of parallel lines of support (L_1, L_2).*

Definition 3.3. *The width W of a convex polygon is the minimum altitude of that polygon.*

If the field is convex and does not contain any obstacles, the coverage planning with parallel tracks is quite simple. The main task is to find the optimal direction of the parallel tracks. By deeper analysis, the problem can be further reduced. As illustrated in Fig. 3.1, covering the field in different directions can produce nearly the same distance along the tracks, but can produce a large difference in number of turns, which means a large difference in distance on turns. Therefore, the total travel distance mainly depends on the number of turns, i.e. the total distance will decrease as the number of turns decreasing. Furthermore, the number of turns in a given track direction is also proportional to the altitude of the convex polygonal field in that direction. Therefore, the problem can be reduced to search the minimum altitude (width) of the field and its corresponding direction. The parallel tracks can be generated along the vertical direction of width.

If the field is non-convex (concave or with obstacles), finding the optimal solution is hard. One possible strategy is to decompose the complex field into convex subfields. Each



(a) (b)

Figure 3.2: Different track directions for non-convex fields. [2]

subfield can be covered by parallel tracks in a different direction. Fig. 3.2 shows an example that assigns each subfield a different track direction results in a better solution than applying only one track direction to the whole fields. Furthermore, the minimum number of turns in each subfield can be determined by the width of the subfield. To obtain an optimal solution for the non-convex field, the total sum of widths must be minimized.

The coverage problem becomes to find a convex decomposition of the non-convex field that has the minimum sum of widths. The authors refer such a decomposition as the Minimum Sum of Widths (MSW) Decomposition. Let $\mathcal{P}$ be the polygon that represents the non-convex field with n vertices. In a convex decomposition $\mathcal{D}$, the polygon $\mathcal{P}$ is decomposed into m $(m \leq n)$ convex polygons $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, whose widths are $\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_m$ respectively. Let $S(\mathcal{D})$ be the sum of width of $\mathcal{D}$. Then the problem can be stated more formally:

Problem 3.4 (MSW Decomposition)**.**

$$\underset{\mathcal{D}}{\textit{minimize}} \quad S(\mathcal{D}) = \sum_{i=1}^{m} \mathcal{W}_i$$

$$\textit{subject to} \quad \mathcal{P}_i \in \textit{convex polygon}$$

Note that the optimal coverage of a convex field can be seen as a degenerate of the above problem statement where $m = 1$.

3.3 Coverage of Convex field

As described in the previous section, the optimal coverage of a convex field can be determined by the width of that polygon. However, a convex polygon admits parallel lines of support in any direction, and for each direction the altitude is usually different. Fortunately, not all directions need to be examined to determine the width. Suppose a convex polygon is given, along with two parallel lines of support. If neither of these lines coincides with an edge, it is always possible to rotate them to decrease the distance between them. Therefore, the width of polygon can be determined by examining only the edge orientations. A formal proof of this result can be found in [1].

Based on this result, a linear time complexity algorithm is given in [77]. The algorithm adopts rotating calipers, which is a method used to construct efficient algorithms for a number of computational geometry problems. The method is analogous to a vernier caliper that rotates around the outside of a convex polygon. Every time one blade of the caliper lies flat against an edge of the polygon, it forms an antipodal pair with the point or edge touching the opposite blade. The complete rotation of the caliper around the polygon detects all antipodal pairs and can be carried out in $O(n)$ time. The process is described in Algorithm 1. The unit of angle is radian.

After the width is calculated, the minimum number of turns N_{turn} can be determined by

$$N_{turn} = \lceil \frac{W}{d} \rceil \tag{3.1}$$

where W is width of the given convex polygon, d is the space between two adjacent tracks and ceiling symbol $\lceil * \rceil$ is the smallest integer not less than $*$. [2] Then the N_{turn} parallel tracks can be generated in the direction *Angle* that is returned by the algorithm.

Algorithm 1 Width of a Convex Polygon

Input: Vertex list of the given convex polygon in counterclockwise order
Output: Width of the polygon and direction of the corresponding parallel lines of support

Delete middle vertices of any collinear sequence of three vertices
$V_a \leftarrow$Vertex with minimum y-coordinate
$V_b \leftarrow$Vertex with maximum y-coordinate
$RotatedAngle \leftarrow 0$
$Angle \leftarrow 0$
$Width \leftarrow \infty$
$Caliper_a \leftarrow$Unit vector along positive x-axis
$Caliper_b \leftarrow$Unit vector along negative x-axis
while $RotatedAngle < \pi$ **do**
    $E_a \leftarrow$Edge from V_a to its next adjacent vertex
    $E_b \leftarrow$Edge from V_b to its next adjacent vertex
    $A_a \leftarrow$Angle between $Caliper_a$ and E_a
    $A_b \leftarrow$Angle between $Caliper_b$ and E_b
    $Altitude \leftarrow 0$
    if $A_a < A_b$ **then**
        Rotate $Caliper_a$ by A_a
        Rotate $Caliper_b$ by A_a
        $V_a \leftarrow$The next adjacent vertex of V_a
        $Altitude \leftarrow$Distance from $vertex_b$ to $Caliper_a$
        $RotatedAngle \leftarrow RotatedAngle + A_a$
    else
        Rotate $Caliper_a$ by A_b
        Rotate $Caliper_b$ by A_b
        $V_b \leftarrow$The next adjacent vertex of $vertex_b$
        $Altitude \leftarrow$Distance from $vertex_a$ to $caliper_b$
        $RotatedAngle \leftarrow RotatedAngle + A_b$
    end if
    if $Altitude < Width$ **then**
        $Width \leftarrow Altitude$
        $Angle \leftarrow RotatedAngle$
    end if
end while
return $Width$ and $Angle$

3.4 Coverage of Non-convex field

To obtain the coverage of a non-convex field, the author uses a strategy based on a multiple sweep line decomposition. Firstly a new convex decomposition method is designed based on the sweep line method in one sweeping direction (3.4.1) and the optimal coverage for

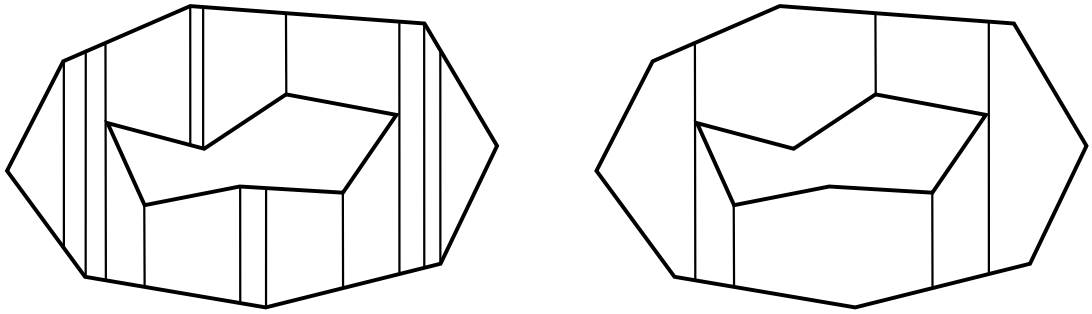
each convex subfield is searched (3.4.2). Then, for each edge orientation (including edges of the polygon and edges of the obstacle), the authors apply the designed convex decomposition and obtain the sum of widths. After that, the convex decomposition with minimum sum of widths is selected (3.4.3). To avoid unnecessary tracks to cover the subfields, the resulting decomposition are finally refined by merging adjacent similar convex polygons (3.4.4).

3.4.1 Convex Decomposition

The proposed convex decomposition method is an enhancement of the trapezoidal decomposition [40] and is designed to reduce unnecessary cells. As illustrated in Fig. 3.3a, the trapezoidal decomposition comprises cells that are shaped like trapezoids or triangles (which can be seen as degenerate trapezoids). To improve time complexity, trapezoidal decomposition adopts a sweep line method and treats each vertex as an event. To form the decomposition, a vertical line is swept from left to right through the polygon field. When an event is encountered, it extends rays upward and downward through the free space of the polygon field until an edge that lies immediately above and below the event is hit. Many events will have either just an upward ray or a downward ray. Trapezoidal cells are formed at the event depending on the event type. Once the sweep line finishes the rightmost event, a trapezoidal decomposition results. However, the drawback of trapezoidal decomposition is that it produces too many redundant convex cells. Some adjacent small convex cells can be merged into a larger convex cells in the sweep line process, as shown in Fig. 3.3b. What follows is an improvement of trapezoidal decomposition that reduces the redundant convex cells.

Events

The following definitions of events are based on the assumption that vertices of the polygon are listed in counter-clockwise order and vertices of holes (obstacles) are listed in clockwise order. Then the interior of the polygon is always to the left of each edge when



(a) (b)

Figure 3.3: (a) Trapezoidal decomposition. (b) The proposed convex decomposition (3.4.1).

following the order. The sweep line is perpendicular to the x-axis and horizontally swept from left to right.

In trapezoidal decomposition, all vertices are classified into five types of events: *OPEN, CLOSE, SPLIT, MERGE* and *INFLECTION*. These types of events are defined as follows:

Definition 3.5. *A vertex v is an OPEN event if its two neighbor vertices lie on the right side of the sweep line and the interior angle at v is less than π; if the interior angle is greater than π, then v is a SPLIT event. A vertex is a CLOSE event if its two neighbor vertices lie on the left side of the sweep line and the interior angle at v is less than π; if the interior angle is greater than π, then v is a MERGE event. A vertex is an INFLECTION event if its two neighbor vertices lie on opposite sides of the sweep line.*

In the proposed convex decomposition, the *INFLECTION* event is replaced with four more types of events: *FLOOR_CONVEX*, *FLOOR_CONCAVE*, *CEIL_CONVEX*, *CEIL _CONCAVE* which are defined as follows:

Definition 3.6. *A vertex v is a FLOOR_CONVEX event if its previous neighbor vertex v_{prev} lies on left side of the sweep line while its next neighbor v_{next} lies on right side of the sweep line, and the interior angle at v is less than π; if the interior angle is greater than π, then v is a FLOOR_CONCAVE event. On the contrary, a vertex v is a CEIL_CONVEX event if its previous neighbor vertex v_{prev} lies on right side of the sweep line while its next neighbor v_{next}*

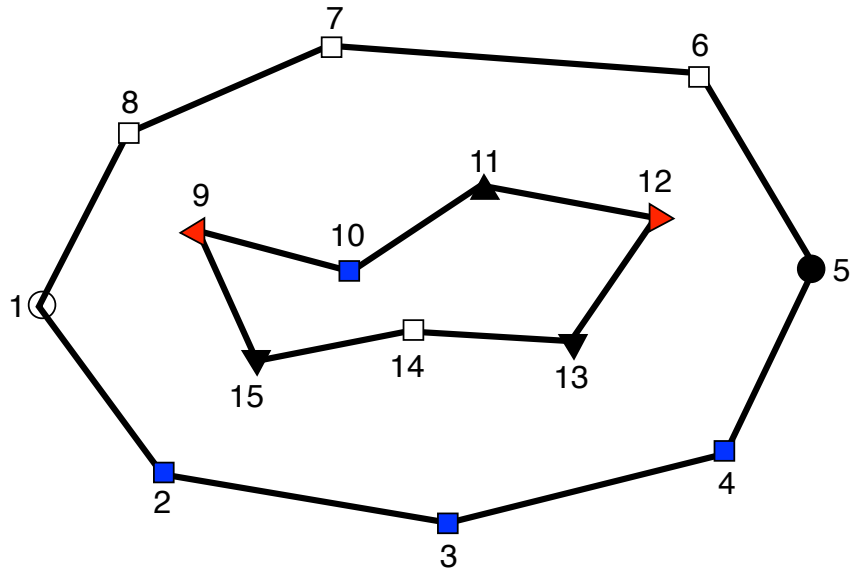


Figure 3.4: Eight event types: *OPEN* (1), *CLOSE* (5), *SPLIT* (9), *MERGE* (12), *FLOOR_CONVEX* (2, 3, 4, 10), *FLOOR_CONCAVE* (11), *CEIL_CONVEX* (6, 7, 8, 14) and *CEIL_CONCAVE* (13, 15). The sweep line is horizontally swept from left to right.

lies on left side of the sweep line, and the interior angle at v is less than π; if the interior angle is greater than π, then v is a CEIL_CONCAVE event.

Examples of eight event types are illustrated in Fig. 3.4.

Sweep Line Algorithm

Next, the sweep line algorithm is applied to form the decomposition. The events are firstly sorted based on their x-coordinates in an ascending order. During the sweeping process, a balanced binary search tree L is used to maintain the "current" edges that the sweep line intersects. A cell in the algorithm can be represented by two lists: ceiling list and floor list, both of which bound the cell. The sweep line algorithm starts from left to right, and visits each event in order. When the sweep line encounters an event, different operations are made depending on the type of event:

OPEN event: Two incident edges of this event are inserted into L. A new cell is opened and the vertex of this event is added to floor list of the cell.

SPLIT event: The edges immediately above and below this event are searched in L. Then the intersection of the sweep line and the above edge, and the intersection of the sweep line and the below edge are determined. Find the cell to which this event belongs. Add the above and below intersection points into ceiling list and floor list of the cell respectively. Now the current cell is considered to be closed. After that, two new cells are opened. Add vertex of this event into floor list of the top new cell and ceiling list of the bottom new cell respectively. Add the above intersection point into ceiling list of the top new cell and the below intersection point into floor list of the bottom new cell. After the cell operations, two incident edges of this event are inserted into L.

FLOOR_CONVEX event: Find the cell to which this event belongs. Add the vertex of this event into floor list of the current cell. Delete the left incident edge of this event from L and insert the right incident edge of this event into L.

CEIL_CONVEX event: Find the cell to which this event belongs. Add the vertex of this event into ceiling list of the current cell. Delete the left incident edge of this event from L and insert the right incident edge of this event into L.

FLOOR_CONCAVE event: Delete the left incident edge of this event from L. Search the edge that is immediately above this event in L, and determine the intersection point of the sweep line and the above edge. Then add the right incident edge of this event into L. Find the cell to which this event belongs. Add the vertex of this event into floor list of the current cell and the intersection point into ceiling list of the current cell respectively. Now the current cell is considered to be closed. After that, a new cell is opened. Add the vertex of this event into floor list of the new cell and the intersection point into ceiling list of the new cell respectively.

CEIL_CONCAVE event: Delete the left incident edge of this event from L. Search the edge that is immediately below this event in L, and determine the intersection point of the sweep line and the below edge. Then add the right incident edge of this event into L. Find the cell to which this event belongs. Add the vertex of this event into ceiling list of the

current cell and the intersection point into floor list of the current cell respectively. Now the current cell is considered to be closed. After that, a new cell is opened. Add the vertex of this event into ceiling list of the new cell and the intersection point into floor list of the new cell respectively.

MERGE event: Two incident edges of this event are deleted from L. The edges immediately above and below this event are searched in L. Then the intersection of the sweep line and the above edge, and the intersection of the sweep line and the below edge are determined. Find the two cells to which this event belongs. Add the vertex of this event into floor list of the top cell and ceiling list of the bottom cell respectively. Add the above intersection point into ceiling list of the top cell and the below intersection point into floor list of the bottom cell respectively. Now the two cells are considered to be closed. After that, a new cell is opened. Add the above intersection point into ceiling list of the new cell and the below intersection point into floor list of the new cell respectively.

CLOSE event: Two incident edges of this event are deleted from L. Find the cell to which this event belongs. The vertex of this event is added to floor list of the current cell and the current cell is considered to be closed.

After all events are visited, the polygonal field is decomposed into a list of convex cells. The adjacency graph of these cells can also be determined in the process. The main difference between the proposed convex decomposition and trapezoidal decomposition is at the *FLOOR_CONVEX* and *CEIL_CONVEX* events. At these two events, the proposed decomposition doesn't open or close a cell, but rather just updates the current cell. Note that the above description of sweep line algorithm assumes that the x-coordinates of all events are distinct. For general cases, the assumption can be achieved by rotating the coordinate system in a sufficiently small amount.

3.4.2 Optimal Coverage for Each Convex Polygon

In one sweeping direction, the field is decomposed into convex sub-polygons by applying the proposed convex decomposition. The width of each sub-polygon can then be determined independently by applying the method described in Section 3.3. Also, the optimal orientation of parallel tracks in each sub-polygon can be determined independently. Then the sum of width of these polygons can be calculated.

3.4.3 Sweep Direction

Until this section, the convex decomposition is done in a particular sweeping direction. However, the best sweep direction is not known and has to be searched. In [1], the author shows that the best sweep direction is perpendicular to one of the boundary or obstacle edges, if all tracks are perpendicular to the sweep direction. In this dissertation, the author also assumes that the best sweep direction is perpendicular to one of these edges. So only sweep directions that are perpendicular to the boundary or obstacle edges are examined. For each sweep direction, the field is decomposed into convex sub-polygons and the width of each sub-polygon can be determined independently. Among all the possible sweep directions, the convex decomposition with minimum sum of width is selected. By now, the minimum sum of widths decomposition is done.

3.4.4 Merging Adjacent Polygons

Since each sub-polygon is covered independently, it may produce redundant tracks to cover two adjacent sub-polygons, when parallel tracks of these two adjacent sub-polygons have the same track orientation [41]. To avoid the redundant tracks, two convex sub-polygons are merged if they have the same track orientation and are entirely adjacent to each other [2]. The definition of adjacency and entire adjacency of two polygons are described as follows:

Definition 3.7 ([2]). *Consider two polygons P_1 with n vertices and P_2 with m vertices. If an edge of P_1, $v_{1i}v_{1(i+1)}\,(i \in [1,n])$, coincides with an edge of P_2, $v_{2j}v_{2(j+1)}\,(j \in [1,m])$, P_1 is adjacent to P_2.*

Definition 3.8 ([2]). *P_1 and P_2 are adjacent to each other, whose coincidence edges are $v_{1i}v_{1(i+1)}$ and $v_{2j}v_{2(j+1)}$ respectively. If $v_{1i} = v_{2j}$ (or $v_{1i} = v_{2(j+1)}$) and $v_{1(i+1)} = v_{2(j+1)}$ (or $v_{1(i+1)} = v_{2j}$), P_1 is entirely adjacent to P_2.*

With the decomposition and adjacency graph, the planner determines a merge process into four steps. First, assign each sub-polygon an unique group number. Secondly, iterate all these sub-polygons. For each sub-polygon, test it with all its neighbors to see whether these two polygons satisfy the merging condition. If the neighbor sub-polygon satisfies, change the group number of the neighbor sub-polygon to that of the current sub-polygon. Thirdly, sort the sub-polygons according to their group numbers. Finally, iterate the ordered sub-polygons and merge the sub-polygons that have the same group number.

Then parallel tracks can be generated in each sub-polygon according to their track orientations.

3.4.5 Time Complexity Analysis

Let n be the number of vertices of the input polygonal field. In the proposed convex decomposition (3.4.1), the first step is to sort the events based on the x-coordinates. This takes $O(n \log n)$ time. Then in the sweep line process, for each event, determining an edge that is immediately above or below the event takes $O(\log n)$ time if the edges are stored in a balanced binary search tree. So for all events, the sweep line process takes $O(n \log n)$ time. Since each sub-polygon takes linear time to calculate the width and the total number of vertices of these sub-polygons is linear to n, it takes $O(n)$ time to determine the sum of width of these sub-polygons (3.4.2). As long as the sweep line process takes $O(n \log n)$ in one sweeping direction, examining sweep directions that perpendicular to all edges of the polygon takes $O(n^2 \log n)$ time (3.4.3). In the merging process (3.4.4), the number of sub-polygons

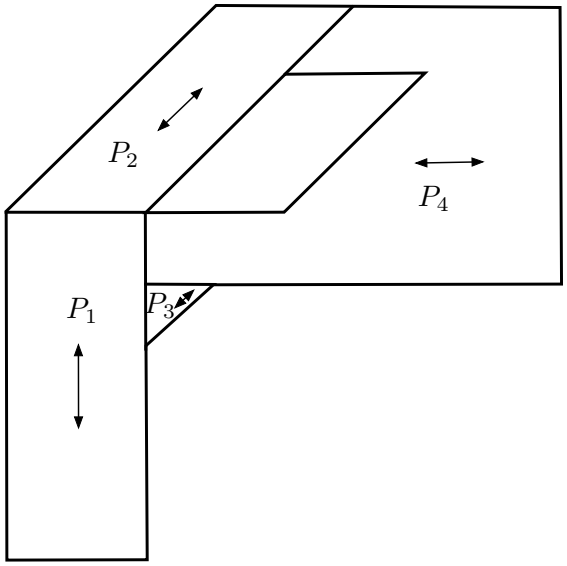
and the number of edges in the adjacency graph are both linear to the number of vertices n. The first step takes $O(n)$ time. In the second step, examining the entire adjacency of two polygons can be taken in $O(n \log n)$. To test the entire adjacency for all sub-polygons, edges in the adjacency graph will be visited twice which is linear to n. So the second step requires $O(n^2 \log n)$. Sorting in the third step requires $O(n \log n)$ time. In the last step, merging of two sub-polygons takes $O(n \log n)$ time. To merge all sub-polygons that have the same group number, it requires $O(n^2 \log n)$ time. So the whole merging process takes $O(n^2 \log n)$ time.

Therefore, the total time complexity of the proposed algorithm in this chapter is $O(n^2 \log n)$.

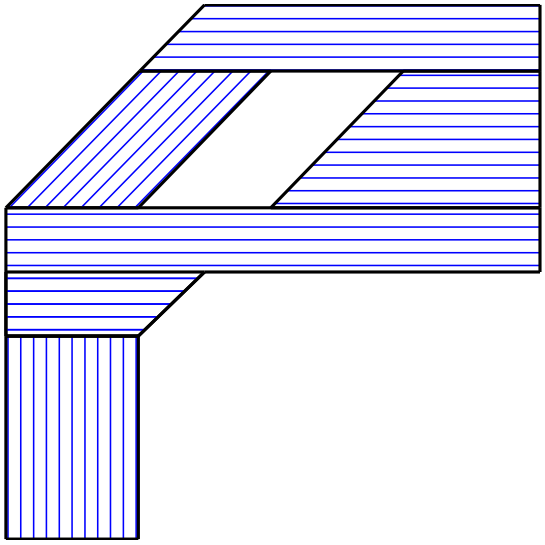
3.5 Test Results

The algorithm is implemented in C++ and tested on a computer with 1.3 GHz CPU and 4 GB RAM. To test the performance, the result of the proposed algorithm is compared with two former researchers' results. Both former algorithms use sum of widths as cost function to search the optimal decomposition. Fig. 3.5a is an example given by Huang [1], who adopts dynamic programming technique to search the optimal decomposition. Since the dynamic programming in [1] searches all possible decompositions, the solution in Fig. 3.5a can be seen optimal in such case. Fig. 3.5b shows the solution generated by the proposed algorithm. By comparing the total width of these two solutions, the total width of the proposed algorithm is 3.5% greater than that of the optimal solution. Note that the method in Huang's work [1] needs exponential time to obtain the optimal solution. However, the proposed algorithm requires only $O(n^2 \log n)$ time, which is much faster to obtain the nearly optimal result. The computational time for the proposed solution is 0.10 second as an average of 10 running times.

