Robot Navigation for RFID-based Inventory Counting

by

Xue Xia

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 7, 2016

Keywords: robot navigation, RFID tags, Traveling Salesman Problem, optimization algorithm, linear-solver

Approved by

Thaddeus Roppel, Chair, Associate Professor of Electrical and Computer Engineering
Shiwen Mao, Ginn Professor of Electrical and Computer Engineering
John Hung, Professor of Electrical and Computer Engineering

Abstract

A mathematical model for a robot navigation system searching for RFID tags in a retail store is constructed. It is based on the Traveling Salesman Problem optimization algorithm. A software simulation is performed in MATLAB. The solutions of the Traveling Salesman Problem and the Traveling Salesman Problem with Neighborhoods are simulated. A linear-solver provides the function to solve simultaneous linear equations. Based on the optimization algorithm using the linear-solver, the optimal route for the robot is found.

Acknowledgments

I would like to thank my advisor Dr. Roppel for the guidance during my thesis writing process. I would like to thank Dr. Mao and Dr. Hung as my masters thesis defense committee members. I would like to express my gratitude to my lab members Jian Zhang and Yibo Lyu, for their suggestions in my study. I would like to show my thankfulness to Huanyi Zhou, Dehua Li, and other friends online as test participants in research. I would like to thank Leo Paz, Bailey, Matthew S, and other staff at The Miller Writing Center and The International Student English Center for their help to correct my English grammar errors. I am grateful to my parents and friends for their encouragement.

Table of Contents

List of Tables

List of Illustrations

List of Abbreviations

UHF	ultra-high frequency
RFID	radio-frequency identification
SA	simulated annealing algorithm
GA	genetic algorithm
ACO	ant colony optimization algorithm
PSO	particle swarm optimization algorithm
TSP	Traveling Salesman Problem

1. INTRODUCTION

Robots can adequately replace human workers in some applications. It has been proposed to replace or assist humans taking inventory in retail stores with robots [1]. To achieve this idea, the first step is to accomplish robot navigation in a retail store environment, which consists of various types of shelves, floor racks, and similar merchandising structures. In order to navigate the environment, the robot needs to know its present location, the goal it will move to next, and the acceptable path(s) from its current location to the goal [2].

In recent years, robot technology has made substantial progress. Navigation technologies are developing at a rapid pace. They are divided into four classes. Those four categories are dead-reckoning-based, landmark-based, vision-based, and behavior-based techniques [3].

The most basic technology is dead-reckoning-based navigation. Instructions such as moving straight, turning left, and turning right, are set in the robot operating system. The robots follow instructions to control wheels to move to the location of the goal with a time counter [4]. The idea of dead-reckoning navigation is simple. It also has a low source cost on robot hardware. However, during testing in real world, noise from the environment and measurement errors on mechanical structure of robots are hardly to be ignored [5].

To decrease noise impact in robot navigation, the hardware provides various functions as support. Cameras represent eyes to robots. They are used in landmark navigation. Using image detection algorithms, robots can distinguish certain figures through the camera. With camera catching symbols, robots can get instruction after analyzing figures by the algorithms. Then robots can move in the exact direction following the symbols [6]. However, when a robot moves into the range where its camera detects a symbol, the position of the robot in relation to the symbol may cause the figure to appear twisted and unrecognizable or rotated beyond an acceptable degree. To improve the performance of vision systems, adaptive learning algorithms such as artificial neural networks can be used to improve performance. With a sufficiently large amount of image data training, the robot vision system can be taught to distinguish symbols more reliably.

Cameras are also needed for vision-based navigation. Visual features in figures are used for algorithms to judge the current location of a robot during navigation. Sometimes a depth image generated by stereo cameras or LIDAR is used to create 2D maps [7]. Factors affecting performance include lighting variations, color, texture, and frame rate. Vision-based navigation can consume a lot of hardware resources and power.

For behavior-based navigation, various sensors are needed for robots. Robots rely on sensors to react to an unknown environment and move in the right direction [8]. Sonars send UHF (ultra-

high frequency) sound waves and receive echoes, as they continue to calculate the distance from objects. IR sensors sense potential objects using infrared reflection. Keeping a constant distance from the wall, robots are capable of going straight as they avoid colliding with obstacles in an indoor environment. Lasers provide the robot with a wide degree of range to sense obstacles , which allows the robot to create a 2D map when arriving at a new environment.

The hardware cost of behavior-based navigation is acceptable when the environment is simple, such as indoor environments with no moving obstacles. When a robot navigates outdoors, environments can be complex, and noise is unavoidable. Robots also need to avoid moving objects, such as people or cars. Hardware quality decides the response time for a robot to act.

Sometimes, RFID tags and readers are used for robot navigation. RFID tags contain needed information read by robots for navigation. RFID readers are installed as I/O devices in the robot operating system. RFID readers can sense tags within a certain range, then robots can get data and write it to database [9]. When a robot senses one tag, it accomplishes the process of reading data and writing to database at one location, then it moves to the next place where a tag exists.