Fig. 3.6a is an example given by Li et al. [2], who design a greedy recursive method for the decomposition. Fig. 3.6b shows the solution generated by the proposed algorithm. The total width of the proposed algorithm is 3.6% less than that of the greedy recursive method



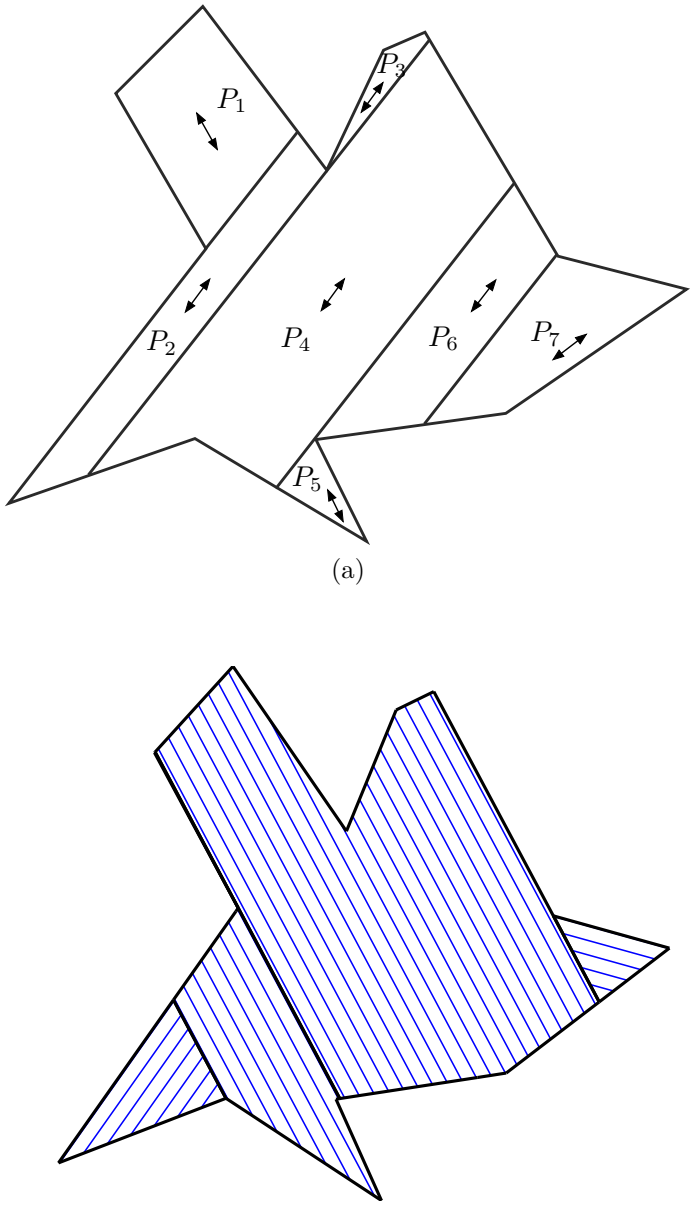
(a)



(b)

Figure 3.5: (a) Solution of Huang's algorithm [1]. Arrows indicate the track directions. (b) Solution of the proposed algorithm.

in [2]. Also the proposed algorithm requires less time to obtain the result, since the greedy recursive algorithm in [2] requires $O(n^4)$ time. The computational time for the proposed solution is 0.09 second as an average of 10 running times.



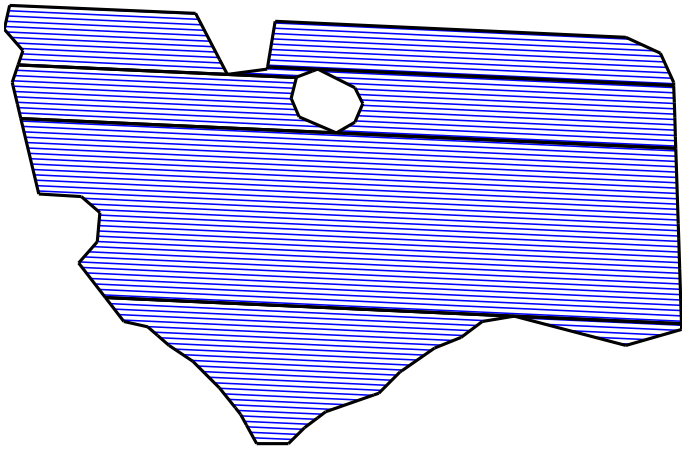
(a)

(b)

Figure 3.6: (a) Solution of Li's algorithm [2]. Arrows indicate the track directions. (b) Solution of the proposed algorithm.

Another experiment field [32°35′29.5″N 85°29′31.3″W], shown in Fig. 3.7a, has an area of 6.65 acres (or 26896.73 m^2) and has one obstacle area. The solution of the proposed algorithm is shown in Fig. 3.7b. The space between two adjacent parallel tracks is set to 2 meters. The solution shows that the optimal sweeping direction is 87.40° and the field is

(a)



(b)

Figure 3.7: Test field near Auburn University and solution of the proposed algorithm.

decomposed into 8 sub-polygons. The computational time for the proposed solution is 0.25 second as an average of 10 running times.

3.6 Summary

In this chapter, the authors try to minimize the number of turns in coverage path planning. The problem is reduced to find the optimal decomposition of a complex field into convex subfields. The criterion of optimization is the sum of width of these decomposed subfields. Firstly, a new convex decomposition algorithm in one sweeping direction is designed based on sweep line method. Then the convex decomposition is performed in all directions that are perpendicular to the edges, and the convex decomposition with minimum sum of width is selected. To avoid unnecessary tracks to cover the subfields, the resulting decomposition is finally refined by merging adjacent similar convex sub-polygons. The time complexity of this algorithm is $O(n^2 \log n)$. The proposed algorithm is compared with two state-of-the-art algorithms. The results show that the proposed algorithm can produce a nearly optimal solution with much greater efficiency. Another experiment, based on real field data, shows that the proposed algorithm can produce feasible and effective solution for a large area field containing obstacles.

Chapter 4

Coverage Path Planning: Optimal Visiting Sequence

4.1 Introduction

In this chapter, the authors propose a novel traversal pattern in finding optimal traversal sequences of parallel tracks. The problem is modeled as a Generalized Traveling Salesman Problem (GTSP), which can be transformed into a standard Asymmetric Traveling Salesman Problem (ATSP). The method can also be extended to connect the field tracks decomposed by the algorithm in Chapter 3. In searching the optimal traversal sequence, the proposed pattern can simultaneously determine the traversal sequence of tracks for each subfield and the visiting sequence of subfields, such that the total non-working distance will be minimized. The remainder of this chapter is organized as follows. In Section 4.2, the vehicle model based on the Dubins vehicle is reviewed. In Section 4.3, an algorithm is designed to find the optimal traversal sequence of tracks in a single convex field, and Section 4.4 extends the method to connect multiple subfields. Several experiments are presented in Section 4.5 to show the performance of the proposed algorithm. Section 6.7 summarizes the key results.

4.2 Vehicle Model

The autonomous vehicle can be modeled as the Dubins vehicle. A Dubins vehicle is one that can only move forward at constant speed and turn with upper bounded curvature (or lower bounded turning radius) in the plane. The configuration of Dubins vehicles can be described by (x, y, θ), where (x, y) defines the position of the vehicle in the plane, and θ

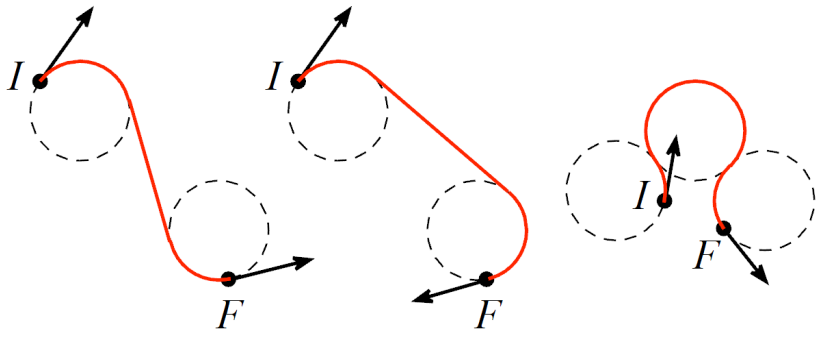


Figure 4.1: Example Dubins Paths [3]

defines the heading of the vehicle. The vehicle dynamic is described by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{\rho}u \end{bmatrix}, |u| \leq 1 \tag{4.1}$$

where v is the speed of the vehicle, ρ is the minimum turning radius and u is the control input.

Dubins' [49] work characterizes the optimal paths between any two configurations of such vehicle. The main results show that the optimal path is contained in a finite number of Dubins paths. This reduces the problem to determine the shortest path between any two configurations by examining only six path types, divided into two families:

- Family CCC: types RLR, LRL
- Family CSC: types LSL, RSR, RSL, LSR

where C is the "curve" or arc segment of radius ρ, and S is the "straight" line segment. When a subgraph is a C-segment, it could be either a left turn or right turn, denoted by L and R respectively. One method for generating the optimal Dubins path and computing

the Dubins distance between two configurations can be found in [78]. Several examples of Dubins paths between initial configuration I and final configuration F are shown in Fig. 4.1.

4.3 Optimization on a single convex field

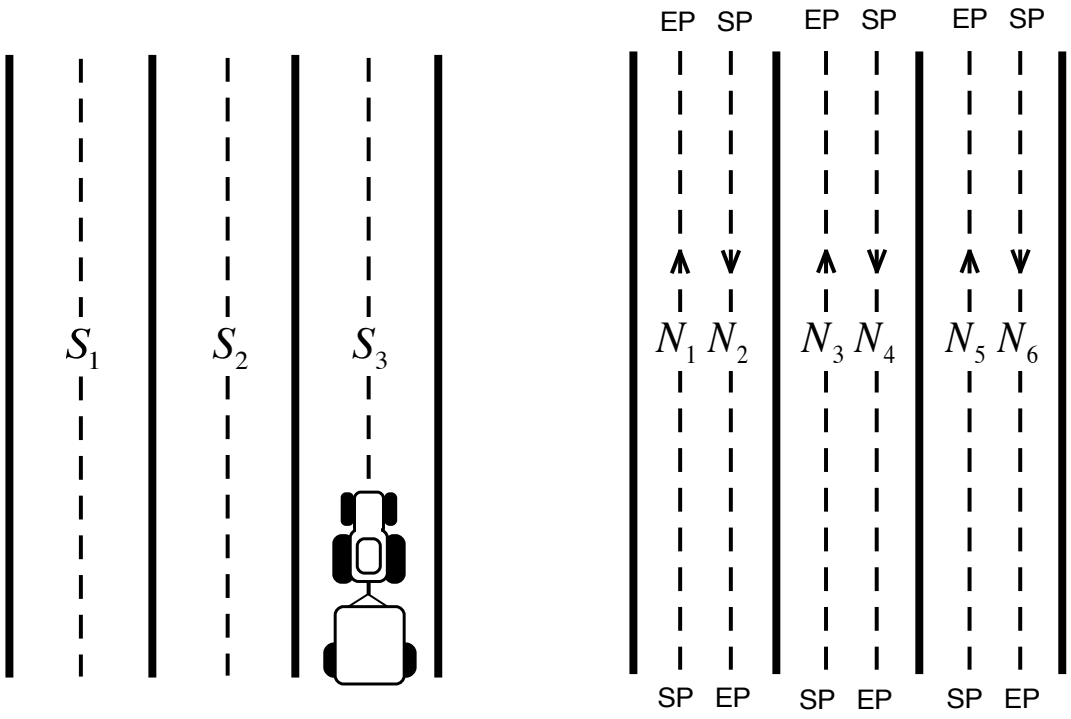
In this section, the optimal traversal sequence of tracks in a single field is presented. The given field is of convex shape and covered by a set of parallel tracks, which start at one boundary of the field and end at a different boundary. It is assumed that the track locations are predetermined and all tracks are parallel. The task is to find the optimal traversal sequence of these field tracks.

4.3.1 Algorithm

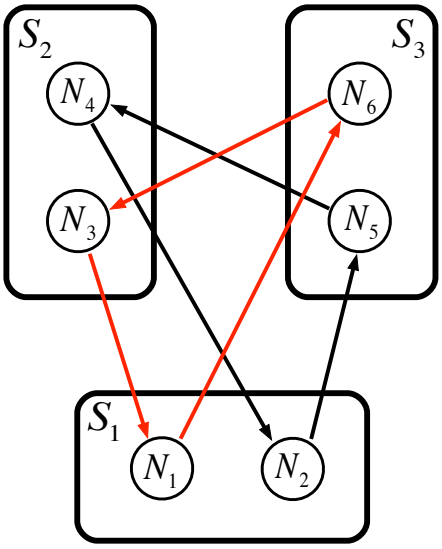
The main idea of the proposed method is to model the problem as a Generalized Traveling Salesman Problem (GTSP). Then the GTSP is transformed into a standard ATSP through the Noon and Bean transformation [71]. Finally a variety of existing ATSP solvers can be applied to find the optimal solution. The GTSP is defined on a directed graph $G = (N, A)$, where N is a node set partitioned into m clusters $S_1, S_2, \ldots, S_m$ and $A = \{(N_i, N_j) : N_i, N_j \in N, i \neq j\}$ is an arc set. For each arc $(N_i, N_j) \in A$, a cost c_{ij} is defined as the distance from N_i to N_j. The matrix $C = (c_{ij})_{n \times n}$ is called the cost matrix, where n is the number of nodes. The objective of GTSP is to find a minimum cost cycle, which includes exactly one node from each cluster.

4.3.2 Nodes

According to the above definition of GTSP, let $S = \{S_1, S_2, S_3, \ldots, S_m\}$ be the arbitrarily ordered set of field tracks. Each field track consists of two endpoints and one line segment. The vehicle can cover the field track from either endpoint to the other one, so there are two directed path options for each track. The algorithm treats each track as a cluster of two nodes. Each node represents a directed path of the track on which the vehicle



(a) (b)



(c)

Figure 4.2: GTSP node representation: (a) A given set of parallel field tracks (dashed lines) (b) Each track has two directed path options (dashed lines, SP: starting point, EP: ending point) (c) Corresponding GTSP node representation and two feasible GTSP solutions (in gray and in black)

may travel. Two directed paths in the same cluster are parallel to the corresponding actual track, but have opposite directions. The node set of the corresponding GTSP can be written as $N = \{N_1, N_2, N_3, \ldots, N_{2m}\}$. The node set is partitioned into m clusters and each cluster contains exactly two nodes. One example of the node representation is illustrated in Fig. 4.2. In order to generate Dubins paths, a set of configurations are required. For each node, the starting point (SP) is the point where the vehicle enters the track, and the ending point (EP) is the point where the vehicle leaves the track. The headings of the Dubins vehicle at the starting point and ending point are equal to the direction of the corresponding directed path. From the node representation, every node contains two configurations: SP configuration and EP configuration.

4.3.3 Cost Between Nodes

Two nodes in the same cluster should never be connected together, because the vehicle should not traverse the same track in two different directions. Therefore, the cost between nodes in the same cluster is set to be M where M is a large positive number. Nodes belonging to different clusters are connected by Dubins path. The path is always computed from the EP configuration of one node to the SP configuration of the other node. Let $\Delta(N_i) \in S, N_i \in N$ denote the cluster where node i belongs and $D(N_i, N_j)$ denote the Dubins distance from node N_i to node N_j. The cost c_{ij} from node N_i to node N_j can be expressed as follows:

$$c_{ij} = \begin{cases} D(N_i, N_j), & if\ \Delta(N_i) \neq \Delta(N_j) \\ M, & if\ \Delta(N_i) = \Delta(N_j) \end{cases} \tag{4.2}$$

The Dubins distances $D(N_i, N_j)$ and $D(N_j, N_i)$ may be different. After the costs being computed for each pair of nodes, the cost matrix for GTSP is constructed. The cost matrix is actually described by the non-working distance, which means the formation of GTSP is to minimize the total non-working distance travelled during turnings. By the above cost expression, there is no constraint that successive ending point and starting point must lie on

the same boundary or the same side of the tracks, which is different from existing methods. This assumption has no effect on the feasibility of the solution, but gives the algorithm more freedom to find the optimal solution.

4.3.4 Depot Considerations

In some applications, the traversal of field tracks starts and ends at a depot outside the working field, such as a barn in agriculture or parking area for vehicles. The travel distance between the depot and field tracks can also be seen as non-working distance, and thus is necessarily considered. A depot can be modeled as one cluster with only one node in the GTSP. The starting point and ending point of the depot node coincide. The coordinate of starting point and ending point is the same as the position of the depot. The starting direction and ending direction of Dubins vehicle at the depot can be determined arbitrarily, e.g. align to the direction of the entry road. The cost from depot node to track node is computed by the Dubins distance from EP configuration of the depot node to SP configuration of the track node. Conversely, the cost from any track node to depot node is computed from EP configuration of track node to SP configuration of depot node. These costs are included in the cost matrix.

In other applications, there is no depot, but the starting location and ending location are separated. In this dissertation, it is assumed that the starting location and ending location are endpoints of different tracks. Two conditions need to be considered: (1) If the starting location and ending location are not specified, the task is to find a least cost path to traverse all field tracks without returning to the start. In such case, a virtual depot node is added to the problem, by setting the cost between depot node and any track node to be zero. Therefore there is no effect of depot position on the optimal solution for the traversal sequence. The paths connecting the depot can be then removed from the final result tour. (2) If the starting location and ending location are specified, the starting track node and ending track node can also be determined. In such case, a virtual depot node with only two

arcs is added to the problem. One arc is from the depot node to the starting track node, and the other is from the ending track node to the depot node. To represent them in the cost matrix, the cost from the depot node to the starting track node and cost from the ending track node to the depot node are set to be zero, while other costs that relate to the depot node are set to be M where M is a large positive number. The paths connecting the depot can be then removed from the final result tour.

4.3.5 Transformation from GTSP into ATSP

After the cost matrix being constructed, the GTSP can be solved by many algorithms. Among them include exact algorithms [67], heuristic algorithms [70], and methods of transforming GTSP into ATSP [71]. In this chapter, the author adopts the Noon and Bean transformation [71]. There are two reasons to use this transformation. First, there exist many efficient ATSP solvers. By using this transformation, the state-of-the-art ATSP solvers are directly available to use. Second, the number of nodes in ATSP is equal to that of the original GTSP, which will not increase the computing complexity for ATSP solvers, while other transformation methods have larger number of nodes for ATSP than that of the original GTSP.

A brief summary of Noon and Bean transformation is described as follows. The transformation consists of two stages. In the first stage, the GTSP is transformed into a Clustered TSP. The nodes in each cluster are firstly given an arbitrary order. Then a single directed cycle is created for each cluster, by adding intra-cluster arcs with zero cost according to the given order. The inter-cluster arcs are then circularly shifted for each cluster so that they emanate from the previous node in the cycle. In the second stage, the Clustered TSP is transformed into a standard ATSP by adding a large cost to all inter-cluster arc costs. Finally the GTSP solution can be extracted from the ATSP solution by taking only the first node visited in each cluster. An illustration of the transformation can be seen in Fig. 4.3. Readers interested in more details should study the reference [71].

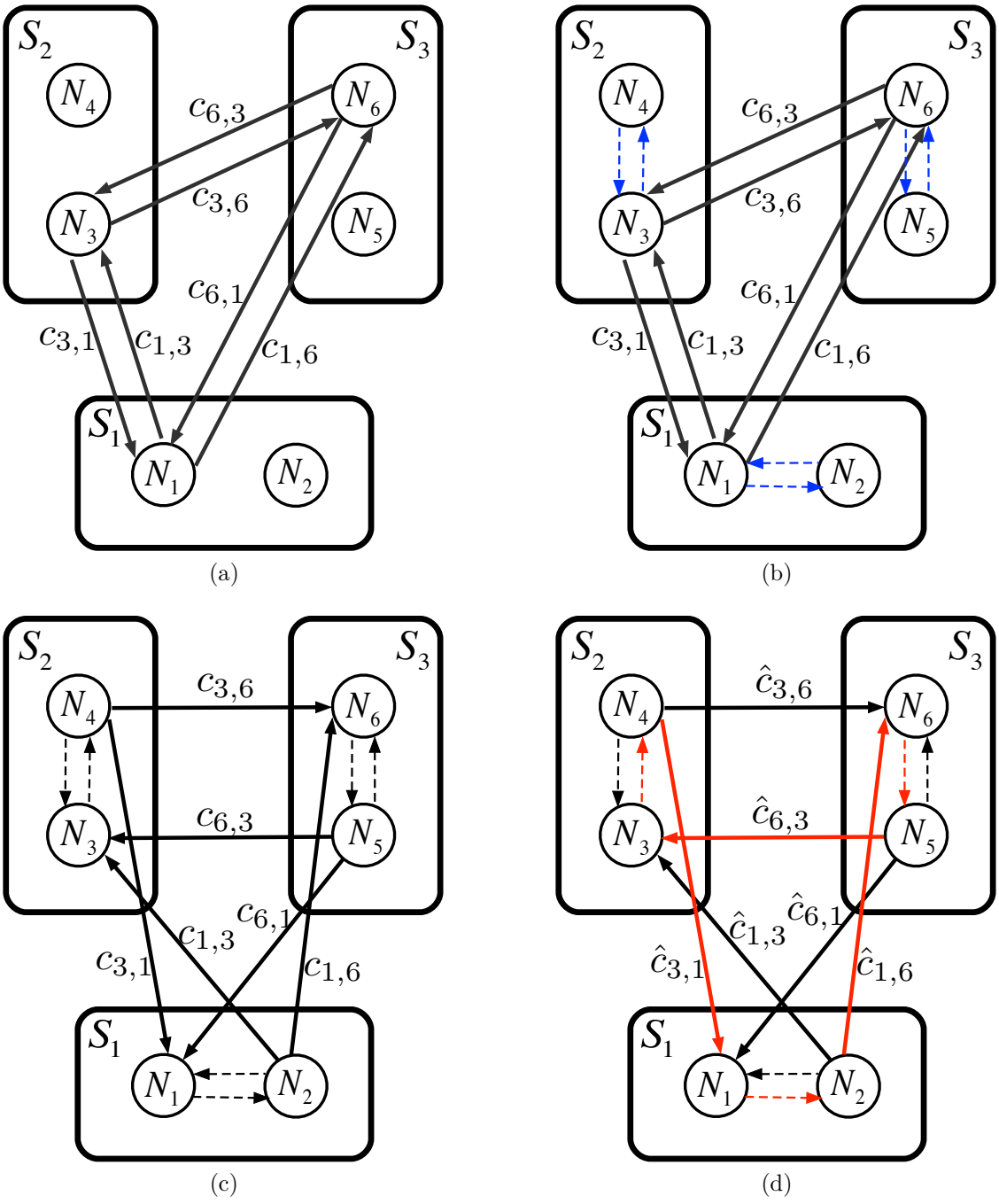


Figure 4.3: Illustration of transformation from GTSP into ATSP: (a) A GTSP representation with arc costs for the example in Fig. 4.2. Note that only an essential subset of arcs is shown for clarity of illustration. (b) A zero-cost directed cycle is created for each cluster by adding zero-cost arcs between consecutive nodes in each cluster. (The dash arcs in blue have zero cost.) (c) The inter-cluster arcs are circularly shifted so they emanate from the previous node in its cycle. (d) A large finite cost β is added to each inter-cluster arc. Here $\hat{c}_{i,j} = c_{i,j} + \beta$, where $+\infty > \beta > \sum_{(i,j)\in A} c_{i,j}$. The optimal ATSP tour is shown in red with a cost of $\hat{c}_{1,6} + \hat{c}_{6,3} + \hat{c}_{3,1}$. The GTSP solution can be extracted from the ATSP solution by taking only the first node visited in each cluster.

4.3.6 Complexity of the Proposed Algorithm

Since each cluster contains at most two nodes, for n clusters, this algorithm will compute the ATSP over at most $2n$ nodes. The worst case computational complexity of the Noon and Bean transformation [71] is $O(n^2)$. Then the worst case computational complexity for solving the ATSP by using the modified version of Christofides' algorithm provided in [79] is $O(n^3)$. So overall the computational complexity of the proposed algorithm is $O(n^3)$.

4.4 Extension to multiple fields

For case where the field is complex, e.g. of concave shape, the field is usually decomposed into multiple simple subfields. By any decomposition algorithm described in the first category of section I.A, the complex field is divided into simple subfields and the layout of parallel tracks is determined in each subfield. Then methods can be applied to find the optimal traversal sequence of these field tracks. Based on different situations, two strategies are shown in this section.

In the first situation, it is assumed that each pair of subfields can be connected. The proposed method in previous section can be extended to minimize the total non-working travel distance of these decomposed subfields. Tracks belongs to different subfields are put together and treated equally as in the same field. In this sense, the cost matrix is constructed for all tracks, rather than tracks belong to individual subfield. The final solution is the optimal traversal sequence of all the tracks.

In the second situation, it is assumed that some connections between tracks or subfields are forbidden. Since most decomposition algorithms consider obstacles in the decomposition process, the result subfields are typically obstacle free. Directly applying the above strategy for all tracks places some constraints on the cost matrix, because some connections between tracks or subfields are not permitted if they intersect with the obstacles or restricted areas. If two tracks are disconnected, the costs between corresponding track nodes are set to be M where M is a large positive number. If two subfields are disconnected, tracks belong to

different subfields are not connected. Among these two subfields, the costs between track nodes that belong to different subfields are all set to be M where M is a large positive number.

4.5 Four Experiments

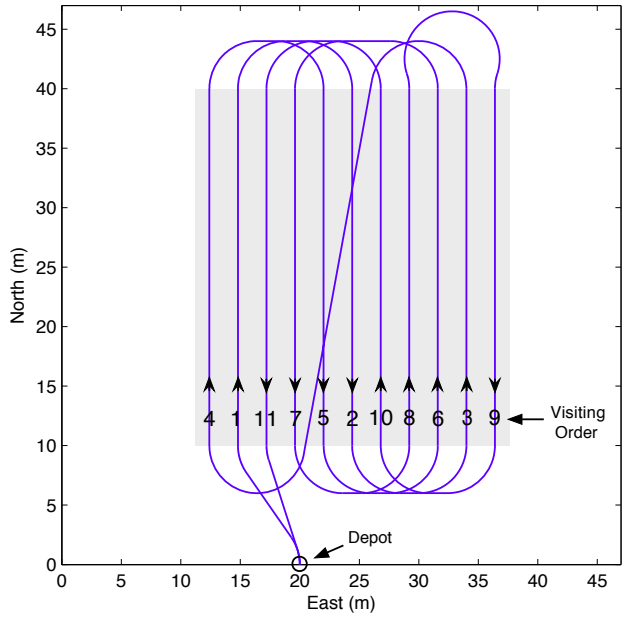
In this section, four experiments are conducted to examine the performance of the proposed method. The minimum turning radius of the test robot is 4 m, and the line spacing between adjacent tracks is 2.4 m. The optimal ATSP tour is obtained using the LKH solver [80], which is an effective implementation of the Lin-Kernighan heuristic for solving a Traveling Salesman Problem. It is widely accepted that LKH will efficiently produce an exact result in scales up to hundreds or even thousands of nodes [80].

Let the track set S be numbered from west to east, then $S = \{1, 2, 3, \ldots, n\}$ where n is the number of tracks. Let $\Pi^*(S)$ be the visiting sequence of S by using a certain pattern, where elements in the sequence are track numbers according to the visiting order.

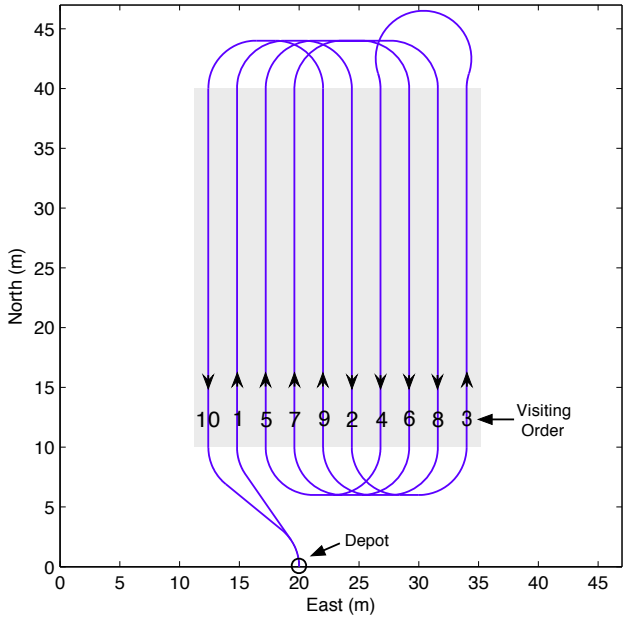
4.5.1 Effect of Parity (Even or Odd Number of Tracks with One Depot)

The first experiment is established to test the effect of parity of the number of tracks, when dealing with applications with one depot. Two rectangular fields are used, which have dimensions of 24 m $\times$ 30 m, 26.4 m $\times$ 30 m respectively (gray shaded areas). The depot positions are both $(20, 0)$. In order to completely cover each field, the required number of field tracks are 10 and 11 respectively.

Fig. 4.4a shows the GTSP pattern for odd number of tracks and the visiting order of each track. The track set is $S = \{1, 2, 3, \ldots, 11\}$, and the visiting sequence is $\Pi^{gtsp}(S) = \langle 2, 6, 10, 1, 5, 9, 4, 8, 11, 7, 3\rangle$. Fig. 4.4b shows GTSP pattern for even number of tracks and the visiting order of each track. The track set is $S = \{1, 2, 3, \ldots, 10\}$, and the visiting sequence is $\Pi^{gtsp}(S) = \langle 2, 6, 10, 7, 3, 8, 4, 9, 5, 1\rangle$. The result of B pattern [4] is also shown in Fig. 4.5a to compare with the proposed GTSP pattern in such case. The results



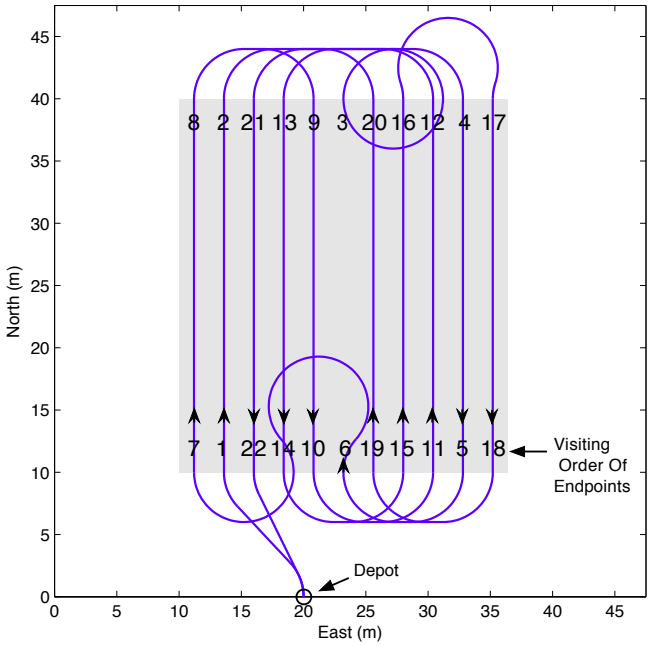
(a)



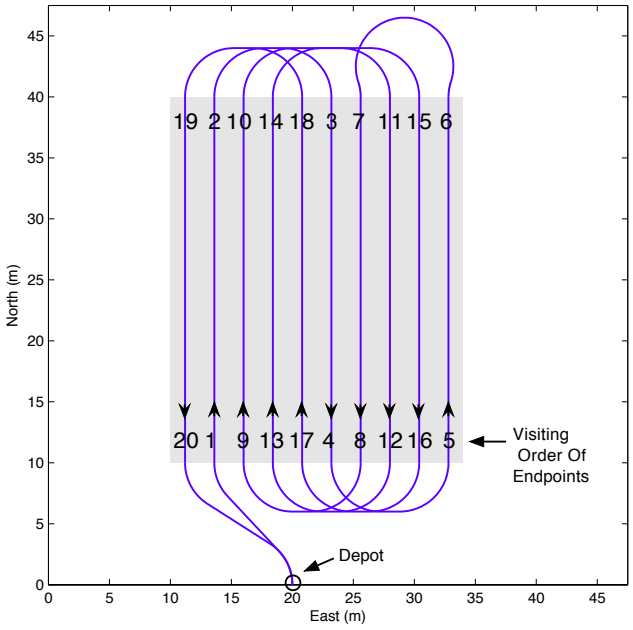
(b)

Figure 4.4: (a) GTSP pattern for odd number of tracks (11 tracks) with one depot. (b) GTSP pattern for even number of tracks (10 tracks) with one depot. Shaded area is field that must be covered. The number on each track is the visiting order of that track. Arrows indicate the driving direction on each track. (Experiment 4.5.1)

demonstrate that all field tracks are successfully covered by the GTSP pattern regardless of parity. In the case with odd number of tracks, however, B pattern may skip tracks by finding



(a)



(b)

Figure 4.5: (a) B pattern [4] for odd number of tracks (11 tracks) with one depot. The result of B pattern skips one track in this case by traversing the endpoints of tracks, i.e., the area in middle of the field is not covered. (b) B pattern [4] for even number of tracks (10 tracks) with one depot. The number on each endpoint of tracks is the visiting order of that endpoint. (Experiment 4.5.1)

the optimal path to traverse the endpoints, even though the cost between two endpoints in the same track is set to zero. The path of B pattern in such case is not feasible because the area in middle of test field will not be covered.

4.5.2 Effect of Specified Start/End Position

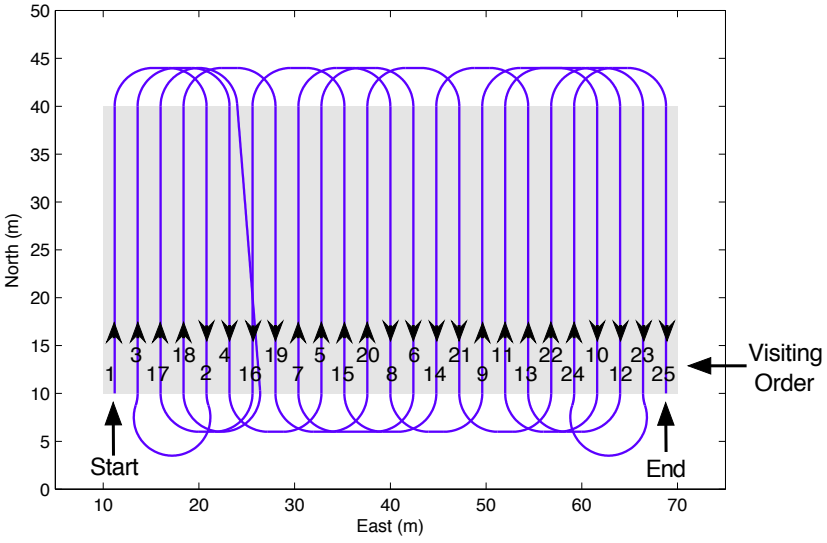
The second experiment is established to test the effect of specified start position and end position on the optimal traversal sequence. In such case, the vehicle starts and ends at specified endpoint of two different tracks. A rectangular field is used, which has dimensions of 60 m $\times$ 30 m (gray shaded areas). In order to completely cover the field, the required number of field tracks is 25.

Two cases are tested. In the first case, the start and end positions are specified on the same side of two different tracks. In the second case, the start and end positions are specified on the opposite sides of two different tracks. Fig. 4.6 shows the result of proposed GTSP pattern and Fig. 4.7 shows the result of B pattern [4]. The result demonstrates that proposed GTSP pattern provides feasible solutions for different specified start/end positions, while the B pattern can not guarantee to obtain a feasible solution because it may skip one track.

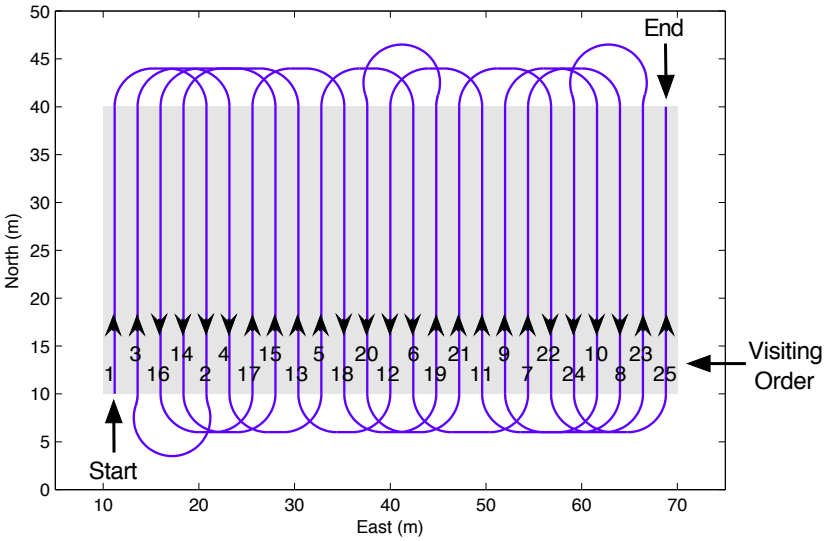
4.5.3 Performance with Unspecified Start/End Position

The third experiment is established to show the performance of the proposed method in dealing with applications that have no depot, as well as the situation of unspecified start location and end location. The proposed pattern in this paper is compared with three other well-known patterns. The field is non-rectangular in this series of tests. The track set is $S = \{1, 2, 3, \ldots, 20\}$.

The first pattern that the vehicle follows is the boustrophedon pattern. The field tracks are traversed continuously from one track to the adjacent uncovered track with alternate directions. Fig. 4.8 shows the solution of traversing the field with boustrophedon path and



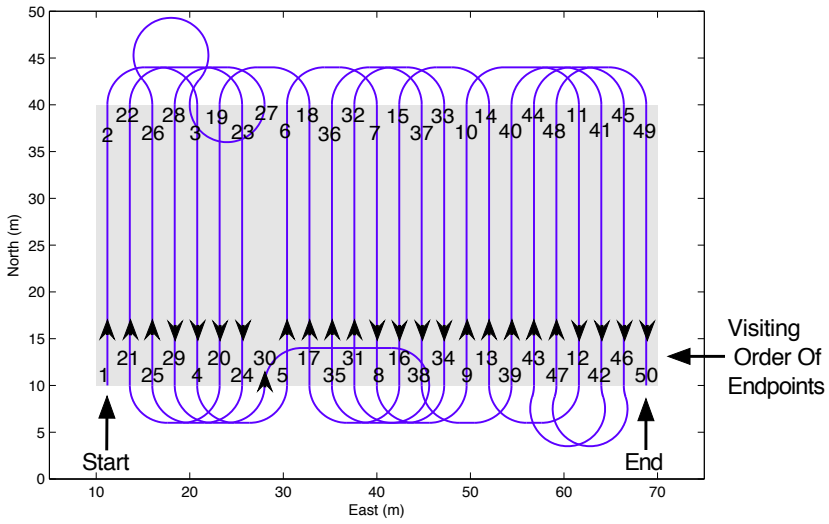
(a)



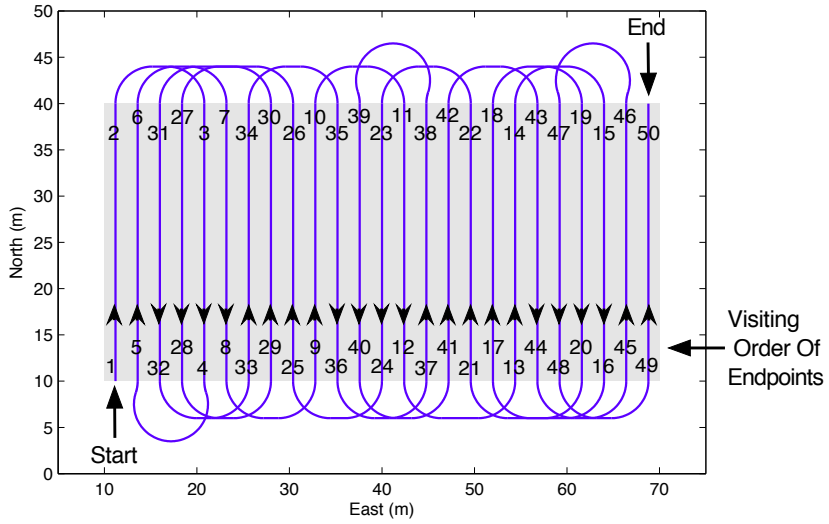
(b)

Figure 4.6: GTSP pattern for specified start and end positions (25 tracks). (a) Start position and end position are on the same side of two different tracks. (b) Start position and end position are on the opposite side of two different tracks. Shaded area is field that must be covered. The number on each track is the visiting order of that track. Arrows indicate the driving direction on each track. (Experiment 4.5.2)

the visiting order of each track. The visiting sequence is $\Pi^{bous}(S) = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 \rangle$.



(a)



(b)

Figure 4.7: B pattern [4] for specified start and end positions (25 tracks). (a) Start position and end position are on the same side of two different tracks. (b) Start position and end position are on the opposite side of two different tracks. The number on each endpoint of tracks is the visiting order of that endpoint. The B pattern skips one track in case (a) by traversing the endpoints of tracks, which results an infeasible solution. (Experiment 4.5.2)

The second pattern that the vehicle follows is the set pattern described in [3]. In this experiment, field tracks are grouped into three sets. The first set and the second set both contain 9 tracks and the third contains 2 tracks. Once the first set has been completed, the

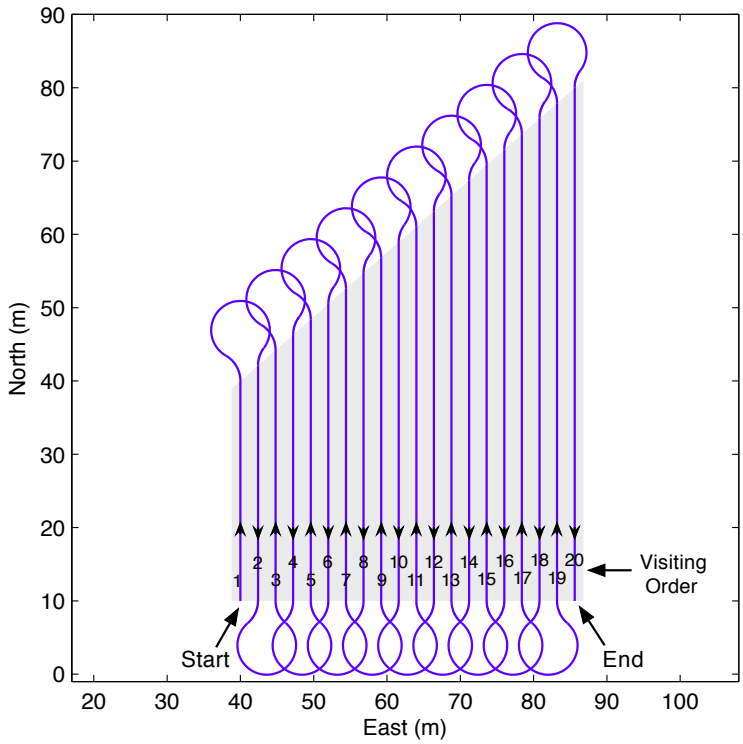


Figure 4.8: Boustrophedon pattern (20 tracks, trapezoidal shaped field). (Experiment 4.5.3)

second set is started with a similar way, but the initial driving direction is opposite. Since the last set has tracks fewer than 9, the tracks in this set are travelled in order. Fig. 4.9 shows the solution of traversing the field with set pattern and the visiting order of each track. The visiting sequence is $\Pi^{set}(S) = \langle 1, 6, 2, 7, 3, 8, 4, 9, 5, 10, 15, 11, 16, 12, 17, 13, 18, 14, 19, 20 \rangle$.

The third pattern that the vehicle follows is the B pattern described in [4]. Since the B pattern can not always provide a feasible solution for specified start/end locations, in this experiment, the start and end locations are not specified. Each track is represented by its two endpoints. The cost between two endpoints in the same track is set to zero and the cost between two endpoints in opposite sides of different tracks is set to a large penalty value. Other costs between endpoints are set to their corresponding Dubins distance. The TSP solution is also calculated by the LKH solver. Fig. 4.10 shows the solution of traversing

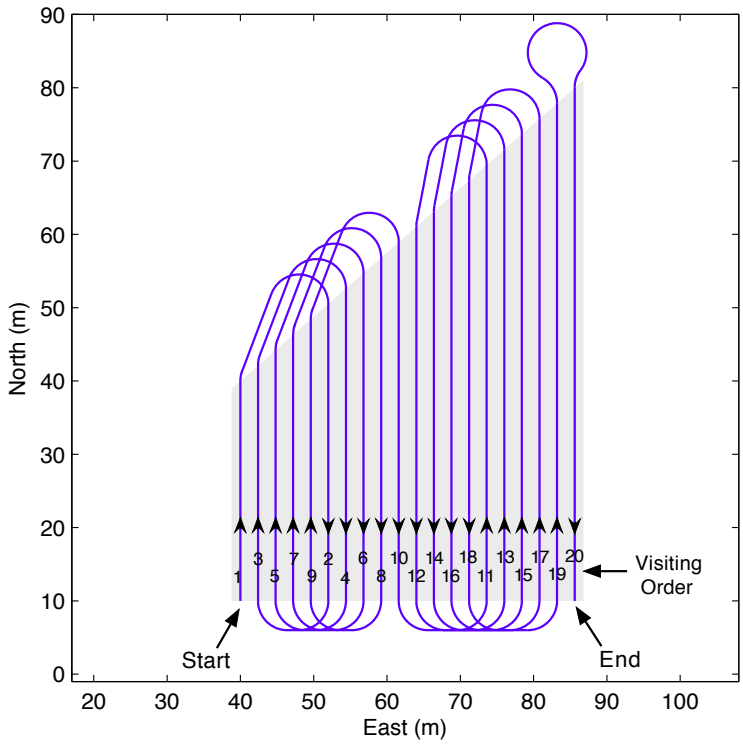


Figure 4.9: Set pattern [3] (20 tracks, trapezoidal shaped field) Set pattern is also called "Zamboni pattern", or "overlapping concentric ovals". (Experiment 4.5.3)

the field with B pattern and the visiting order of each track. The visiting sequence is $\Pi^b(S) = \langle 14,10,7,11,8,4,1,5,2,6,3,9,12,18,15,19,16,20,17,13 \rangle$

The fourth pattern that the vehicle follows is the GTSP pattern proposed in this paper. The GTSP pattern is also implemented with unspecified start and end locations. Fig. 4.11 shows the optimal solution of traversing the field with GTSP pattern and the visiting order of each track. The visiting sequence is $\Pi^{gtsp}(S) = \langle 13,17,20,16,19,15,18,14,11,7,10,4,1,5, 8,12,9,3,6,2 \rangle$.

The non-working distances measured during turnings by the above four traversal patterns are given in Table 4.1. The waste percentages of different patterns comparing to the working distance are also shown in this table.