The work described in this thesis addresses the use of mobile robots to take inventory in a retail environment. It is presumed that all items in the store have had passive RFID tags applied. The robot, or team of robots, must navigate the store and read all of the tags. The navigation strategy might be any combination of the aforementioned – dead reckoning, beacon, landmark, and RFID marker. The use of robots in this application can presumably reduce labor cost and improve inventory accuracy.

In order to avoid missing tags and failing to get data from labels successfully, the robot navigation algorithm is designed to move close to the location of the stock. More precisely, the robot should move parallel to shelving, and circle around tables and free-standing racks.
The floor in retail environments and warehouses is usually fairly uncluttered. Robots can move to any place except locations where shelves and boxes are stacked. There are various available routes to finish visiting all RFID tags. However, different routes of robot navigation vary in distance. In order to accomplish tasks with efficiency, the shortest path is necessary for robots. Robots can decide the tag visiting order in robot navigation with the algorithms running on the robot operating system. The algorithms are required to work out the optimum route in order to improve the distance that each time visiting costs.

Various algorithms are developed for different conditions, considering a study with optimized results. There are already several algorithms for certain conditions related to the shortest distance path searching.

In graph theory, vertices and the edges that connect the vertices form a graph. In our robot navigation scenario, the vertices represent locations near expected high populations of RFID tags, such as walls and merchandise display equipment. In our scenario, the robot needs to visit all vertices in a graph (store map) and then return to the beginning point in the most efficient manner possible.

To test RFID hardware performance in a lab, the basic algorithm is designed assuming that a robot begins its navigation at a predetermined starting point, and the next tag which the robot

will visit is defined as the nearest tag that has not been reached. This simple algorithm can ensure that the robot finishes visiting all tags without missing any. Analyzing the path left behind by a robot, its end point and start point are at the same place, which makes it a circular route. The problem seems similar to the Traveling Salesman Problem.

In robot navigation, the basic algorithm selects the nearest tag location as the local optimal result. However, the total length sum of all local optimal results is not equal to the global optimum value of the Traveling Salesman Problem. The distance of the path visited depends on the location of the tags. In the worst case, it can be long. Therefore, a more efficient searching algorithm is needed.

2. LITERATURE REVIEW

An RFID systems which includes RFID tags and RFID readers is used in robot navigation. RFID tags store information that the robot needs. RFID readers transmit and receive RF signals. The robot uses RFID readers to read data from RFID tags [10].

In robot navigation, when axis length, wheel radius, wheel angular velocity, head posing direction and previous position of robot are provided, the operating system makes an estimate of the robot current position. When the robot speed is fast in relation to the sampling rate, excessive odometric error can occur, leading to instability.

In robot navigation, besides the support from hardware, algorithms are also needed. As a well known optimization problem, there are various algorithms developed for the Traveling Salesman Problem. To design algorithm structure, programmers get inspiration from creature behavior and physical particle movement with probability searching.

One of the algorithms that is used for solving the Traveling Salesman Problem is the Simulated Annealing algorithm. It is a kind of greedy algorithm. However, on the basis of greedy algorithm, Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983 [11], and Vlado Černý in 1985

gained ideas from solid physical annealing process and designed the Simulated Annealing Algorithm respectively [12].

Metal is melted and annealed to improve its quality. Before metal annealing , atoms stay at the place that has local minimal internal energy. When the temperature is high, atoms inside metal gain enough energy to have a chance to leave their original position. Atoms move in the whole space of metal randomly. During the progress, it is possible for atoms to move to a new place that is at lower internal energy than before. When metal temperature drops with a certain speed, atoms are locked at their own place and can hardly move to another place. Through the whole process, the entirety of the atoms move to the place that is at lower internal energy in metal space than before. The new atom distribution of metal leads to metal quality improvement.

To apply to the Traveling Salesman Problem, SA (Simulated Annealing) sets a high temperature as the beginning value. Nodes on TSP graph represents atoms in metal. To begin the Simulated Annealing algorithm, nodes connect with each other randomly. The initiatal route is usable but not optimal. Then nodes at a high temperature have a high probability to break original connections and link to new nodes. After atom motion, the distance of the new route will be calculated. Nodes move randomly, so that motion results can be both better or worse to the whole route. When the temperature is high, worse results have a chance to be accepted. Simulated Annealing Algorithm creates the probability to break the local minimum and jump to the global optimum answer. When the temperature goes down gradually, the whole route becomes stable, and new changes have a tiny chance to occur. During the loop of temperature

dropping, SA through probability search converges to optimal result of the Traveling Salesman Problem.

At the same time, the Genetic algorithm (GA) is often mentioned with the Simulated Annealing algorithm. GA is also used to solve the Traveling Salesman Problem. GA, originally published by J.H.Holland [13], gets its inspiration from the Darwinian theory of evolution.

On the TSP map, the route which includes all nodes is seen as a chromosome by the Genetic Algorithm. The visiting order of nodes is turned into the DNA information which each chromosome contains. Routes that visit all nodes are needed. They might be far from the optimal route, but they are used as the initial chromosomes to begin evolution. The GA pairs route vectors and exchanges random parts of node order on the vectors [14]. This progress comes from the behavior that chromosome pairs exchange their DNA. In order to avoid dead loops to miss the optimal result, the GA adds low probability chance to generate a new random node order on route vector. This step is the same as mutation that rarely occurs in nature.