In order to test the effect of minimum turning radius and operating width (space between tracks) on the non-working travel distance, cases with different minimum turning radius and operating width are also tested in this experiment. The minimum turning radius is ranged

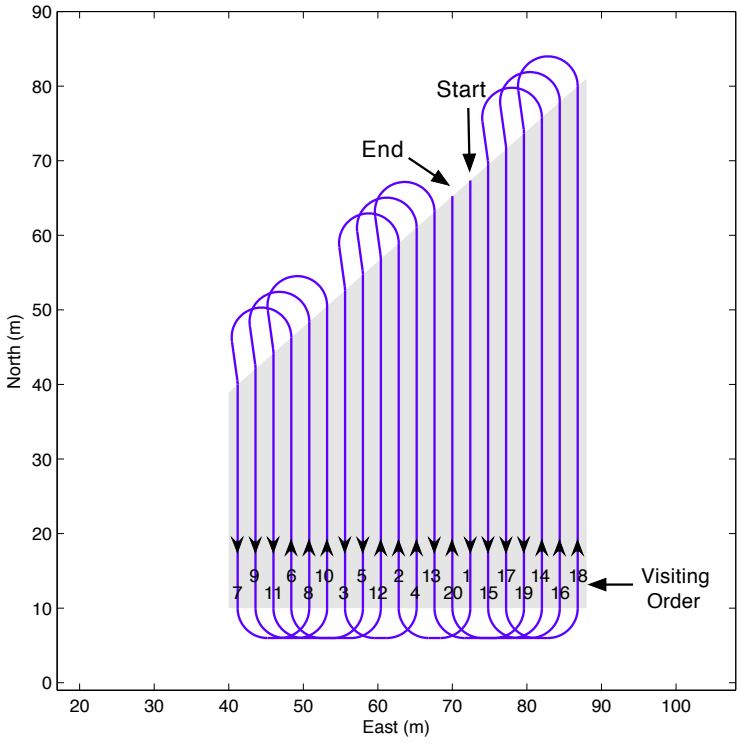


Figure 4.10: B pattern with no specified start position and end position (20 tracks, trapezoidal shaped field). (Experiment 4.5.3)

Table 4.1: Non-working distance of different path patterns for a single field (4m turning radius, 2.4m operating width). (Experiment 4.5.3)

Path pattern	**Working distance (m)**	**Non-working distance (m)**	**Waste (%)**
GTSP pattern (Fig. 4.11)	1000	321.68	32.17
B pattern (Fig. 4.10)	1000	321.68	32.17
Set pattern [3] (Fig. 4.9)	1000	369.28	36.93
Boustrophedon pattern (Fig. 4.8)	1000	498.38	49.84

from 4 m to 8 m with an incremental step of 0.5 m. The operating width is ranged from 1.2 m to 9.6 m with an incremental step of 1.2 m. In each case, the non-working distances of four patterns are measured. The savings of using GTSP pattern instead of the other three patterns are shown in Fig. 4.12, Fig. 4.13 and Fig. 4.14 respectively. As shown in Fig. 4.12, the savings for Boustrophedon pattern are increased as a function of increasing turning radius and as a function of decreasing operating width. By using GTSP pattern, the reduction of

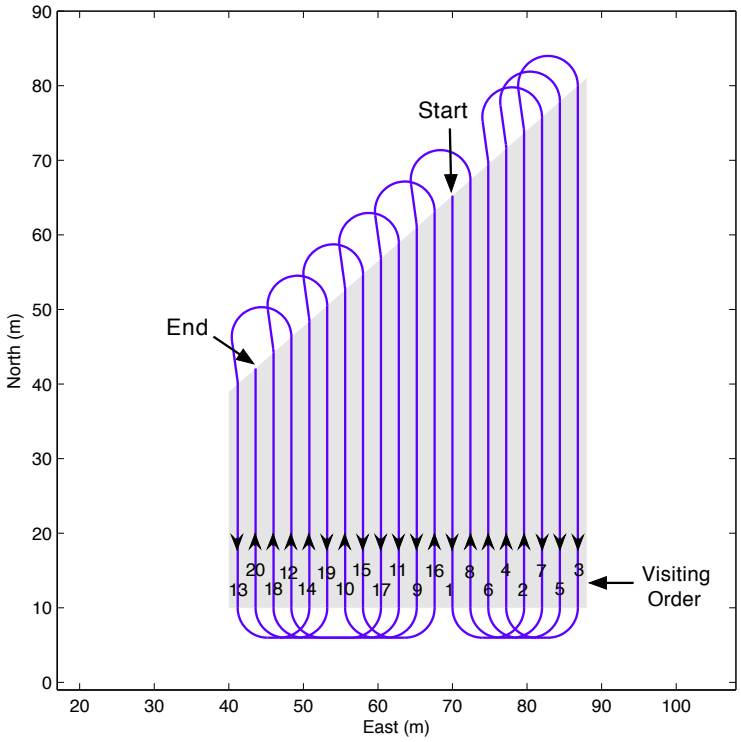


Figure 4.11: GTSP pattern with no specified start position and end position (20 tracks, trapezoidal shaped field). (Experiment 4.5.3)

non-working distance can reach up to 50% by comparing with the Boustrophedon pattern . Unlike the trend in Fig. 4.12, Fig. 4.13 shows that the savings for the Set pattern increase as the tuning radius decreases and the operating width increases. It is because when the turning radius is near half operating width or less than half operating width, the optimal solution tends to visit adjacent tracks first, which is very similar to boustrophedon pattern, but the Set pattern only follows path by skipping adjacent tracks in that case, which increases the non-working distance. The savings of non-working distance by using GTSP pattern instead of Set pattern can reach up to 40%. Fig. 4.14 shows that the saving percentages of non-working distance range from -0.4% to 0.9% by using GTSP pattern instead of B pattern. Considering the saving percentages are within a very small amount of range, the GTSP pattern performs very similarly to B pattern in the test field with unspecified start and end locations.

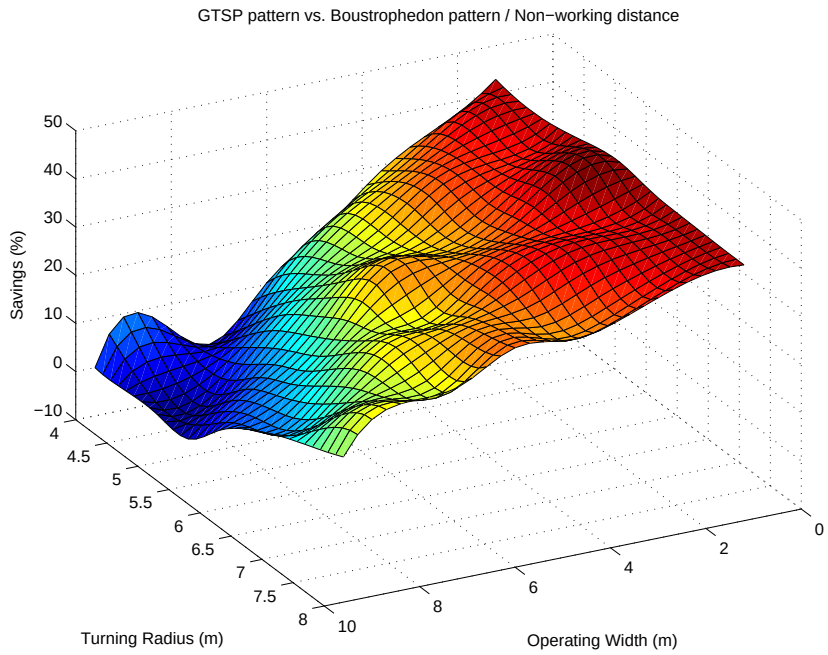


Figure 4.12: Savings in non-working distance by using GTSP pattern instead of Boustrophedon pattern. (Experiment 4.5.3)

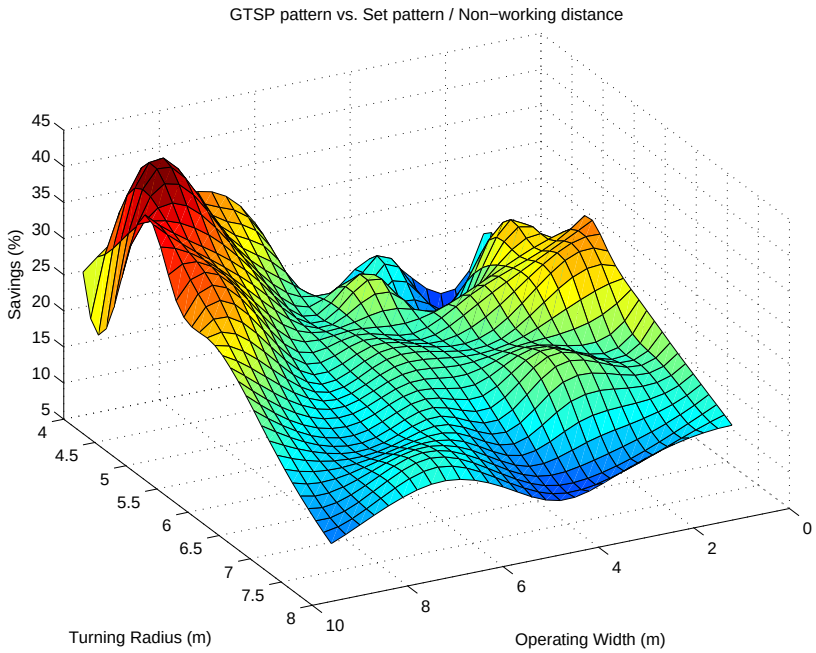


Figure 4.13: Savings in non-working distance by using GTSP pattern instead of Set pattern [3]. (Experiment 4.5.3)

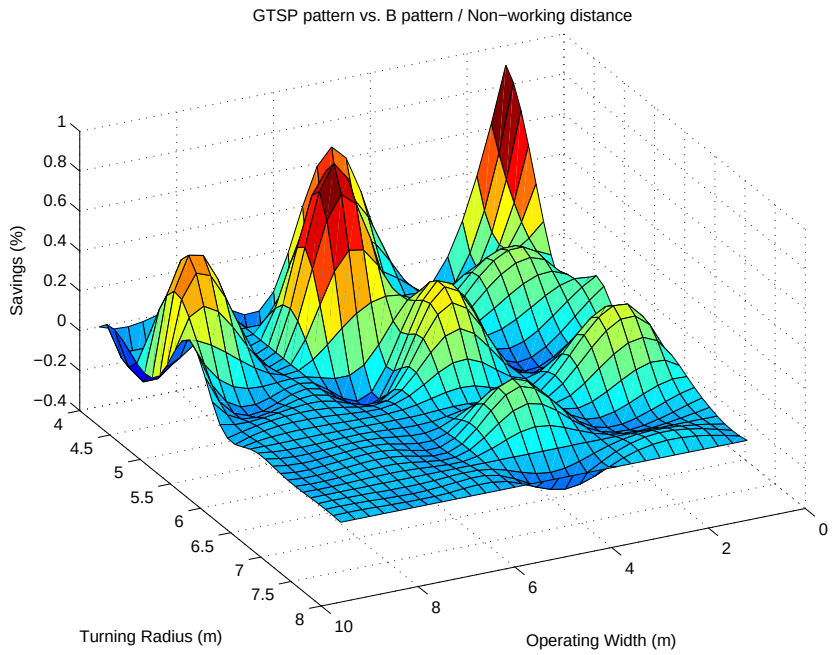


Figure 4.14: Savings in non-working distance by using GTSP pattern instead of B pattern [4]. (Experiment 4.5.3)

4.5.4 Performance on Multiple Decomposed Subfields

The fourth experiment is established to show the performance of the proposed method in dealing with multiple decomposed subfields. The experimented field is decomposed into three subfields. The dimensions of these subfields are: 52.8 m $\times$ 100 m (subfield 1), 19.2 m $\times$ 40 m (subfield 2), 19.2 m $\times$ 30 m (subfield 3).

Situation I: Each Pair of Tracks Can Be Connected

The distance between adjacent tracks is 2.4 m in each subfield. The numbers of tracks in these subfields are 22 tracks (subfield 1), 8 tracks (subfield 2) and 8 tracks (subfield 3). The traversal path is obtained by the first situation method in section 4.4, that is all tracks are treated equally as in the same field. The starting and ending locations are separated and not specified in this experiment. The track set S is numbered from west to east and from subfield 1 to subfield 3, represented by $S = \{1, 2, 3, \ldots, 37, 38\}$. Fig. 4.15 shows

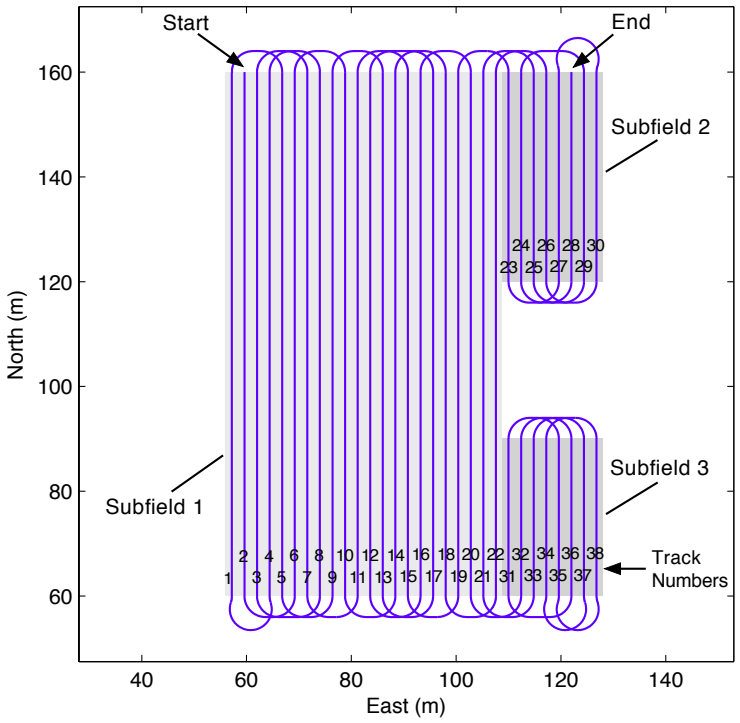


Figure 4.15: GTSP pattern for multiple subfields (4 m turning radius, 2.4 m operating width). The number on each track is the track number. Visiting sequence is in the paper. (Experiment 4.5.4)

the optimal traversal path with 4 m turning radius. The track numbers are also marked in Fig. 4.15. The optimal visiting sequence is $\Pi^{gtsp}(S) = \langle \overbrace{2, 6, 10, 14, 18, 22}^{subfield1}, \overbrace{26, 30, 27, 23}^{subfield2}, \overbrace{19, 15, 11, 7, 3, 8, 4, 1, 5, 9, 13, 17, 12, 16, 20}^{subfield1}, \overbrace{32, 36, 31, 35, 38, 34, 37, 33}^{subfield3}, \overbrace{21,}^{subfield1} \overbrace{25, 29, 24, 28}^{subfield2} \rangle$.

From the traversal sequence, it is noticed that the resulting traversal pattern is not similar to any conventional pattern. In conventional patterns, one subfield is started only if the previous subfield is completely covered. But in the proposed pattern, one subfield can be started without completion of the previous subfield. In this test, subfield 1 is visited three times to complete, and subfield 2 is visited twice to complete. This property can also be seen in the case with 6 m turning radius, as shown in Fig. 4.16. As the turning radius increasing, the non-working path at lower side of subfield 2 and the non-working path at upper side of subfield 3 are getting closer. The path planner can find a shorter path by connecting tracks

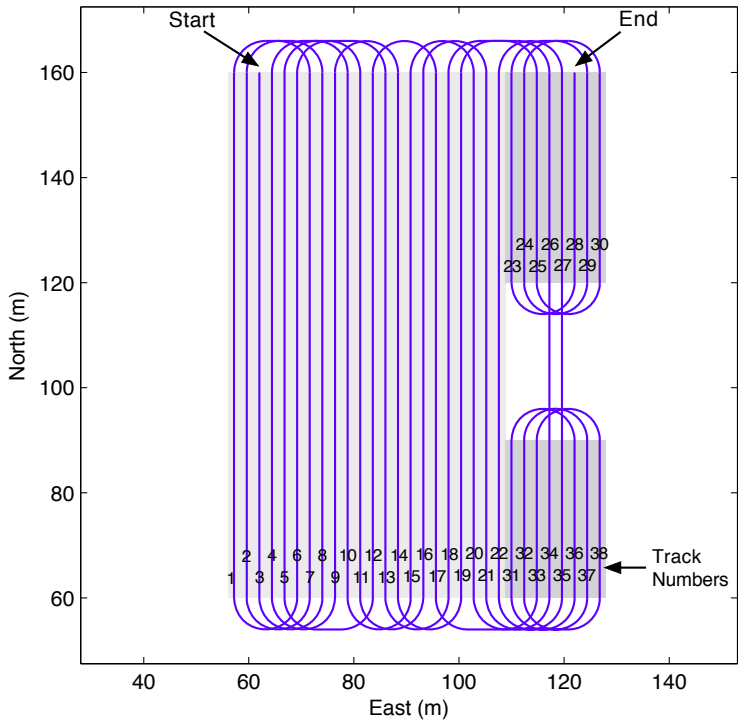


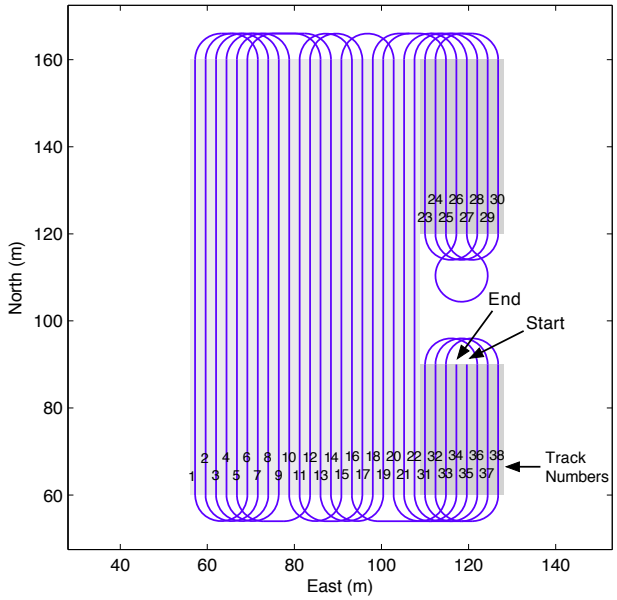
Figure 4.16: GTSP pattern for multiple subfields (6 m turning radius, 2.4 m operating width). (Experiment 4.5.4)

in subfield 2 and subfield 3. It can be seen that the path between subfield 2 and subfield 3 runs in an alternate way to minimize the non-working travel distance.

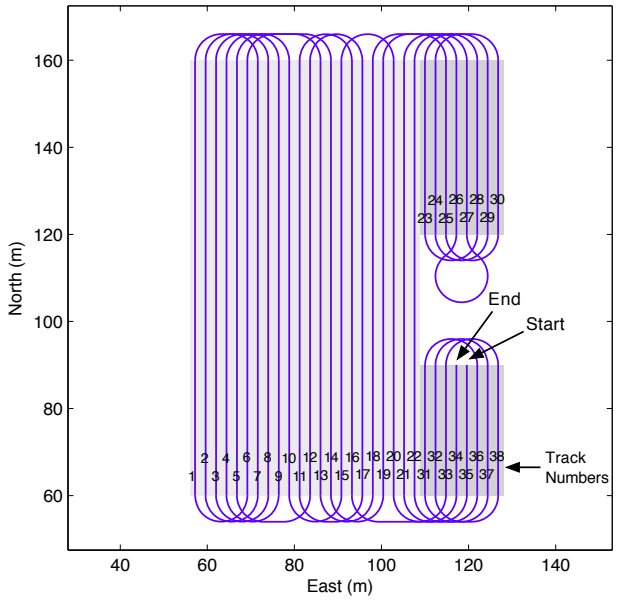
Situation II: Some Connections Between Tracks Are Forbidden

For some applications, connections between some subfields are not permitted (possibly due to an obstacle). In the following tests, it is assumed that connections between subfield 2 and subfield 3 are not permitted. The connections between subfield 1 and subfield 2 are permitted only if the connection between tracks is through the upper side of these two subfields. And the connections between subfield 1 and subfield 3 are permitted only if the connection between tracks is through the lower side of these two subfields.

By adopting the second situation method in section 4.4, the traversal path with 6 m turning radius and 2.4 m operating width is shown in Fig. 4.17a. Comparing to the results in Fig. 4.16, the tracks in subfield 2 and subfield 3 are not connected directly and other



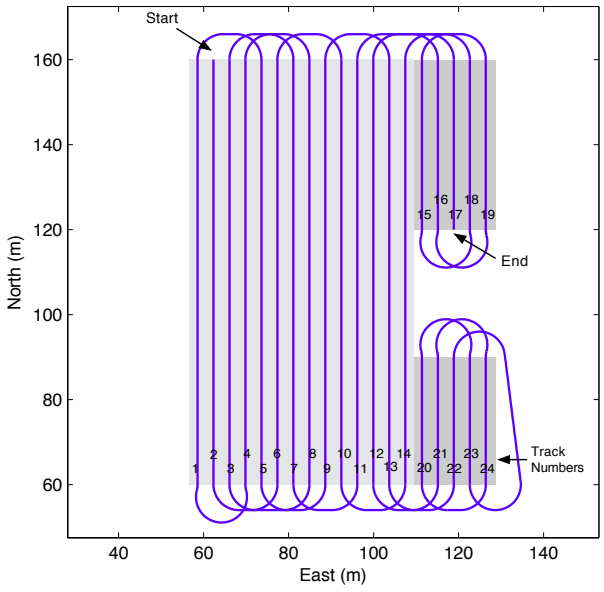
(a)



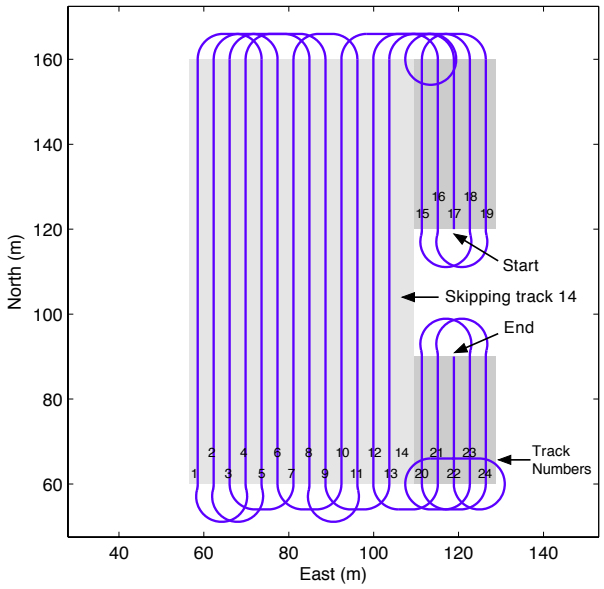
(b)

Figure 4.17: GTSP pattern and B pattern for multiple subfields (6 m turning radius, 2.4 m operating width). (a) Solution of GTSP pattern with restricted connections. (b) Solution of B pattern with restricted connections. (Experiment 4.5.4)

constraints are also satisfied. The result of B pattern under the same constraints is also shown in Fig. 4.17b for comparison. In this case, both GTSP pattern and B pattern can find the feasible solution and have the same traversal sequence.



(a)



(b)

Figure 4.18: GTSP pattern and B pattern for multiple subfields (6 m turning radius, 3.76 m operating width). (a) Solution of GTSP pattern with restricted connections. (b) Solution of B pattern with restricted connections. (Experiment 4.5.4)

To further compare the feasibility of GTSP pattern and B pattern for multiple subfields, the other test case with 6 m turning radius and 3.76 m operating width is implemented on

the same subfields. The numbers of tracks in these subfields are 14 tracks (subfield 1), 5 tracks (subfield 2) and 5 tracks (subfield 3), correspondingly. The results of GTSP pattern and B pattern are shown in Fig. 4.18. As illustrated, The GTSP pattern can find the feasible solution under the constraints. The B pattern skips track 14, which is not feasible for the application. In fact, in many other cases with different operating widths, the B pattern can not always obtain a feasible solution under constraints for multiple subfields.

Effect of Turning Radius and Operating Width

In order to test the effect of minimum turning radius and operating width (space between tracks) on the non-working travel distance, cases with different minimum turning radius and operating width are considered for GTSP pattern. The minimum turning radius is ranged from 4 m to 8 m with an incremental step of 0.5 m. The operating width is ranged from 1.2 m to 9.6 m with an incremental step of 1.2 m. Because the B pattern can not always obtain a feasible solution under constraints for the multiple subfields case, only boustrophedon pattern and set pattern are implemented to compare with the GTSP pattern. For boustrophedon pattern and set pattern, tracks in each subfield are traversed independently and the visiting sequence of subfields is set to be subfield 2 $\rightarrow$ subfield 1 $\rightarrow$ subfield 3. Then the final tour is obtained by connecting these sub-tours according to the visiting sequence of subfields. For GTSP pattern, it follows the above constraints that connections between subfield 2 and subfield 3 are not permitted; the connections between subfield 1 and subfield 2 are permitted only if the connection between tracks is through the upper side of these two subfields; the connections between subfield 1 and subfield 3 are permitted only if the connection between tracks is through the lower side of these two subfields.

In each testing case, the non-working distances of three patterns are measured. The savings of using GTSP pattern instead of boustrophedon pattern and set pattern are shown in Fig. 4.19 and Fig. 4.20 respectively. As illustrated, the similar trends of savings in the single field case are also held in the multiple subfields case. The GTSP pattern can reduce

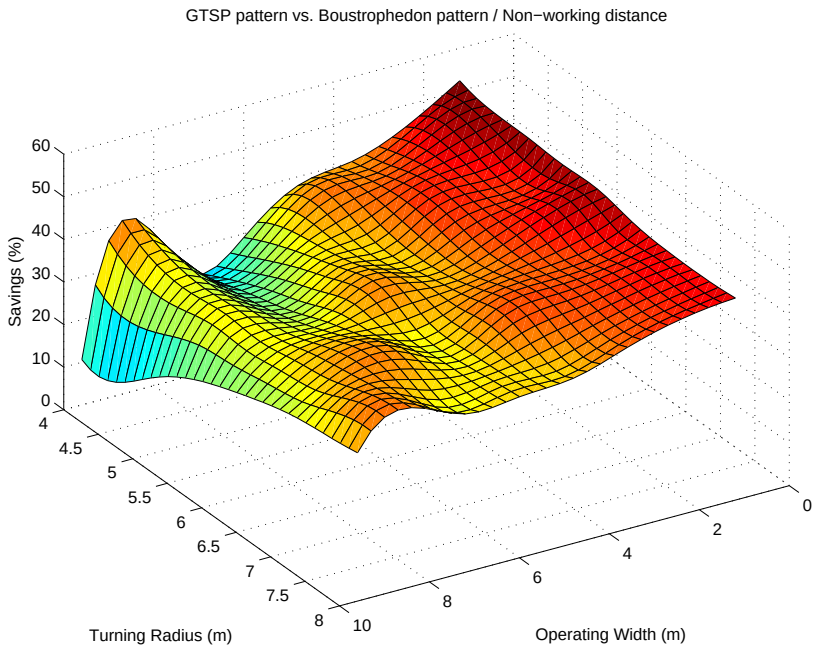


Figure 4.19: Savings in non-working distance by using GTSP pattern instead of Boustrophedon pattern for multiple subfields. (Experiment 4.5.4)

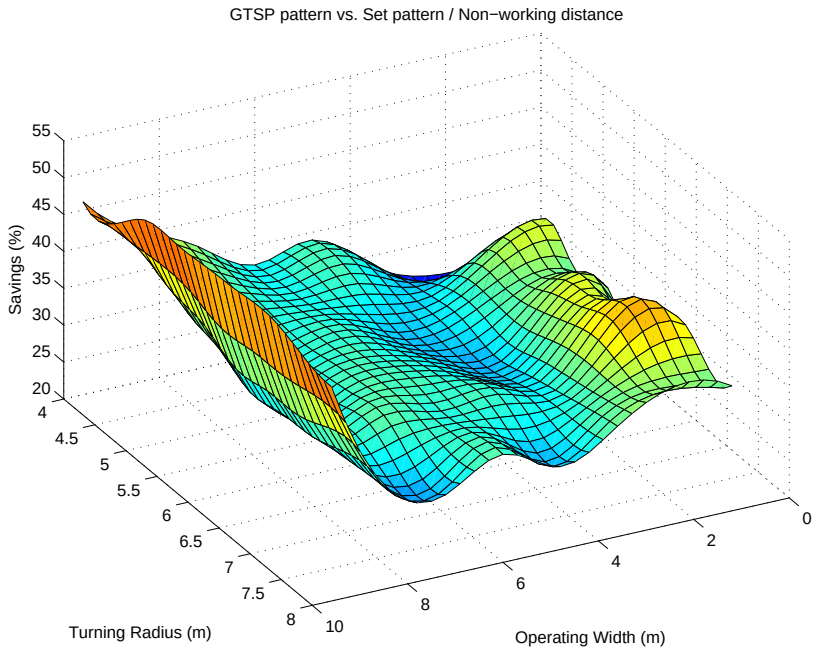


Figure 4.20: Savings in non-working distance by using GTSP pattern instead of Set pattern [3] for multiple subfields. (Experiment 4.5.4)

Table 4.2: Non-working distance of different path patterns for multiple subfields (4m turning radius, 2.4m operating width). (Experiment 4.5.4)

Path pattern	**Working distance (m)**	**Non-working distance (m)**	**Waste (%)**
GTSP pattern (Fig. 4.15)	2760	547.68	19.84
Set pattern [3]	2760	661.86	23.98
Boustrophedon pattern	2760	1037.05	37.57

Table 4.3: Non-working distance of different path patterns for multiple subfields (6m turning radius, 2.4m operating width). (Experiment 4.5.4)

Path pattern	**Working distance (m)**	**Non-working distance (m)**	**Waste (%)**
GTSP pattern (Fig. 4.16)	2760	791.69	28.68
Set pattern [3]	2760	1124.21	40.73
Boustrophedon pattern	2760	1571.17	56.93

the non-working distance up to 55% when compared with boustrophedon pattern and up to 50% when compared with set pattern.

The non-working distances of different patterns with 4 m turning radius and 2.4 m operating width are selected and shown in Table 4.2. The non-working distances of different patterns with 6 m turning radius and 2.4 m operating width are also selected and shown in Table 4.3.

4.6 Summary

The work in this chapter has introduced an optimization approach that takes the vehicle's characteristics into account to minimize the non-working distance in coverage path planning problems. The main task is to find the optimal traversal sequence for a given set of field tracks. The problem is modeled as a Generalized Traveling Salesman Problem (GTSP), which can be transformed into a standard Asymmetric Traveling Salesman Problem (ATSP),

and then the existing ATSP solvers can be applied to find the optimal solution. The experiments show that the proposed method provides feasible solutions for either odd number of tracks or even number of tracks with one depot. When compared to classical boustrophedon path or even set pattern, the total non-working distance can be reduced for both single convex field and multiple decomposed fields. In the given examples, wasted travel was reduced by up to 55% using the proposed methods, compared to the boustrophedon pattern and set pattern. The proposed work also solves several previously unresolved issues of B pattern in coverage path planning. While the authors focus on the Dubins vehicles in this paper, the proposed method could be applied to any non-holonomic vehicle whose cost between nodes is well defined.

Chapter 5

Dubins Traveling Salesman Problem

5.1 Introduction

In Traveling Salesman Problem (TSP), given a set of waypoints, the task is to determine the shortest tour to visit each waypoint only once and return to the starting waypoint. If the distance between any two waypoints is determined by Euclidean distance, it is called the Euclidean Traveling Salesman Problem (ETSP). ETSP has been applied to many robot path planning problems. However, when working with a car-like robot, researchers have to consider kinematic constraints such as minimum turning radius, i.e. the ETSP result may not provide an optimal solution. Typically, the car-like robot that can only move forward at a constant speed, with a minimum turning radius can be modeled as a Dubins vehicle [49]. In this chapter, the authors will focus on the Dubins Traveling Salesman Problem (DTSP) and design a genetic algorithm to solve the DTSP.

The rest of this chapter is organized as follows. In Section 5.2, the problem statement of the DTSP is introduced. In Section 5.3, a genetic algorithm is designed for the DTSP, and several numerical experiments are conducted in Section 5.4 to compare the performance of different algorithms. Section 5.5 summarize the key results of this chapter.

5.2 Problem Statement

Let $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ denote a permutation of the given waypoints $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n)\}$, and $\Theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_n\}$ denote the corresponding orientations of a Dubins vehicle. Here $\sigma_i \in \{1, 2, \ldots, n\}$, $\theta_i \in [0, 2\pi]$, for $i = 1, 2, \ldots, n$. The Dubins distance between configuration (x_i, y_i, θ_i) and configuration (x_j, y_j, θ_j) is denoted by $\mathcal{D}(i, j)$.

The optimal path between any two waypoints in the ordered sequence can be calculated by Dubins path. Therefore the Dubins Traveling Salesman Problem (DTSP) is to find sets Σ and Θ that minimizes the total tour length $\mathcal{L}$:

$$\mathcal{L} = \sum_{i=1}^{n-1} \mathcal{D}(\sigma_i, \sigma_{i+1}) + \mathcal{D}(\sigma_n, \sigma_1) \tag{5.1}$$

5.3 Algorithm Design for DTSP

5.3.1 Genetic Algorithm

Genetic algorithm (GA) [81] is a global searching method that mimics the natural evolution process to optimize the searching problem. It attempts to find the best solution by generating a collection (called population) of strings (called chromosomes) that encode the potential solutions (called individuals), and uses genetic operations such as selection, crossover, and mutation to produce a new better solution. This process continues until an acceptable solution is found. GA has many advantages including the ability to search in continuous variable domain, discrete variable domain or mix type variable domain. In this chapter, the algorithm has several parameters, *maximum generation size* (N_g), *population size* (N_p), *elitism rate* (P_e), *crossover rate* (P_c), *permutation mutation rate* (P_m), *orientation mutation rate* (P_a). The main process is described in Algorithm 3, and the detail implementation of each operation will be discussed in the following sections.

5.3.2 Encoding and Initialization

Encoding of the individual is the first problem when starting genetic algorithms. The choice of encoding method depends largely on the specific problem. Since DTSP must determine not only the visiting order of the waypoints but also the corresponding orientations, both discrete and continuous variables must be considered. Therefore the mixed permutation encoding and value encoding method is adopted: $\{(\sigma_1, \theta_{\sigma_1}), (\sigma_2, \theta_{\sigma_2}), \ldots, (\sigma_n, \theta_{\sigma_n})\}$.

Algorithm 2 process of GA for DTSP

Input: List of given waypoints
Output: Optimal visiting sequence and optimal orientation at each waypoint

set values to $N_g, N_p, P_e, P_c, P_m, P_a$
encode the given waypoints
initialize the current generation G_c with N_p
calculate the fitness value of each individual in G_c
sort population of G_c by fitness values in decreasing way
for $i \leftarrow 2$ to N_g **do**
    elitism selection for top $\lfloor N_p \times P_e \rfloor$ individuals
    for $j \leftarrow \lfloor N_p \times P_e \rfloor + 1$ to N_p **do**
        $parent_1 \leftarrow$ roulette wheel selection
        $parent_2 \leftarrow$ roulette wheel selection
        if possibility $\leq P_c$ **then**
            $child \leftarrow$ crossover $parent_1$ and $parent_2$;
        else if $parent_1$ has greater fitness value **then**
            $child \leftarrow parent_1$
        else
            $child \leftarrow parent_2$
        end if
        if possibility $\leq P_m$ **then**
            reversion mutation of *child*;
        end if
        if possibility $\leq P_m$ **then**
            reciprocal exchange mutation of *child*;
        end if
        if possibility $\leq P_a$ **then**
            shift mutation of *child*;
        end if
        add *child* to the next generation G_n
    end for
    calculate the fitness values of each individual in G_n
    sort G_n by fitness values in decreasing way
    substitute G_c with G_n
end for
return top individual of G_c

For example, $\{(1, 2.1),\ (2, 1.5),\ (4, 6.2),\ (5, 1.6),\ (3, 4.2)\}$ means the robot starts from waypoint 1 with orientation 2.1, then goes to waypoint 2 with orientation 1.5, then goes to waypoint 4 with orientation 6.2, and so on, finally returning to the starting waypoint 1 with orientation 2.1. For initialization, given population size N_p, the permutation order and the corresponding orientations are randomly generated for each individual.

5.3.3 Fitness Function

The fitness function guides the genetic algorithm to the best solution within a large search space, and must be well-chosen so that the GA searches effectively and avoids local minima. For DTSP, the goal is to find the shortest total length to visit all waypoints, so the authors use the reciprocal of the total length to be the fitness function:

$$fitness = \frac{1}{\mathcal{L}} \tag{5.2}$$

where $\mathcal{L}$ is defined in (6.1). In this case, one individual is more fit than another one if $fitness_1 > fitness_2$.

5.3.4 Selection Operator

Selection is the stage of a genetic algorithm in which individuals are selected as parents to generate the next generation. In this dissertation, the elitism selection [82] and roulette wheel selection [82] are adopted. In elitism, the top $\lfloor N_p \times P_e \rfloor$ fittest individuals in the current generation are directly copied into the next generation. Elitism prevents loss of the best found solution. Then the remaining individuals are selected by roulette wheel selection. The process of roulette wheel selection is described as follows:

1. Calculate the accumulated fitness values, where the accumulated fitness value of an individual is the sum of its own fitness value and the fitness values of all the previous individuals.

2. Chose a random number m between 0 and the sum of fitness values of all individuals.

3. The selected individual is the first one whose accumulated fitness value is greater than m.

5.3.5 Crossover Operator

Crossover is the stage that two parents are joined together to produce a new offspring. The underlying idea of this operation is that the new offspring may be better than both of the parents if it obtains the best characteristics from each of the parents. In this dissertation, a greedy crossover method [83] is adopted. Let $E(i, j)$ denotes the Euclidean distance between configuration (x_i, y_i, θ_i) and configuration (x_j, y_j, θ_j). Let $p_1(i)$ denotes the waypoint at the position i in $parent_1$, and $p_2(j)$ denotes the waypoint at the position j in $parent_2$. The process is as follows:

1. Make two copies of the selected parents, say $parent_1$ and $parent_2$ respectively.

2. Randomly select a waypoint w to be the current waypoint of the *child*.

3. Find the positions of the current waypoint in $parent_1$ and $parent_2$, say i, j respectively.

4. Calculate the Euclidean distance from the right neighbor waypoint to the current waypoint in each parent, $E(p_1(i), p_1(i+1))$ and $E(p_2(j), p_2(j+1))$. If i or j is the last position in the parent, then calculate distance $E(p_1(i), p_1(1))$ or $E(p_2(j), p_2(1))$.

5. If neither the right neighbor waypoint in $parent_1$ or $parent_2$ exists in the *child*, then select the right neighbor which yields the shortest distance to be the current waypoint. If the distances are equal, then select one of the two right neighbors randomly. If both the right neighbor waypoints in $parent_1$ and $parent_2$ exist in the *child*, then randomly select the other waypoint which does not exist in the *child* to be the current waypoint.

6. Remove w from $parent_1$ and $parent_2$.

7. If the *child* tour is complete, stop; otherwise, go to step 3.

5.3.6 Mutation Operator

The mutation operator introduces and sustains diversity of the generation. It allows the algorithm to avoid local minima by preventing the individuals from becoming too similar to each other. In this dissertation, the mutation operation consists of reversion, reciprocal exchange and shift mutation [82].

- *Reversion Mutation*: Reversion mutation selects two waypoints along the length of the *child*, which is cut at these waypoints, and waypoints between the two waypoints are reversed, including the orientations.

- *Reciprocal Exchange Mutation*: Reciprocal exchange randomly selects two waypoints in the *child*, and swap these two waypoints, including the orientations.

- *Shift Mutation*: In shift mutation, a waypoint is chosen randomly, then the orientation of this point is reset randomly in the interval $[0, 2\pi]$.

5.4 Experiment

The performance of GA has been examined for both low waypoint density case and high waypoint density case. The authors conduct experiments on two cases. The first experiment (low density case) is implemented on a 20×20 square with minimum turning radius 1. The other (high density case) is on a 5×5 square with minimum turning radius 1. In both cases, the waypoints are generated randomly, and the numbers of waypoints varies from 5 to 50 with increment 5. For each given number of waypoints, 30 samples are randomly generated, and the average length of these samples is computed. The authors also compare the result of GA with Alternating Algorithm (AA) [84] and the Randomized Headings Algorithm (RHA) [58]. The headings of RHA in this experiment are randomly generated in the interval $[0, 2\pi]$. The Lin-Kernighan Heuristic (LKH) [80] is used to calculate the Euclidean Traveling Salesman

Table 5.1: Parameter Table for 20x20 case

Parameter	Symbol	Value
maximum generation size	P_g	7000
population size	P_n	20
elitism rate	P_e	20%
crossover rate	P_c	90%
permutation mutation rate	P_m	7%
orientation mutation rate	P_a	90%

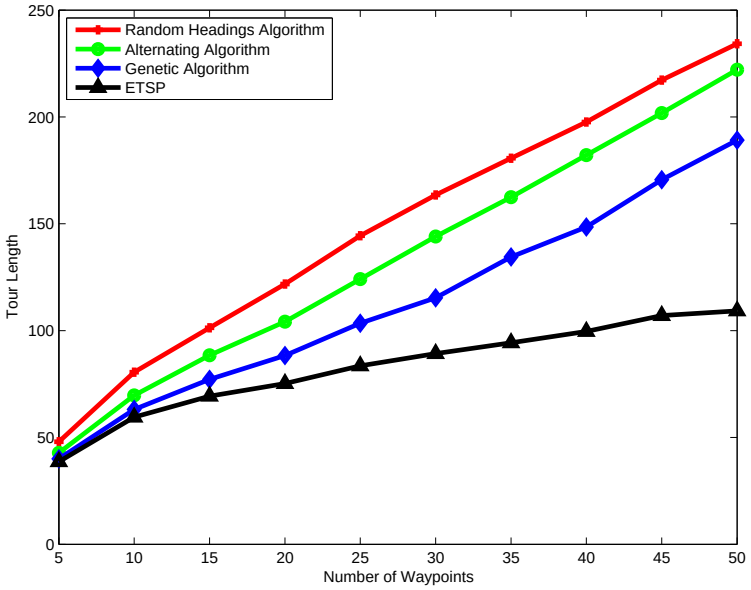


Figure 5.1: 20x20 square (low density) case comparison.

Problem (ETSP) orders for AA and Asymmetric Traveling Salesman Problem (ATSP) orders for RHA. The ETSP result also represents the lower bound. Fig. 5.1 shows the performance of different algorithms under low waypoint density. Fig. 5.3, Fig. 5.4 and Fig. 5.5 are instances of different algorithms in this case. Fig. 5.2 shows the performance of different algorithms under high waypoint density. Fig. 5.6, Fig. 5.7 and Fig. 5.8 are instances of different algorithms in this case. The genetic algorithm parameters for these two cases are shown in Table 5.1 and Table 5.2.

Table 5.2: Parameter Table for 5x5 case

Parameter	**Symbol**	**Value**
maximum generation size	P_g	11000
population size	P_n	35
elitism rate	P_e	55%
crossover rate	P_c	100%
permutation mutation rate	P_m	7%
orientation mutation rate	P_a	7%

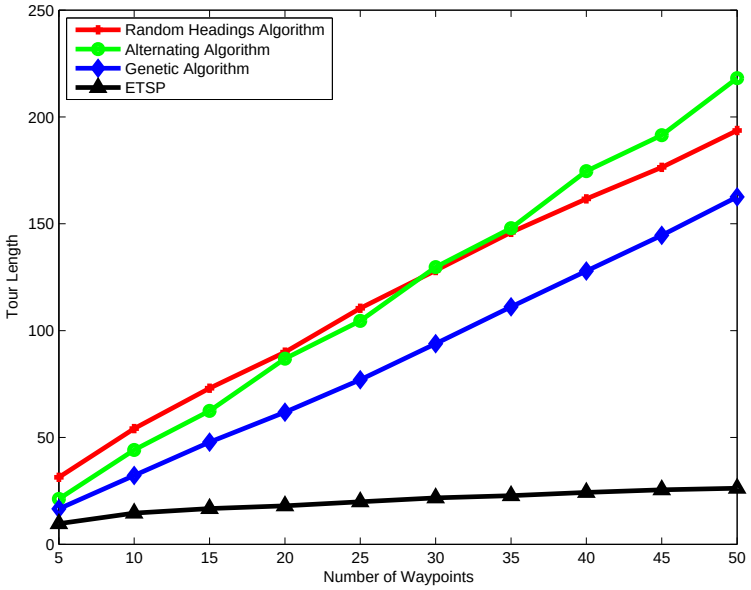


Figure 5.2: 5x5 square (high density) case comparison.

As shown in Fig. 5.1 and Fig. 5.2, when the density of the waypoints increases, the performance of the Alternating Algorithm decreases comparing to the performance of the Randomized Headings Algorithm. However when waypoints are far apart, the Dubins problem becomes similar to the Euclidean problem, therefore the Alternating Algorithm performs almost optimally. This has already be described in Le Ny's work [58]. Therefore the authors focus the discussion on the comparison between GA and AA in low waypoint density case and comparison between GA and RHA in high waypoint density case.

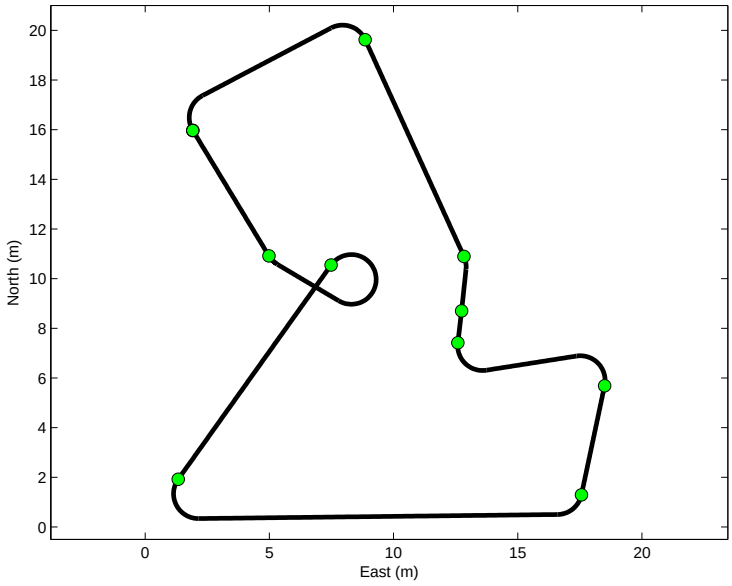
When the waypoint density is low, the performance of GA is much better than AA, because the ETSP order plays the dominant role in the DTSP. But since AA does not optimize the orientations, then when the result of ETSP is obtained, AA is finished after connecting the ordered points. In contrast, the proposed GA optimizes the orientations after achieving the ETSP order.

When the waypoint density is high, i.e. the distance between each pair of waypoints is small relative to the turning radius, the performance of GA is no worse than RHA. The reason is the optimal order of the ETSP is no longer the optimal order for DTSP. Therefore, searching other orders for DTSP becomes important. RHA takes this situation into consideration in its implementation, but once the initial random headings are fixed, the algorithm result is based on those fixed headings. RHA discards other possible headings. The GA algorithm addresses this problem with a shift mutation operator, which enhances the searching ability for other possible headings.

It should be noted, however, that when waypoint number is larger, the time performance of GA decreases comparing to RHA and AA. The reason is that RHA and AA benefit from the state of the art TSP solving tool to calculate the optimal asymmetric or symmetric TSP result. The implementation of GA described in this chapter is not as fast as those tools when dealing with large number of waypoints. Introducing more advanced GA searching techniques, however, may improve the computing time issue.

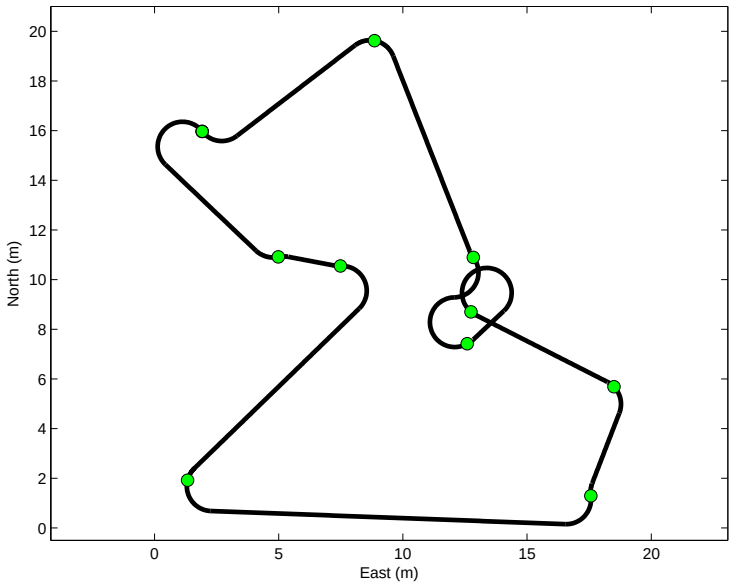
5.5 Summary

In this chapter, a genetic algorithm is designed to find the shortest length tour within a given set of waypoints for the Dubins vehicle (DTSP). The numerical results show that the GA method performs well in both low waypoint density case and high waypoint density case because of the global searching ability. When the number of waypoints becomes very large, however, the proposed GA algorithm is not as efficient as the well-established Alternating Algorithm and Randomize Headings Algorithm. In the future work, the authors will



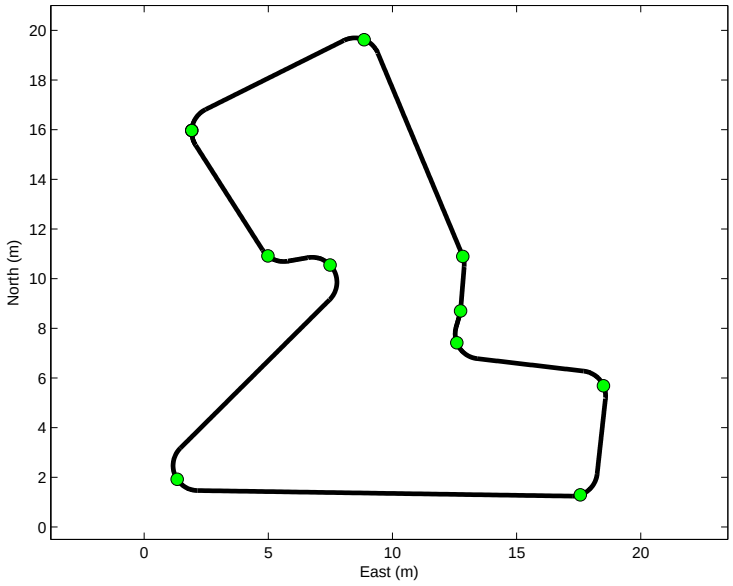
(a)

Figure 5.3: Alternating algorithm for low waypoint density case with 10 waypoints. The tour length is 76.64 m.



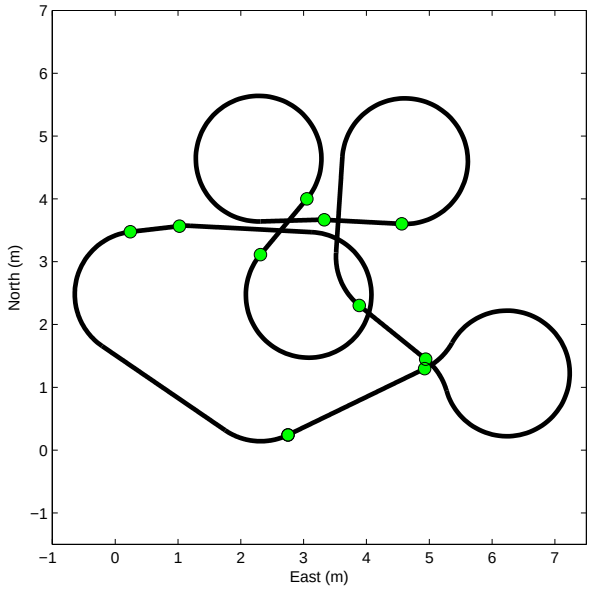
(a)

Figure 5.4: Random headings algorithm for low waypoint density case with 10 waypoints. The tour length is 83.89 m.



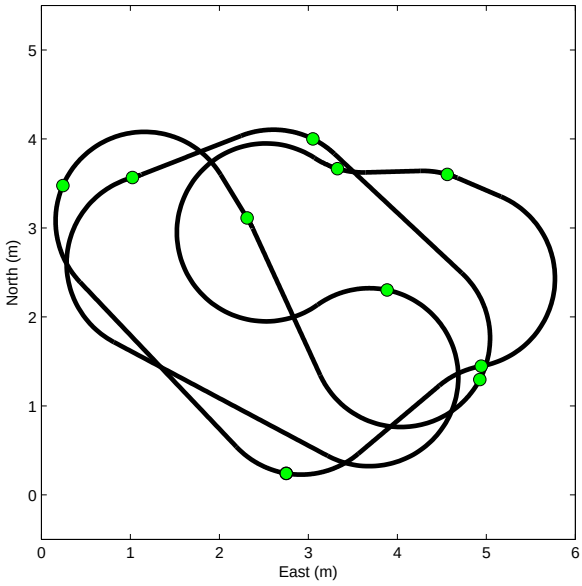
(a)

Figure 5.5: Genetic algorithm for low waypoint density case with 10 waypoints. The tour length is 68.89 m.



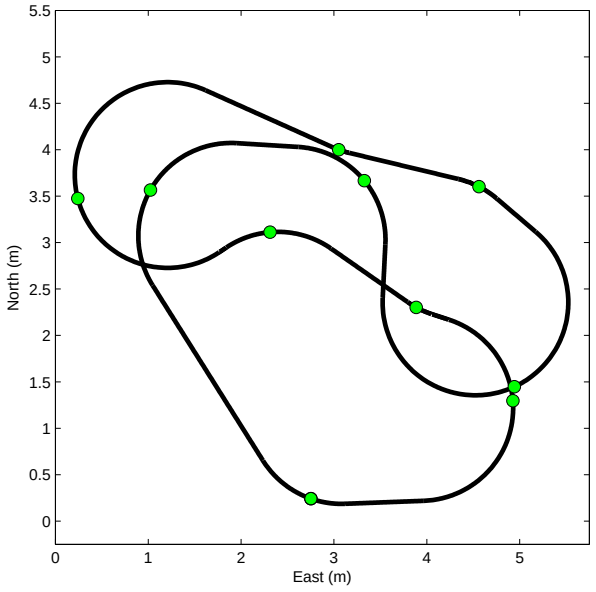
(a)

Figure 5.6: Alternating algorithm for high waypoint density case with 10 waypoints. The tour length is 40.62 m.



(a)

Figure 5.7: Random headings algorithm for high waypoint density case with 10 waypoints. The tour length is 38.03 m.



(a)

Figure 5.8: Genetic algorithm for high waypoint density case with 10 waypoints. The tour length is 26.82 m.

explore the searching ability by adapting the parameters of the genetic algorithms during the searching process to enhance the global searching ability, and mix state of the art local searching techniques to improve the local searching ability.

Chapter 6

Dubins Traveling Salesman Problem with Neighborhoods

6.1 Introduction

The work in this chapter takes the physical size of sensor scope into consideration when planning a path for the waypoints visiting problem. Considering the sensor scope, there is no constraint that the center of sensor scope need to locate at the exact waypoint position, i.e. a waypoint may be located anywhere within the perimeter of the sensor scope. Thus, each waypoint can be treated as a disk, whose center locates at the waypoint and radius equals to the radius of sensor scope. Once the center of sensor scope enters a disk during traversal, it is assured that the waypoint represented by this disk will be covered by the sensor. The problem is to find a shortest length tour such that the sensor center traverses every disk and returns to the starting one. Then the path planning problem of the robot-trailer system can be modeled as a Dubins Traveling Salesman Problem with Neighborhoods (DTSPN) [76], where in this dissertation the neighborhoods are represented by disks. Considering the nature of the DTSPN, the author proposes a new approach to approximate the DTSPN. The main idea is to decompose this problem into two stages: firstly the authors design a new algorithm for the TSPN to determine the visiting sequence of the disks and entry points, which can be applied for both disjoint case and overlapped case; secondly use the result of TSPN to form a DTSP, and design a DTSP algorithm to determine the heading at each entry point.

The remainder of this chapter is organized as follows. In Section 6.2, the problem statement of DTSPN is formally introduced. In Section 6.3, an algorithm is designed for the DTSPN. The theoretical analysis of this algorithm is shown in Section 6.4. Then numerical experiments and practical experiments are conducted in Section 6.5 and Section 6.6 respectively. Section 6.7 summarizes the key results in this chapter.

6.2 Problem Statement

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ be a set of n compact regions in the plane, and $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be an ordered permutation of $\{1, \ldots, n\}$. Let $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ denote configurations of a Dubins vehicle when it enters corresponding region R_i, for $i = 1, \ldots, n$. The Dubins distance between configuration X_i and configuration X_j with minimum turning radius ρ is denoted by $\mathcal{C}_\rho(X_i, X_j)$. The Dubins Traveling Salesman Problem with Neighborhoods (DTSPN) is to find a set Σ and $\mathcal{X}$ that minimizes the total tour length $\mathcal{L}$:

$$\mathcal{L} = \sum_{i=1}^{n-1} \mathcal{C}_\rho(X_{\sigma_i}, X_{\sigma_{i+1}}) + \mathcal{C}_\rho(X_{\sigma_n}, X_{\sigma_1}) \tag{6.1}$$

In this chapter, the regions are described as disks. The disk radius equals to radius of sensor scope and disk centers locate at the given set of waypoints. The sensor center is assumed to follow the tour path.

6.3 Algorithm Design

The proposed algorithm composes of four sequential steps. It firstly takes the disk centers to be entry points and obtains an optimal Euclidean Traveling Salesman Problem (ETSP) tour over these entry points; then adopts a Combination Operation to combine overlapped disks and seek entry points that can simultaneously visit several disks; and then uses an Alternating Iterative Algorithm (AIA) to further shorten the tour length by finding alternative entry points of the disks; finally uses the alternative entry points to form a DTSP, and obtains the headings of entry points by a DTSP method. An illustration of the process is shown in Fig. 6.1.

6.3.1 Find the Optimal ETSP Tour

In this section, the algorithm firstly takes the disk centers to be entry points and obtain the optimal ETSP tour T_{tsp} over these entry points. The optimal ETSP tour can be obtained

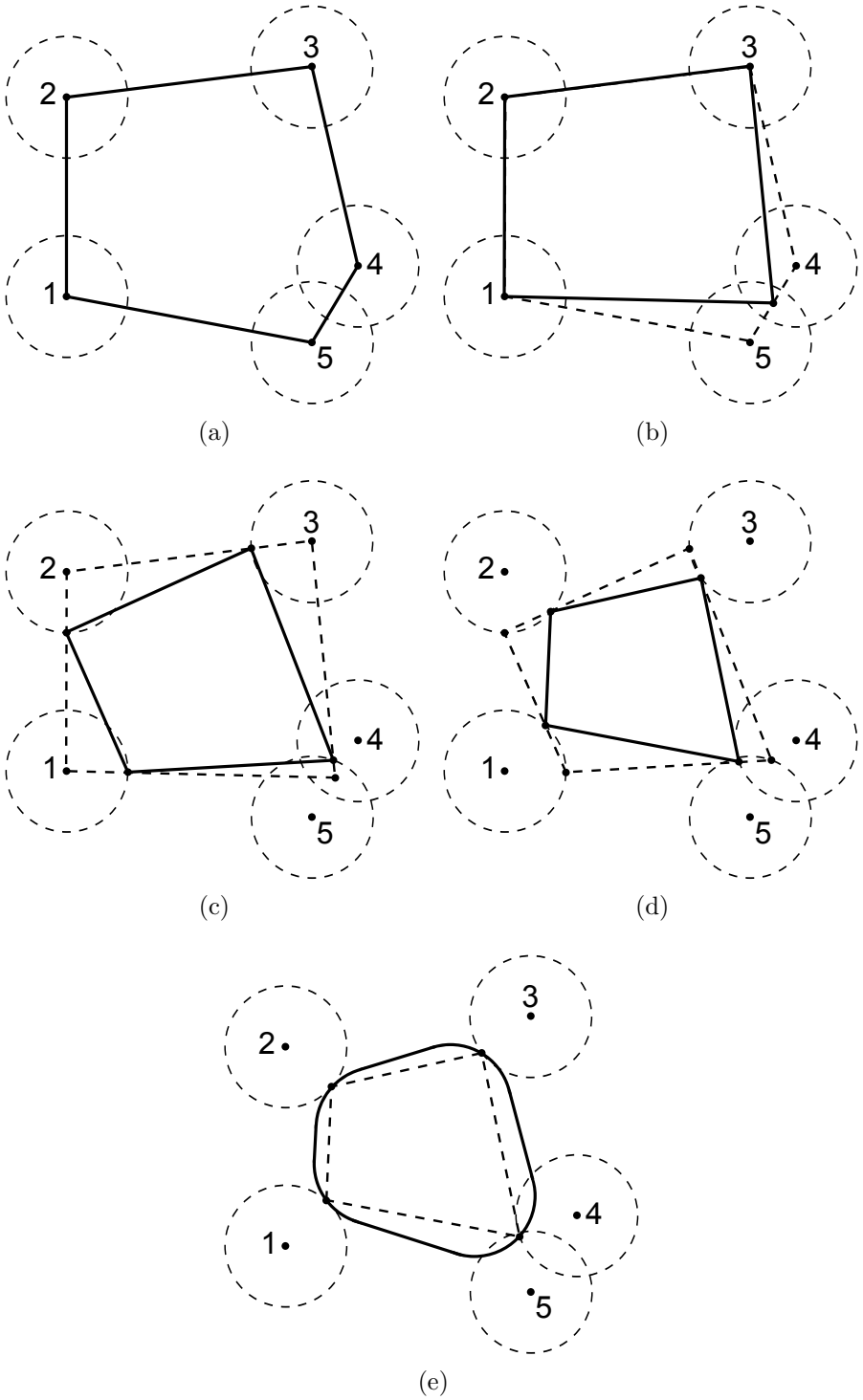


Figure 6.1: Illustration of DTSPN process. (a) Optimal ETSP tour (b) Combination Operation (c) Odd step of Alternating Iterative Algorithm (d) Even step of Alternating Iterative Algorithm (e) Optimal DTSP tour

by the LKH solver [80], which is an effective implementation of the Lin-Kernighan heuristic for solving Traveling Salesman Problem. Although it is a heuristic approach, computational experiments have shown that it performs very well in practice. It is widely accepted that LKH will produce an efficiently exact result in scales up to hundreds or even thousands of cities [80].

6.3.2 Combination Operation

For the case in which the disks are overlapped, the authors further introduce a Combination Operation [66] that seeks to take advantage of the possibility of visiting several disks at one single entry point.

In [66], the researchers modified the Welzl's algorithm [85] to combine several entry points into one entry point that can simultaneously visit several disks, based on the order of the optimal ETSP tour. The original Welzl's algorithm is a randomized algorithm for solving the minimum enclosing disk problem in the Euclidean plane. It can find the smallest disk that contains a given set of points in expected $O(n)$ time.

In this section, the authors adopt a new version of implementation for the Combination Operation. The main idea of Combination Operation is to find the maximal group of entry points covered by an enclosing disk whose radius equals to that of the given disks, via Welzl's algorithm. The algorithm iterates the ordered entry points. At each entry point, it tests whether the current entry point belongs to the previous enclosing disk. If it does, the algorithm includes the current entry point in the previous enclosing disk and proceeds to test the next entry point. If the current entry point does not belong to the previous enclosing disk, the algorithm obtains the intersection region of given disks whose entry points included in the previous enclosing disk, and then takes the center of previous enclosing disk as a new entry point, instead of the entry points included in the previous enclosing disk. Since the center of previous enclosing disk lies inside this intersection region of overlapped disks, the algorithm associates it with this intersection region. Then the algorithm restarts Welzl's

algorithm from the current entry point and goes on testing the rest entry points. The process is described in Algorithm 3. Note that the number of entry points in any tour of this chapter is 1 greater than the actual number of entry points, because the first entry point of the tour is also the last entry point, thus ensuring the tour is closed.

The notations used in the algorithm are listed as follows:

- Optimal ETSP tour over entry points, T_{tsp}, where $T_{tsp}[i]$ denotes the ith entry point in the tour.
- Radius of the given disks, r.
- Test entry point set, T.
- Tour after Combination Operation, T_{com}.
- Current enclosing disk, D_c.
- Previous enclosing disk, D_p.

In line 10 of Algorithm 3, the reason to obtain the intersection region of disks is based on the fact that there are many choices to place one entry point that can visit several overlapped disks, e.g. any point within the intersection region of these overlapped disks. In order to seek the optimal entry point within the intersection region in the next step, it is necessary to identify the intersection region of the overlapped disks. It is possible that there is only one entry point in T, in such case the intersection region is the the corresponding disk whose entry point in T.

The intersection region of overlapped disks is a convex shape bounded by circular arcs that meet at vertices where two or more circles intersect, as shown in Fig. 6.2. To obtain it, the first thing to do is to identify these vertices and the arcs that connect them to each other. The process is as follows: consider every pair of circles; find their two intersection points, which form two candidate vertices; if a candidate vertex is inside all other circles, then it is indeed a vertex on the boundary of intersection region; sort the result vertices in a

Algorithm 3 Process of Combination Operation

Input: T_{tsp} and r
Output: Tour after Combination Operation T_{com}

$T \leftarrow \varnothing$, $T_{com} \leftarrow \varnothing$
$n \leftarrow$ Number of entry points in T_{tsp}
for $i \leftarrow 1$ to $n-1$ **do**
    $T \leftarrow T \cup T_{tsp}[i]$
    $D_c \leftarrow$ Welzl's algorithm of T
    if Radius of $D_c \leq r$ **then**
        $D_p \leftarrow D_c$
    else
        $T \leftarrow T \setminus T_{tsp}[i]$
        Obtain intersection region of disks whose entry points in T and associate it with center of D_p
        Add center of D_p to T_{com}
        $T \leftarrow \varnothing$
        $T \leftarrow T_{tsp}[i]$
        $D_p \leftarrow$ Welzl's algorithm of T
    end if
end for
Add center of D_p to T_{com}
Add $T_{com}[1]$ to T_{com}
return T_{com}

consistent order around the intersection region, i.e. clockwise or counterclockwise. (Notice that there may be more than three circles coincide at one vertex, the result vertices need to be set unique if multiple vertices coincide.) Then the circular arcs on the boundary can be determined by the ordered vertices and disk radius. The intersection region can be finally represented by a list of these circular arcs.

Theorem 6.1 ([66]). *Given a set of $n \geq 2$ possibly intersecting disks, with $r > 0$, the tour length of T_{com} is no longer than that of T_{tsp},*

$$length(T_{com}) \leq length(T_{tsp})$$

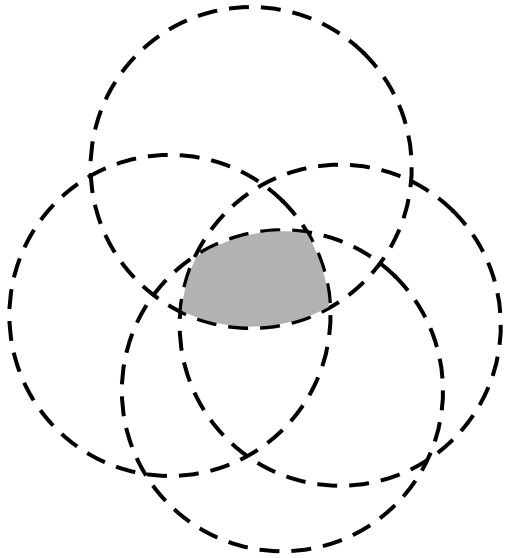


Figure 6.2: Intersection region of the overlapped disks (in grey). The entry point lies within the intersection region.

6.3.3 Alternating Iterative Algorithm for TSPN

Through the previous step, the entry points are all located at either the center of the disjoint disk or the intersection region of overlapped disks. In this step, the authors design an Alternating Iterative Algorithm (AIA) to find alternative entry points of the disks that may further shorten the TSPN tour length.

The input is the tour T_{com} after Combination Operation. The main idea of AIA is that it utilizes the intersection points between the tour path and the boundaries of the intersection regions or disjoint disks. For an intersection region, the boundary is an ordered list of circular arcs that enclose the region. For a disjoint disk, the boundary is the circle that encloses the disk. Each intersection region or disjoint disk is associated with an entry point in T_{com}.

The algorithm firstly selects the first intersection point on each boundary and connects these points to form a new tour; then in the next step, the new tour path is used to compute new intersection points. Updating the entry points with the second intersection point on each boundary, the algorithm forms a new tour path again; then repeats the former steps. For each boundary, the entry point alternates between the first intersection point and the second intersection point in successive steps. In order to stop the process, this algorithm

Algorithm 4 Process of Alternating Iterative Algorithm

Input: T_{com}
Output: ETSPN tour T_{aia} obtained by AIA
1: $L_p \leftarrow$ Tour length of T_{com}
2: OddStepTSPN (T_{com})
3: $L_n \leftarrow$ Tour length of T_{com}
4: $i \leftarrow 2$
5: **while** L_p - $L_n > \delta$ **do**
6: $L_p \leftarrow L_n$
7: **if** i mod $2 = 0$ **then**
8: EvenStepTSPN (T_{com})
9: **else**
10: OddStepTSPN (T_{com})
11: **end if**
12: $L_n \leftarrow$ Tour length of T_{com}
13: $i \leftarrow i + 1$
14: **end while**
15: $T_{aia} \leftarrow T_{com}$
16: **return** T_{aia}

introduces a termination condition parameter δ. If the reduction between the previous step tour length (L_p) and the next step tour length (L_n) is less than δ, then AIA considers that there is no more improvement of the length and stops. The main process is described in Algorithm 4. It calls Algorithm 5 and Algorithm 6.

Theorem 6.2. *Given a set of $n \geq 2$ possibly intersecting disks, with $r > 0$, let T_{com} and T_{aia} denote the tours produced after combination operation and the AIA, respectively. Then the tour length of T_{aia} is no longer than that of T_{com},*

$$length(T_{aia}) \leq length(T_{com})$$

Proof. The process of AIA starts from the T_{com}. By triangle inequality, the tour length of each step in AIA is no longer than that of previous step. This property can be seen from Fig. 6.1c and Fig. 6.1d. □

Algorithm 5 OddStepTSPN (T_{com})

Input: T_{com}
Output: New entry points of T_{com}

$T \leftarrow \varnothing$
$n \leftarrow$ Number of entry points in T_{com}
for $i \leftarrow 1$ to $n-1$ **do**
    $L \leftarrow$ Line segment from $T_{com}[i]$ to $T_{com}[i+1]$
    $Boundary \leftarrow$ Region Boundary associated with $T_{com}[i+1]$
    $Num \leftarrow$ Number of intersection points between L and $Boundary$
    if $Num \geq 1$ **then**
        $P_1 \leftarrow$ First intersection point between L and $Boundary$
        Add P_1 to T
    else
        Add $T_{com}[i+1]$ to T
    end if
end for
Add $T[1]$ to T
$T_{com} \leftarrow T$

Algorithm 6 EvenStepTSPN (T_{com})

Input: T_{com}
Output: New entry points of T_{com}

$T \leftarrow \varnothing$
$n \leftarrow$ Number of entry points in T_{com}
for $i \leftarrow 1$ to $n-1$ **do**
    $L \leftarrow$ Line segment from $T_{com}[i]$ to $T_{com}[i+1]$
    $Boundary \leftarrow$ Region Boundary associated with $T_{com}[i]$
    $Num \leftarrow$ Number of intersection points between L and $Boundary$
    if $Num = 2$ **then**
        $P_2 \leftarrow$ Second intersection point between L and $Boundary$
        Add P_2 to T
    else
        Add $T_{com}[i]$ to T
    end if
end for
Add $T[1]$ to T
$T_{com} \leftarrow T$

6.3.4 Compute the Headings for Entry Points to Form a DTSP

Once the set of entry points are determined, the Dubins Traveling Salesman Problem with Neighborhoods (DTSPN) is reduced into a Dubins Traveling Salesman Problem (DTSP)

that finds the shortest Dubins path connecting all these entry points. All categories of algorithms for DTSP as mentioned in Section 2.2.2 can be applied in this problem, and have their advantages and disadvantages for this application. *Category (1)* algorithms can utilize the visiting order of the disks which is determined by the previous step in this chapter. They need only to determine the heading at each entry point, in order to calculate the corresponding Dubins path. That saves the computing efforts. But these algorithms may perform worse when the distances between entry points are very small relative to the turning radius. *Category (2)* can perform well when the distances between entry points are very small relative to the turning radius, but need more computing effort because the visiting order needs to be determined again. The genetic algorithms can find an approximate solution efficiently, but have no proven performance guarantees.

Since the entry points in many practical robot-trailer applications are spaced relatively far apart, in this section the authors adopt the algorithms in *Category (1)*. One of the well-established techniques is called Alternating Algorithm (AA) [84]. By using AA, the optimal ordered entry points T_{aia} are connected by straight line segments, after which the odd-numbered edges along with respective headings are retained; the even-numbered edges are replaced by the Dubins paths. Since the solution of AA depends on the choice of starting entry point, there are many possible solutions for AA: (1) for a tour with even number of edges, starting AA from different entry points gets two sets of parities of edge numbers, thus has two kinds of possible solutions; (2) for a tour with odd number of edges, AA requires to replace not only all even-numbered edges, but also the last odd-numbered edge by the Dubins path. There is one special entry point that connects two Dubins paths, and starting AA from different entry points will obtain different special entry points. Thus, if the number of entry points is N, there are N possible solutions. If the entry points are visited in a reversed order, the lengths of Dubins paths that connect to the special entry point will change as the headings change, which account for another set of N possible solutions. Thus the total number of possible solutions in such case is $2N$.

In order to obtain the minimum cost tour, one option is to generate all the possible solutions and see which one is the shortest, as mentioned in [56], but it requires $O(n^2)$ time complexity. In this chapter, the authors propose a more efficient method to obtain the minimum cost tour, and call it Improved Alternating Algorithm (IAA). The method follows the visiting order determined by the previous section, and consists of the following steps:

1. For each entry point, compute two candidate headings: one pointing to the next entry point, and the other pointed from the previous entry point. Construct a cluster that contains two nodes. The cluster corresponds to the entry point, and each node corresponds to a candidate heading. (Note that the last cluster and the first cluster are identical, because the last entry point in T_{aia} is also the first entry point.)

2. Create arcs from each node in each cluster to all nodes in the successive cluster. The arc costs equal to the corresponding Dubins distances between two nodes.

3. For the above directed graph, apply Dijkstra's Algorithm to find the shortest path from the first node in the first cluster to the first node in the last cluster. Then repeat Dijkstra's Algorithm to find the shortest path from the second node in the first cluster to the second node in the last cluster.

4. Pick the path with minimum cost between the two result paths in step 3.

6.4 Performance Analysis

Let $\mathcal{P}$ be a set of n waypoints in a compact region $\mathcal{Q}\subset\mathbb{R}^2$ and $\mathcal{P}_n$ be the collection of all waypoint sets with cardinality n. Let $ETSP(\mathcal{P})$ denotes the shortest tour length of Euclidean Traveling Salesman Problem over $\mathcal{P}$, and $DTSP_\rho(\mathcal{P})$ denotes the shortest tour length of Dubins Traveling Salesman Problem over $\mathcal{P}$ with minimum turning radius ρ. Let $\mathcal{D}$ be a set of n equal disks with radius of r in a compact region $\mathcal{Q}\subset\mathbb{R}^2$ and $\mathcal{D}_n$ be the collection of all disk sets with cardinality n. Let $ETSPN(\mathcal{D})$ denote the shortest tour length of Euclidean Traveling Salesman Problem with Neighborhoods over disks $\mathcal{D}$, and $DTSPN_\rho(\mathcal{D})$

denote the shortest tour length of Dubins Traveling Salesman Problem with Neighborhoods over disks $\mathcal{D}$ with minimum turning radius ρ. Let $L_{AIA}(\mathcal{D})$ denote the tour length of ETSPN over disks $\mathcal{D}$ after the Alternating Iterative Algorithm. Let $L_{AA,\rho}(\mathcal{P})$ denote the tour length of DTSP over waypoint set $\mathcal{P}$ as given by Alternating Algorithm with minimum turning radius ρ. Let $L_{IAA,\rho}(\mathcal{P})$ denote the tour length of DTSP over waypoint set $\mathcal{P}$ as given by Improved Alternating Algorithm with minimum turning radius ρ. Let $L_{AIA+IAA,\rho}(\mathcal{D})$ denote the tour length of DTSPN over disks $\mathcal{D}$ after the Alternating Iterative Algorithm and Improved Alternating Algorithm with minimum turning radius ρ.

Theorem 6.3. *(Upper bound on ETSPN for disks) Given a compact region $\mathcal{Q}$, there exists a finite constant $\beta(\mathcal{Q})$, for $\mathcal{D} \in \mathcal{D}_n$, such that*

$$ETSPN(\mathcal{D}) \leq \beta(\mathcal{Q})\sqrt{n}$$

Proof. Assuming that $\mathcal{P}$ is the set of entry points in the optimal ETSPN tour over $\mathcal{D}$, it can be seen that the optimal ETSPN tour over $\mathcal{D}$ is also the optimal ETSP tour over $\mathcal{P}$. Recall the fact for ETSP in [86], given a compact region $\mathcal{Q}$, there is finite constant $\beta(\mathcal{Q})$, for $\mathcal{P} \in \mathcal{P}_n$, such that $ETSP(\mathcal{P}) \leq \beta(\mathcal{Q})\sqrt{n}$. Therefore, the statement is proved. □

Theorem 6.4. *(Performance of AIA) Given $n \geq 2$ possible intersecting disks $\mathcal{D}$ with $r > 0$,*

$$ETSPN(\mathcal{D}) \leq L_{AIA}(\mathcal{D}) \leq ETSPN(\mathcal{D}) + 2nr$$

Proof. It is fairly easy to see that $ETSPN(\mathcal{D}) \leq L_{AIA}(\mathcal{D})$. For the second inequality, an extended tour can be obtained by going along the optimal ETSPN tour and making a detour of length at most $2r$ to visit the center of each disk, when the ETSPN tour enters the disk. The length of optimal ETSP tour over these centers is less than that of the extended tour. Furthermore, from Theorem 6.1 and Theorem 6.2, the tour length of AIA is less than the length of the extended tour. □

Lemma 6.5. *(Theorem 3.4 in [50]) Given two configurations* $X = (x, y, \theta)$ *and* $X' = (x', y', \theta')$*, and* $\rho > 0$ *for Dubins vehicle, there exists a constant* $\kappa \in [2.657, 2.658]$*, such that*

$$\mathcal{C}_\rho(X, X') \leq \sqrt{(x - x')^2 + (y - y')^2} + \kappa\pi\rho$$

Lemma 6.6. *(Lemma 3.5 in [50]) Given* $\mathcal{P} \in \mathcal{P}_n$ *with* $n \geq 2$ *and* $\rho > 0$ *for Dubins vehicle,*

$$L_{AA,\rho}(\mathcal{P}) \leq ETSP(\mathcal{P}) + \kappa\lceil\frac{n}{2}\rceil\pi\rho$$

Theorem 6.7. *(Bounds on the DTSPN for disks) Given any set of* $n \geq 2$ *possible intersecting disks* $\mathcal{D}$ *with* $r > 0$*, and* $\rho > 0$ *for Dubins vehicle,*

$$ETSPN(\mathcal{D}) \leq DTSPN_\rho(\mathcal{D}) \leq ETSPN(\mathcal{D}) + \kappa\lceil\frac{n}{2}\rceil\pi\rho$$

Proof. Given the optimal DTSPN tour over $\mathcal{D}$, a feasible ETSPN tour can be formed by connecting the entry point in each disk with Euclidean path. The new ETSPN tour is no longer than the optimal DTSPN tour and no shorter than the optimal ETSPN tour, thus $ETSPN(\mathcal{D}) \leq DTSPN_\rho(\mathcal{D})$. Conversely, given the optimal ETSPN tour over $\mathcal{D}$, a feasible DTSPN tour can be formed by connecting the entry point in each disk with Alternating Algorithm, whose tour length will be no shorter than that of the optimal DTSPN tour. While from Lemma 6.6 for the Alternating Algorithm, the length of the feasible DTSPN tour is no longer than $ETSPN(\mathcal{D}) + \kappa\lceil\frac{n}{2}\rceil\pi\rho$. Thus, $DTSPN_\rho(\mathcal{D}) \leq ETSPN(\mathcal{D}) + \kappa\lceil\frac{n}{2}\rceil\pi\rho$. □

Theorem 6.8. *(Performance of Improved Alternating Algorithm for DTSP) Given* $\mathcal{P} \in \mathcal{P}_n$ *with* $n \geq 2$ *and* $\rho > 0$ *for Dubins vehicle,*

$$L_{IAA,\rho}(\mathcal{P}) \leq L_{AA,\rho}(\mathcal{P})$$

Proof. Considering different starting waypoints in an order, there are two possible headings of each waypoint by AA, which are included in the node set of IAA. The possible Dubins paths

between successive waypoints in AA are also included in the arc set of IAA. If the waypoints are visited in a reversed order, there are another two possible headings of each waypoint in AA, which are opposite to the two headings of each waypoint in the original order. By changing the headings of waypoints in the reversed tour to their opposite directions, the reversed tour can be transformed into a tour in the original order, without changing the tour length. Since the opposite directions are also included in the node set of IAA, the directed graph of IAA in the original order is sufficient to find the minimum cost AA tour. The Dijkstra's Algorithm can guarantee to find a shortest path from one node to another in a graph. Therefore, if the result of AA is the shortest path, then it is also the result of IAA; otherwise, it is longer than the result of IAA.

Notice that, AA requires that the straight line segment and Dubins path are connected in an alternate way (except the last edge in odd number of edges). From the representation of IAA, there is no such constraints, which gives the algorithm more freedom to find a shorter solution than the method in [56] of testing all possible AA solutions. □

Theorem 6.9. *(Upper bound on the performance of AIA with IAA) Given any set of* $n \geq 2$ *possible intersecting disks* $\mathcal{D}$ *with* $r > 0$*, and* $\rho > 0$ *for Dubins vehicle,*

$$L_{AIA+IAA,\rho}(\mathcal{D}) \leq ETSPN(\mathcal{D}) + 2nr + \kappa\lceil\frac{n}{2}\rceil\pi\rho$$

Proof. From Lemma 6.6 and Theorem 6.8, $L_{AIA+IAA,\rho}(\mathcal{D})$ is no longer than $L_{AIA}(\mathcal{D}) + \kappa\lceil\frac{n}{2}\rceil\pi\rho$. From Theorem 6.3, the tour length $L_{AIA}(\mathcal{D})$ is no longer than $ETSPN(\mathcal{D}) + 2nr$. □

Note. *For* $\mathcal{D} \in \mathcal{D}_n$*, Theorem 6.3 implies that* $ETSPN(\mathcal{D})$ *belongs to* $O(\sqrt{n})$ *and Theorem 6.7 implies that* $DTSPN(\mathcal{D})$ *belongs to* $O(n)$ *and* $\Omega(\sqrt{n})$*. Furthermore, Theorem 6.9 implies that* $L_{AIA+IAA,\rho}(\mathcal{D})$ *belongs to* $O(n)$*.*

For the time complexity, the running time of the new version Combination Operation is upper bounded by $O(n^2)$. The running time of obtaining the intersection region for

Table 6.1: Experiment Parameters

Parameter	Symbol	Value
Minimum Turning Radius	ρ	1.0
Disk Radius	r	1.0
Termination Condition Parameter	δ	0.001

overlapped disks is $O(n^2)$. The running time of Alternating Iterative Algorithm is $O(n)$. The running time of Improved Alternating Algorithm is $O(n\log(n))$. Taking LKH to be the ETSP solver, the average running time of LKH is approximately $O(n^{2.2})$ [80].

6.5 Numerical Experiment

The performance of the proposed algorithm is examined for both high density case (most disks are overlapped) and low density case (most disks are disjoint). The first experiment (high density case) is implemented on a 10×10 square. The other (low density case) is on a 40×40 square. In both cases, the positions of disk center are generated randomly, and the numbers of disks varies from 5 to 50 with increment 5. For each given number of disks, the authors randomly generate 30 samples, and compute the average length of these samples. The experiment parameters are listed in Table 6.1. The performance of the proposed algorithm is compared step by step. The final result is compared with Dubins Traveling Salesman Problem (DTSP) over disk centers, which simulates the case that does not take advantage of the sensor scope to reduce the tour length. The DTSP result is computed by ETSP + IAA. The result of ETSP + CO + IAA is also provided to compare the effects of Combination Operation (CO) and Alternating Iterative Algorithm (AIA) in DTSPN process.

Fig. 6.3 shows the performance of different steps in high density case. Fig. 6.4 is an instance in this case. Fig. 6.5 shows the performance of different steps in low density case. Fig. 6.6 is an instance in this case. As shown in Fig. 6.3 and Fig. 6.5, the proposed algorithm in this chapter performs much better than DTSP over disk centers, in both low

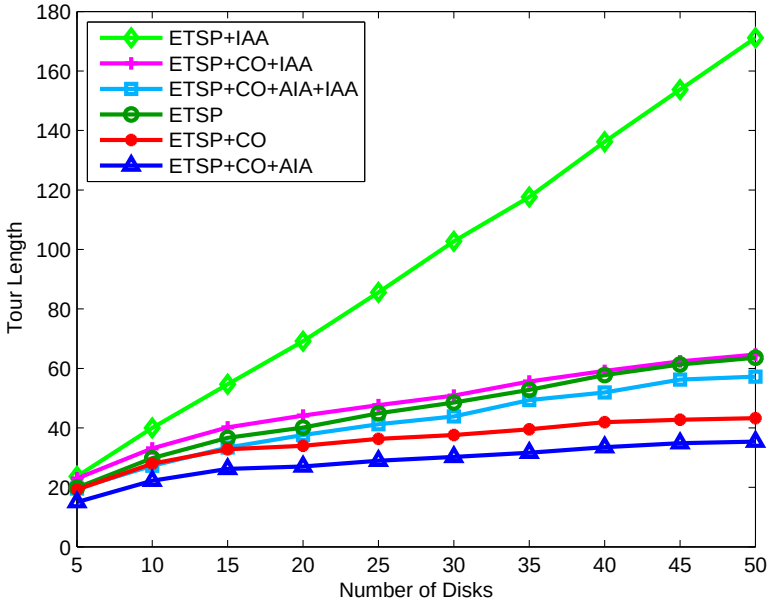
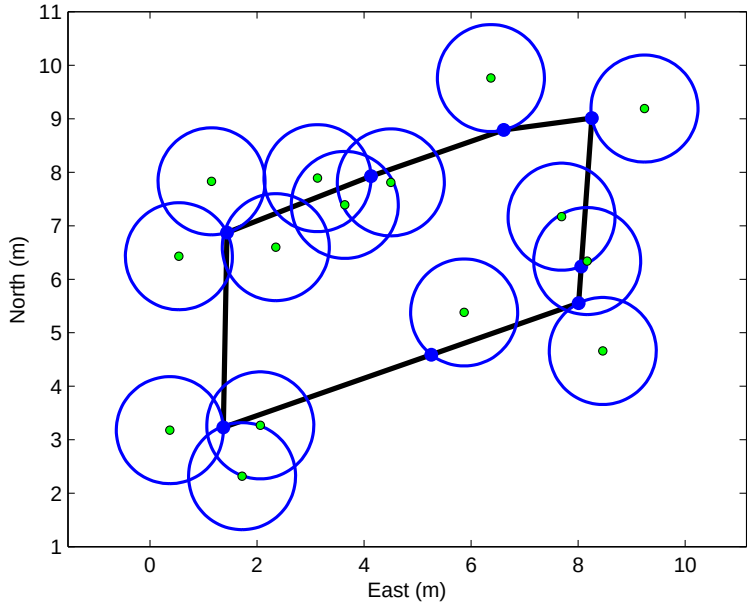


Figure 6.3: 10x10 square (high density) case comparison.

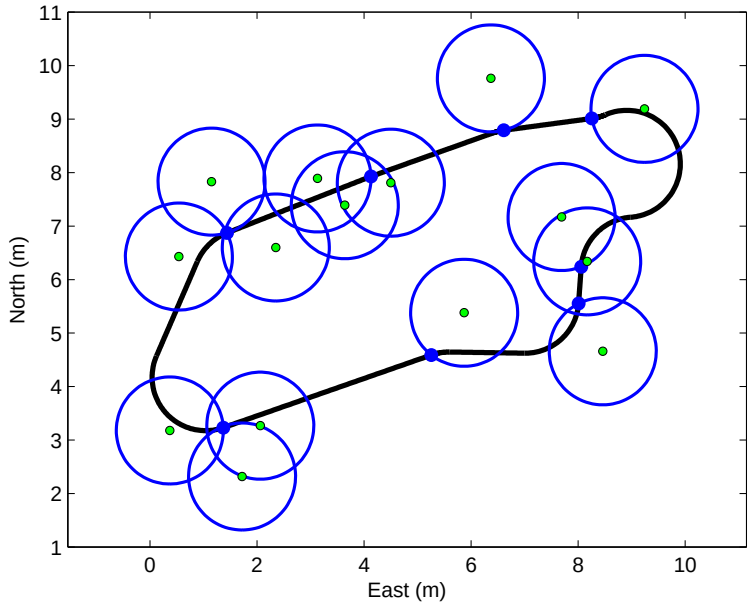
density case and high density case. It reduces the tour length by taking the sensor scope into consideration. Especially, when the disks are overlapped frequently, the Combination Operation greatly reduces the number of entry points for the last step to calculate Dubins paths. For the low density case that most disks are disjoint, Alternating Iterative Algorithm can further reduce the TSPN tour length after the Combination Operation, finally reduce the total DTSPN tour length.

6.6 Practical Experiment

In this section, a practical experiment is established to test performance of the proposed algorithm based on practical parameters of a robot-trailer application. The towing robot is a modified Segway® Robotic Mobility Platform (RMP) 440. Position information is provided to centimeter accuracy by a commercial integrated differential Global Positioning System (GPS) / Inertial Navigation System (INS) solution, the Novatel® SPAN system with the high precision Honeywell® HG1700 AG58 gyro. The towed sensor is a Geonics® EM61-MK2 metal detector. The location of sensor center is determined by geometric calculations based



(a)



(b)

Figure 6.4: Instances for high density case with 15 disks. (a) Euclidean Traveling Salesman Problem with Neighborhoods (b) Dubins Traveling Salesman Problem with Neighborhoods.

on tow bar hitch angles and the fixed tow bar lengths. The test field is about 30×90 m^2 and 20 waypoints are randomly placed within the test field. The towed sensor must cover all

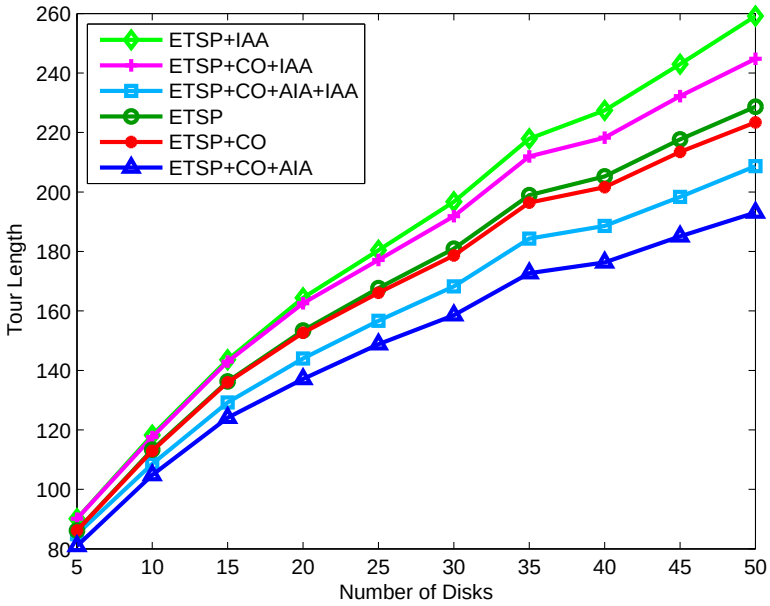
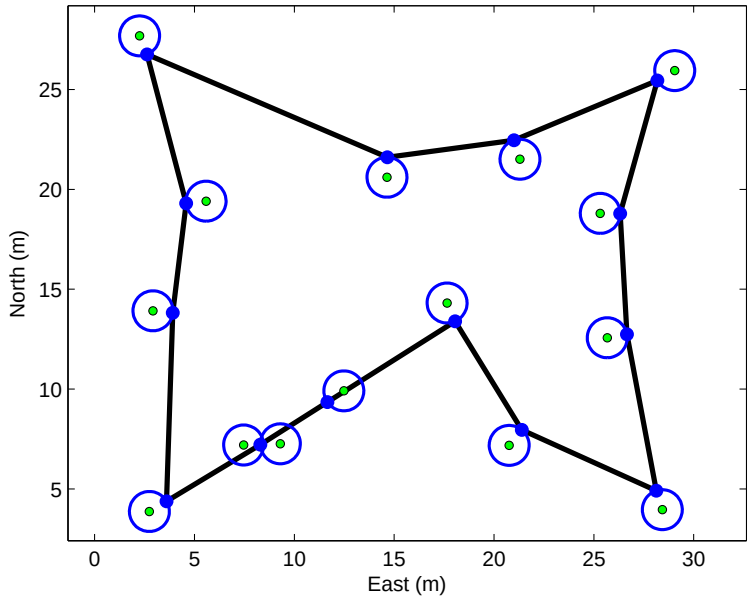


Figure 6.5: 40x40 square (low density) case comparison.

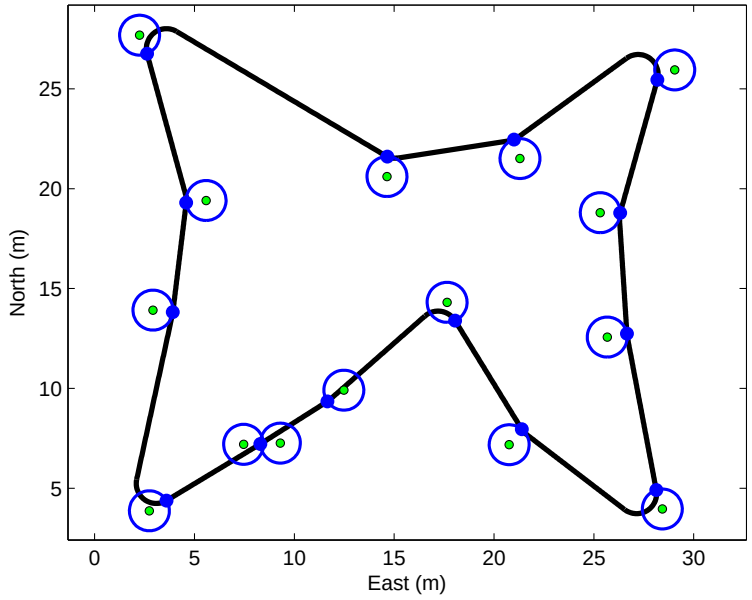
waypoints during traversal. The minimum turning radius of the system is 4 m. The sensor width is 2 m, thus the disks whose center at waypoints should have radius of 1 m. Fig. 6.7 shows the practical experiment result. As shown in that figure, all disks are traversed by the tour path, which means all waypoints are covered by the towed sensor. Therefore, the DTSPN model is feasible for this robot-trailer application.

6.7 Summary

In this chapter, the author takes the sensor scope into consideration for the path planning problem and model it as a Dubins Traveling Salesman Problem with Neighborhoods (DTSPN) where the neighborhoods are represented by disks. A new algorithm is proposed for solving the DTSPN, in which the visiting sequence of the neighborhoods and the corresponding entry points are firstly determined by a TSPN algorithm, and then the task of finding the headings for the entry points is formed as a DTSP. The advantage is that the DTSPN can be divided into two subproblems, TSPN and DTSP, then one can directly exploit the existing methods on TSPN and DTSP. The theoretical and numerical studies show



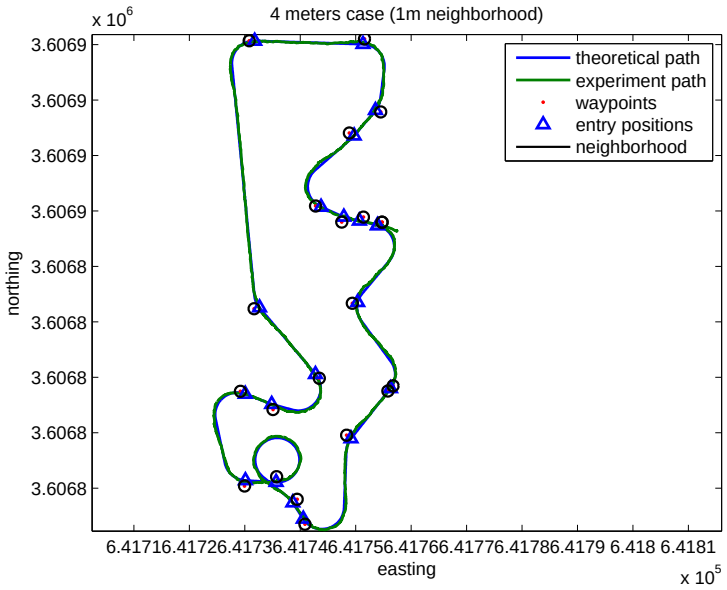
(a)



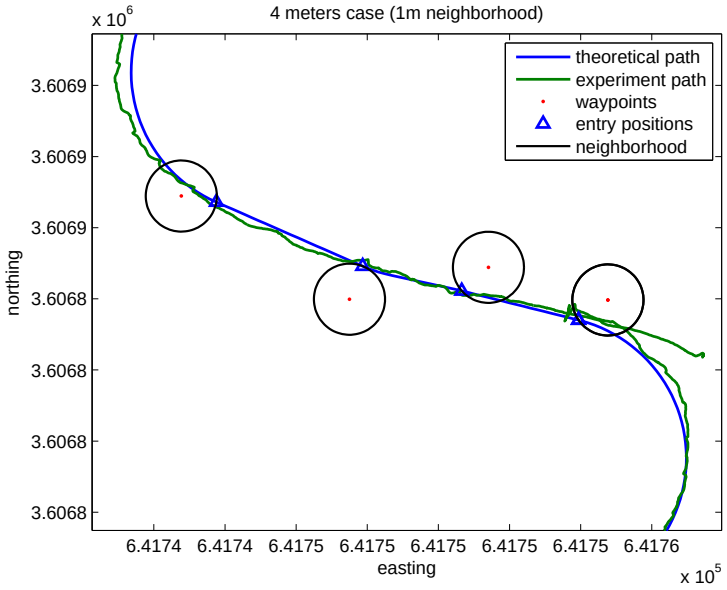
(b)

Figure 6.6: Instances for low density case with 15 disks. (a) Euclidean Traveling Salesman Problem with Neighborhoods (b) Dubins Traveling Salesman Problem with Neighborhoods.

that the proposed algorithm performs well in both disjoint disks case and overlapped disks case. The practical experiment shows that the DTSPN model is feasible for the robot-trailer



(a)



(b)

Figure 6.7: The green line represents the positions of the trailer center taken during the test. The waypoints and disk regions are represented by red and black circles respectively. The desired entry point of each disk is represented by blue triangle. (a) Practical experiment result. (b) One portion of the path.

application. This work can be extended in many directions. While the author focuses on a robot-trailer application, the proposed algorithm of DTSPN could be applied to any Dubins vehicle that has similar mission requirements, such as a fixed-wing Unmanned Aerial Vehicle (UAV) in a multi-target surveillance mission, or a car-like mobile robot that collects data in Wireless Sensor Networks (WSNs).

Chapter 7

Conclusion

In this concluding chapter, the author summarizes the contributions of this dissertation, and offers a few comments on future work.

7.1 Review of Contributions

In Chapter 3, the author presents an optimization approach to minimize the number of turns of autonomous vehicles in coverage path planning. For complex polygonal fields, the problem is reduced to finding the optimal decomposition of the original field into simple subfields. The optimization criterion is minimization of the sum of widths of these decomposed subfields. A new algorithm is designed based on a multiple sweep line decomposition. The time complexity of the proposed algorithm is $O(n^2 \log n)$. Experiments show that the proposed algorithm can provide nearly optimal solutions very efficiently when compared against recent state-of-the-art. The proposed algorithm can be applied for both convex and non-convex fields.

In Chapter 4, the author presents an optimization approach that takes the vehicle's characteristics into account to minimize the non-working travel of the robots in coverage path planning. The aim is to minimize the cost on a fixed number of turns, by finding the optimal traversal sequence of parallel tracks for the surveyed field. The author firstly presents a novel traversal pattern of parallel tracks for a single convex field, then extends the proposed traversal pattern to connect with the decomposition algorithm, providing a complete coverage path planning method for non-convex fields. Experiments show that the proposed method can provide feasible solutions and the total wasted distance can be greatly

reduced for both single convex field and multiple decomposed fields, when compared against classical boustrophedon path or recent state-of-the-art.

In Chapter 5, the author studies the traveling salesman problem. Taken the vehicle's characteristics into account, the problem is modeled as a Dubins Traveling Salesman Problem (DTSP). A genetic algorithm is designed to find the shortest path and the performance is evaluated in numerical study. The experiments show that the proposed algorithm can perform better than the well-known alternating algorithm and random headings algorithm, in both low waypoint density and high waypoint density situations.

In Chapter 6, the author takes the physical size of the actual sensors into consideration for the path planning. The trailer equipped with sensors collects data among a collection of waypoint neighborhoods. The concept of a neighborhood is used to model the size of sensor scope. The problem is modeled as a Dubins Traveling Salesman Problem with Neighborhoods (DTSPN), where the neighborhoods are represented by disks. The author uses a two-stage approach to solve the problem: (1) design a new algorithm for the TSPN to search the optimal visiting sequence and entry positions; (2) design a new algorithm for the Dubins vehicle to determine the heading at each entry position. The time complexity analysis shows that the first stage runs in $O(n^2)$ and the second stage runs in $O(n \log(n))$. The theoretical and numerical studies show that the proposed approach can perform very well for both disjoint and overlapped disks cases. The practical experiment shows that the model is feasible for the robot-trailer application.

7.2 Future Work

Several aspects of the path planning module can be improved in future revisions. The algorithms developed in this dissertation are suitable for single robot applications. A more general algorithm can be designed for the situation in which multiple robots can work simultaneously to finish a task. Consider the vehicle's characteristics, the first stage of the task (coverage path planning) can be modeled as a Generalized Vehicle Routing Problem

(GVRP) and the second stage of the task (waypoints traversal planning) can be modeled as a Multiple Traveling Salesman Problem (MTSP) for Dubins Vehicles. Both GVRP and MTSP are challenging and emerging research areas, especially for Dubins vehicles.

Another improvement that can be made in future revisions is for the obstacle avoidance. The algorithms of Traveling Salesman Problems developed in this dissertation make assumptions that there is no obstacles in an area where the system is turning. A more robust algorithm should be designed to take such cases into consideration when planning an optimal route. Also, the path in the clearance region around obstacles and boundary can be taken into consideration when planning the optimal route.

Bibliography

[1] W. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, Seoul, Korea, May 2001, pp. 27–32.

[2] Y. Li, H. Chen, M. Joo Er, and X. Wang, "Coverage path planning for UAVs based on enhanced exact cellular decomposition method," *Mechatronics*, vol. 21, no. 5, pp. 876–885, Aug. 2011.

[3] D. W. Hodo, D. M. Bevly, J. Y. Hung, S. Millhouse, and B. Selfridge, "Optimal path planning with obstacle avoidance for autonomous surveying," in *Proc. Annu. Conf. IEEE Ind. Electron. Soc.*, Glendale, AZ, Nov. 2010, pp. 1577–1583.

[4] D. D. Bochtis, S. G. Vougioukas, and H. W. Griepentrog, "A mission planner for an autonomous tractor," *Trans. Amer. Soc. Agricultural and Biosystem Eng.*, vol. 52, no. 5, pp. 1429–1440, 2009.

[5] O. of the Secretary of Defense, "Report of the defense science board task force on unexploded ordnance," Nov. 2003.

[6] D. W. Hodo, "Development of an autonomous mobile robot-trailer system for UXO detection," Master's thesis, Auburn University, 2007.

[7] X. Yu and J. Y. Hung, "A genetic algorithm for the Dubins traveling salesman problem," in *Proc. IEEE Int. Symp. Ind. Electron.*, Hangzhou, China, May 2012, pp. 1256–1261.

[8] ——, "Optimal path planning for an autonomous robot-trailer system," in *Proc. Annu. Conf. IEEE Ind. Electron. Soc.*, Montreal, Canada, Oct. 2012, pp. 2762–2767.

[9] N. Sudha and A. Mohan, "Hardware-efficient image-based robotic path planning in a dynamic environment and its FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 5, pp. 1907–1920, May 2011.

[10] R. Cowlagi and P. Tsiotras, "Multiresolution motion planning for autonomous agents via wavelet-based cell decompositions," *IEEE Trans. Syst., Man, Cybern. B*, vol. 42, no. 5, pp. 1455–1469, Oct. 2012.

[11] L. S. Kei, K. Sridharan, and T. Srikanthan, "Hardware-efficient schemes for logarithmic approximation and binary search with application to visibility graph construction," *IEEE Trans. Ind. Electron.*, vol. 51, no. 6, pp. 1346–1348, Dec. 2004.

[12] K. Sridharan and T. Priya, "The design of a hardware accelerator for real-time complete visibility graph construction and efficient FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 52, no. 4, pp. 1185–1187, Aug. 2005.

[13] Y. Kang, H. Kim, S.-H. Ryu, N. L. Doh, Y. Oh, and B.-J. You, "Dependable humanoid navigation system based on bipedal locomotion," *IEEE Trans. Ind. Electron.*, vol. 59, no. 2, pp. 1050–1060, Feb. 2012.

[14] L. Vachhani and K. Sridharan, "Hardware-efficient prediction-correction-based generalized-Voronoi-diagram construction and FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 55, no. 4, pp. 1558–1569, Apr. 2008.

[15] L. Vachhani, K. Sridharan, and P. Meher, "Efficient FPGA realization of CORDIC with application to robotic exploration," *IEEE Trans. Ind. Electron.*, vol. 56, no. 12, pp. 4915–4929, Dec. 2009.

[16] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, Jan. 2014.

[17] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[18] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, 1999, pp. 473–479 vol.1.

[19] E. Besada-Portas, L. de la Torre, J. De La Cruz, and B. de Andrés-Toro, "Evolutionary trajectory planner for multiple UAVs in realistic scenarios," *IEEE Trans. Robot.*, vol. 26, no. 4, pp. 619–634, Aug. 2010.

[20] C.-C. Tsai, H.-C. Huang, and C.-K. Chan, "Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, pp. 4813–4821, Oct. 2011.

[21] C.-T. Cheng, K. Fallahi, H. Leung, and C. Tse, "A genetic algorithm-inspired UUV path planner based on dynamic programming," *IEEE Trans. Syst., Man, Cybern. C*, vol. 42, no. 6, pp. 1128–1134, Nov. 2012.

[22] V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 132–141, Feb. 2013.

[23] D. Zhu, H. Huang, and S. Yang, "Dynamic task assignment and path planning of multi-AUV system based on an improved self-organizing map and velocity synthesis method in three-dimensional underwater workspace," *IEEE Trans. Cybern.*, vol. 43, no. 2, pp. 504–514, Apr. 2013.

[24] H. Rezaee and F. Abdollahi, "A decentralized cooperative control scheme with obstacle avoidance for a team of mobile robots," *IEEE Trans. Ind. Electron.*, vol. 61, no. 1, pp. 347–354, Jan. 2014.

[25] H. Kim and B. K. Kim, "Online minimum-energy trajectory planning and control on a straight-line path for three-wheeled omnidirectional mobile robots," *IEEE Trans. Ind. Electron.*, vol. 61, no. 9, pp. 4771–4779, Sept. 2014.

[26] B. Shiu and C. Lin, "Design of an autonomous lawn mower with optimal route planning," in *Proc. IEEE Int. Conf. Ind. Technology*, Chengdu, China, Apr. 2008, pp. 1–6.

[27] J. S. Oh, Y.-H. Choi, J.-B. Park, and Y. Zheng, "Complete coverage navigation of cleaning robots using triangular-cell-based map," *IEEE Trans. Ind. Electron.*, vol. 51, no. 3, pp. 718–726, June 2004.

[28] S. X. Yang and C. Luo, "A neural network approach to complete coverage path planning," *IEEE Trans. Syst., Man, Cybern. B*, vol. 34, no. 1, pp. 718–724, Feb. 2004.

[29] G. Kim and W. Chung, "Tripodal schematic control architecture for integration of multi-functional indoor service robots," *IEEE Trans. Ind. Electron.*, vol. 53, no. 5, pp. 1723–1736, Oct 2006.

[30] C. Luo and S. X. Yang, "A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments," *IEEE Trans. Neural Netw.*, vol. 19, no. 7, pp. 1279–1298, July 2008.

[31] C.-H. Kuo, H.-C. Chou, and S.-Y. Tasi, "Pneumatic sensor: A complete coverage improvement approach for robotic cleaners," *IEEE Trans. Instrum. Meas.*, vol. 60, no. 4, pp. 1237–1256, Apr. 2011.

[32] E. Acar, H. Choset, and J. Lee, "Sensor-based coverage with extended range detectors," *IEEE Trans. Robot.*, vol. 22, no. 1, pp. 189–198, Feb. 2006.

[33] D. Anisi, P. Ogren, and X. Hu, "Cooperative minimum time surveillance with multiple ground vehicles," *IEEE Trans. Autom. Control*, vol. 55, no. 12, pp. 2679–2691, Dec. 2010.

[34] A. Yazici, G. Kirlik, O. Parlaktuna, and A. Sipahioglu, "A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints," *IEEE Trans. Cybern.*, vol. 44, no. 3, pp. 305–314, Mar. 2014.

[35] A. Das, M. Diu, N. Mathew, C. Scharfenberger, J. Servos, A. Wong, J. S. Zelek, D. A. Clausi, and S. L. Waslander, "Mapping, planning, and sample detection strategies for autonomous exploration," *J. Field Robotics*, vol. 31, no. 1, pp. 75–106, Jan. 2014.

[36] L. Paull, S. Saeedi, M. Seto, and H. Li, "Sensor-driven online coverage planning for autonomous underwater vehicles," *IEEE/ASME Trans. Mechatronics*, vol. 18, no. 6, pp. 1827–1838, Dec. 2013.

[37] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," *J. Field Robotics*, vol. 26, no. 8, pp. 651–668, Aug. 2009.

[38] H. Choset, "Coverage for robotics–a survey of recent results," *Ann. Math. Artificial Intell.*, vol. 31, no. 1, pp. 113–126, 2001.

[39] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robot. Auton. Syst.*, vol. 61, no. 12, pp. 1258–1276, 2013.

[40] J. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[41] H. Choset and P. Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and Service Robotics*. London, United Kingdom: Springer, 1998, pp. 203–209.

[42] C. Fang and S. Anstee, "Coverage path planning for harbour seabed surveys using an autonomous underwater vehicle," in *Proc. IEEE OCEANS*, Sydney, Australia, May 2010, pp. 1–8.

[43] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Voronoi diagrams: The post office problem," in *Computational Geometry: Algorithms and Applications*, 3rd ed. Berlin, Germany: Springer-Verlag, 2008, ch. 7, pp. 147–172.

[44] J. Jin and L. Tang, "Optimal coverage path planning for arable farming on 2D surfaces," *Trans. Amer. Soc. Agricultural and Biosystem Eng.*, vol. 53, no. 1, pp. 283–295, 2010.

[45] A. Rankin, C. Crane, A. Armstrong, A. Nease, and H. Brown, "Autonomous path planning navigation system used for site characterization," in *Proc. SPIE 10th Annu. Symp. AeroSense*, vol. 2738, Orlando, FL, Apr. 1996, pp. 176–186.

[46] D. D. Bochtis and S. G. Vougioukas, "Minimising the non-working distance travelled by machines operating in a headland field pattern," *Biosystems Eng.*, vol. 101, no. 1, pp. 1–12, Sept. 2008.

[47] G. Gutin and A. P. Punnen, Eds., *The traveling salesman problem and its variations*, ser. Combinatorial optimization. Dordrecht, London: Kluwer Academic, 2002.

[48] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ, USA: Princeton University Press, 2007.

[49] L. E. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *Amer. J. Math.*, vol. 79, no. 3, pp. 497–516, July 1957.

[50] K. Savla, E. Frazzoli, and F. Bullo, "Traveling salesperson problems for the Dubins vehicle," *IEEE Trans. Autom. Control*, vol. 53, no. 6, pp. 1378–1391, July 2008.

[51] X. Ma and D. A. Castanon, "Receding horizon planning for Dubins traveling salesman problems," in *Proc. 45th IEEE Conf. Decision and Control*, San Diego, CA, Dec. 2006, pp. 5453–5458.

[52] Z. Tang and U. Ozguner, "Motion planning for multitarget surveillance with mobile sensor agents," *IEEE Trans. Robot.*, vol. 21, no. 5, pp. 898–908, Oct. 2005.

[53] A. C. Medeiros and S. Urrutia, "Discrete optimization methods to determine trajectories for Dubins' vehicles," *Electronic Notes in Discrete Mathematics*, vol. 36, pp. 17 – 24, Aug. 2010.

[54] A. Aggarwal, S. Khanna, R. Motwani, and B. Schieber, "The angular-metric traveling salesman problem," *Proc. Annu. ACM-SIAM Symp. Discrete Algorithms*, vol. 29, pp. 221–229, Jan. 1997.

[55] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[56] D. G. Macharet, A. Alves Neto, V. F. da Camara Neto, and M. F. M. Campos, "Nonholonomic path planning optimization for Dubins' vehicles," in *Proc. IEEE Int. Conf. Robotics and Automation*, Shanghai, China, May 2011, pp. 4208–4213.

[57] T. Feo and M. Resende, "Greedy randomized adaptive search procedures," *J. Global Optimization*, vol. 6, no. 2, pp. 109–133, Mar. 1995.

[58] J. Le Ny, E. Frazzoli, and E. Feron, "The curvature-constrained traveling salesman problem for high point densities," in *Proc. 46th IEEE Conf. Decision and Control*, New Orleans, LA, Dec. 2007, pp. 5985–5990.

[59] J. Le Ny, E. Feron, and E. Frazzoli, "On the Dubins traveling salesman problem," *IEEE Trans. Autom. Control*, vol. 57, no. 1, pp. 265 –270, Jan. 2012.

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

[61] J. S. B. Mitchell, "A PTAS for TSP with neighborhoods among fat regions in the plane," in *Proc. ACM-SIAM Symp. Discrete Algorithms*, New Orleans, LA, Jan. 2007, pp. 11–18.

[62] A. Dumitrescu and J. S. B. Mitchell, "Approximation algorithms for TSP with neighborhoods in the plane," *J. Algorithms*, vol. 48, no. 1, pp. 135–159, Aug. 2003.

[63] M. de Berg, J. Gudmundsson, M. J. Katz, C. Levcopoulos, M. H. Overmars, and A. F. van der Stappen, "TSP with neighborhoods of varying size," *J. Algorithms*, vol. 57, pp. 22–36, Sept. 2005.

[64] K. Elbassioni, A. V. Fishkin, N. H. Mustafa, and R. Sitters, "Approximation algorithms for euclidean group TSP," in *Proc. Int. Conf. Automata, Languages and Programming*, vol. 3580, Lisbon, Portugal, July 2005, pp. 1115–1126.

[65] B. Yuan, M. Orlowska, and S. Sadiq, "On the optimal robot routing problem in wireless sensor networks," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1252–1261, Sept. 2007.

[66] L. He, J. Pan, and J. Xu, "Reducing data collection latency in wireless sensor networks with mobile elements," in *Proc. IEEE Conf. Computer Communications Workshops*, Shanghai, China, 2011, pp. 572–577.

[67] A. G. Chentsov and L. N. Korotayeva, "The dynamic programming method in the generalized traveling salesman problem," *Math. Comput. Modelling*, vol. 25, no. 1, pp. 93–105, Jan. 1997.

[68] C. E. Noon and J. C. Bean, "A lagrangian based approach for the asymmetric generalized traveling salesman problem," *Operations Research*, vol. 39, no. 4, pp. 623–632, 1991.

[69] J. Renaud and F. F. Boctor, "An efficient composite heuristic for the symmetric generalized traveling salesman problem," *European J. Operational Research*, vol. 108, no. 3, pp. 571–584, Aug. 1998.

[70] L. V. Snyder and M. S. Daskin, "A random-key genetic algorithm for the generalized traveling salesman problem," *European J. Operational Research*, vol. 174, no. 1, pp. 38–53, Oct. 2006.

[71] C. E. Noon and J. C. Bean, "An efficient transformation of the generalized traveling salesman problem," Dept. Ind. Ops. Eng., Univ. Michigan, Ann Arbor, MI, Tech. Rep. 91-26, 1991.

[72] Y.-N. Lien, E. Ma, and B. W.-S. Wah, "Transformation of the generalized traveling-salesman problem into the standard traveling-salesman problem," *Inform. Sciences*, vol. 74, no. 1, pp. 177–189, Oct. 1993.

[73] V. Dimitrijević and Z. Šarić, "An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs," *Inform. Sciences*, vol. 102, no. 1, pp. 105–110, Nov. 1997.

[74] K. J. Obermeyer, "Path planning for a UAV performing reconnaissance of static ground targets in terrain," in *Proc. AIAA Conf. Guidance, Navigation and Control*, Chicago, IL, Aug. 2009.

[75] K. J. Obermeyer, P. Oberlin, and S. Darbha, "Sampling-based roadmap methods for a visual reconnaissance UAV," in *Proc. AIAA Conf. Guidance, Navigation and Control*, Toronto, Canada, Aug. 2010.

[76] J. T. Isaacs, D. J. Klein, and J. P. Hespanha, "Algorithms for the traveling salesman problem with neighborhoods involving a Dubins vehicle," in *Proc. Amer. Control Conf.*, San Francisco, CA, June 2011, pp. 1704–1709.

[77] M. Houle and G. Toussaint, "Computing the width of a set," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 10, no. 5, pp. 761–765, Sept. 1988.

[78] A. M. Shkel and V. Lumelsky, "Classification of the Dubins set," *Robotics and Autonomous Syst.*, vol. 34, no. 4, pp. 179 – 202, Mar. 2001.

[79] A. M. Frieze, G. Galbiati, and F. Maffioli, "On the worst-case performance of some algorithms for the asymmetric traveling salesman problem," *Networks*, vol. 12, no. 1, pp. 23–39, 1982.

[80] K. Helsgaun, "An effective implementation of the Lin-Kernighan traveling salesman heuristic," *European J. Operational Research*, vol. 126, no. 1, pp. 106–130, Oct. 2000.

[81] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Editors, Ed. Addison-Wesley, 1989.

[82] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. New York, NY, USA: Springer-Verlag New York, Inc., 1994.

[83] Y. Liu and J. Huang, "A novel genetic algorithm and its application in TSP," in *Proc. Int. Conf. Network and Parallel Computing*, Shanghai, China, Oct. 2008, pp. 263–266.

[84] K. Savla, E. Frazzoli, and F. Bullo, "On the point-to-point and traveling salesperson problems for Dubins' vehicle," in *Proc. Amer. Control Conf.*, Portland, OR, June 2005, pp. 786–791.

[85] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New Results and New Trends in Computer Science*, H. Maurer, Ed. Berlin, Germany: Springer, 1991, vol. 555, pp. 359–370.

[86] J. M. Steele, "Probabilistic and worst case analyses of classical problems of combinatorial optimization in euclidean space," *Math. Operations Research*, vol. 15, no. 4, pp. 749–770, Nov. 1990.

[87] M. Ma, Y. Yang, and M. Zhao, "Tour planning for mobile data-gathering mechanisms in wireless sensor networks," *IEEE Trans. Veh. Technol.*, vol. 62, no. 4, pp. 1472–1483, May 2013.

[88] L. He, J. Pan, and J. Xu, "A progressive approach to reducing data collection latency in wireless sensor networks with mobile elements," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1308–1320, Jul. 2013.

[89] D. Kim, R. Uma, B. Abay, W. Wu, W. Wang, and A. Tokuta, "Minimum latency multiple data mule trajectory planning in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 4, pp. 838–851, Apr. 2014.

[90] T. Shima, S. Rasmussen, and D. Gross, "Assigning micro UAVs to task tours in an urban terrain," *IEEE Trans. Control Syst. Technol.*, vol. 15, no. 4, pp. 601–612, Jul. 2007.

[91] E. Stump and N. Michael, "Multi-robot persistent surveillance planning as a vehicle routing problem," in *Proc. IEEE Int. Conf. Automation Sci. Eng.*, Aug. 2011, pp. 569–575.

[92] C. Murray and W. Park, "Incorporating human factor considerations in unmanned aerial vehicle routing," *IEEE Trans. Syst., Man, Cybern.,Syst*, vol. 43, no. 4, pp. 860–874, Jul. 2013.

[93] P. Wang, K. Gupta, and R. Krishnamurti, "Some complexity results for metric view planning problem with traveling cost and visibility range," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 3, pp. 654–659, Jul. 2011.

[94] S. Rathinam, R. Sengupta, and S. Darbha, "A resource allocation algorithm for multivehicle systems with nonholonomic constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 1, pp. 98–104, Jan. 2007.

[95] N. Chakraborty, S. Akella, and J. Wen, "Coverage of a planar point set with multiple robots subject to geometric constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 1, pp. 111–122, Dec. 2010.

[96] H. C. W. Lau, T. Chan, W. T. Tsui, and W. K. Pang, "Application of genetic algorithms to solve the multidepot vehicle routing problem," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 383–392, Apr. 2010.

[97] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic, "Time windows based dynamic routing in multi-AGV systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 1, pp. 151–155, Jan. 2010.

[98] K. Sundar and S. Rathinam, "Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 287–294, Jan. 2014.