After each loop of evolution, the distance of the routes will be calculated. The fitness of each vector will be judged by the distance. When the next evolution loop begins, the GA chooses vector pairs through fitness. The selection process is a probability search. When vectors have higher fitness, they are more likely to be chosen.

With only one or two generation loops, the improvement of the Traveling Salesman Problem route vectors is not clear enough by the GA. However, when the loop time value is set to

hundreds or thousands, the optimal route of the TSP is filtered out by the GA. The whole process of the GA is to achieve biological evolution through natural selection. The GA selects the fittest route as the optimization result, and eliminates others.

Another TSP algorithm that follows biologic behavior is the Ant Colony Optimization algorithm (ACO). In 1992, Marco Dorigo published the ACO according to the behavior of ants [15]. The basic idea of the ACO is that a group of ants find the shortest path to get food. In ACO, all ants are considered as an entirety. The ant group is defined as swarm intelligence. Each ant in the swarm behaves simply. However, when the whole ant group works cooperatively, the entirety can accomplish complex tasks.

To solve the Traveling Salesman Problem, ants need to visit all nodes on the TSP map. A road represents a line that connects a node pair. When an ant traverses a road, it leaves its pheromone (chemical signal) on the road. The pheromone evaporates over time. When a path that contains all nodes has a short distance, an ant can finish moving on the path and return to the colony with less time. There is more pheromone left on short roads than long roads during the same time length. Ants choose a path according to the pheromone amount left on road. With biological instinct, ants prefer to choose roads with more pheromone left. The more ants choose the same road, the more pheromone is left on road. The choice of path and the pheromone left on the path builds a loop.

After each searching loop, the pheromone left on each road will be updated. After a certain number of loops, all ants will choose the same path. The stable route result is the optimal path of the Traveling Salesman Problem.

Parameters in the ACO control the speed of pheromone volatilizing and the probability that an ant will choose one road. The performance of the ACO depends on the setup values of parameters. In order to improve the ACO, the suitable values of the parameters are needed. To find the usable values for the parameters, neural networks are used to train the results [16].

Considering that the Traveling Salesman Problem is a worthy optimization topic, there are more algorithms developed. Kennedy, Eberhart and Shi developed another algorithm to solve the TSP problem as the Particle Swarm Optimization algorithm (PSO) [17]. The underlying idea of PSO is also a kind of swarm intelligence. modeled on a school of fish or flock of birds.

In PSO, every particle is the same as a bird in a bird flock. Imitating a bird flying in the sky, each particle has its own position and velocity. When birds fly to search for food, no birds knows the exact location of food at the beginning. When one bird finds food in its visual field, it flies to the food. As a swarm in the algorithm, each particle shares information with each other. Seeing one bird find food, other birds will leave their previous location and change their velocity direction to fly to the area where the first bird found food. After certain times of swarm communication, all birds fly near to the food location. In the Particle Swarm Optimization Algorithm, all particle locations converge to the optimal answer.

However, when applying to the Traveling Salesman Problem, the Particle Swarm Optimization Algorithm is easy to lock to the local optimal answer rather than the global best result. The Simulated Annealing algorithm is used to improve the performance of the Particle Swarm Optimization algorithm [18]. Another method is to set the Particle Swarm Optimization Algorithm as an overall framework, and take crossover and mutation steps from the Genetic Algorithm. Two algorithms are mixed as a new hybrid algorithm to solve the Traveling Salesman Problem [19].

3. ALGORITHM BASIS

The simulated environment used in this thesis is a model of our real laboratory, in which a robot is equipped with an RFID reader to detect RFID tags placed on retail store merchandise. Since RFID readers sense tags within a non-zero range, the robot can read all tags by coming within sufficiently close proximity to merchandise locations.. The robot moves from one merchandise location to another, until ideally all tags are read. With the constraint of minimizing the total path length, the robot navigation process is the same as the definition of the Traveling Salesman Problem (TSP).

3.1 Traveling Salesman Problem

According to the name of Traveling Salesman Problem, the basic idea of the problem is about a salesman who travels through all cities on his plan. The distance between cities are different. The salesman needs to visit all cities without repeating his path. After finishing visiting all cities, he returns to the city where he began his travel. In order to make the salesman travel efficiently, the route needs to have the shortest distance while including all cities at the same time.

The Traveling Salesman Problem is a problem about routes and distance, which can be switched into a graph problem model. Vertices of the graph represent cities that the salesman needs to visit. Edges connect vertices. They are the possible paths that the salesman can choose to leave from one city and go to the next. Each edge of the graph has a weight, and the weights stand for the distance between each pair of cities.

The TSP model used for robot navigation in the RFID lab is an undirected weight graph. An undirected weight graph means that if there is an edge existing between vertex A and B, it can be used from both direction A to B and direction B to A. At two opposite directions, the weight of the edge does not change. Robots read RFID tags in the lab, when the robot leaves from one tag to another and returns from another to the first one, the distance of both trails are equal.

In a graph theory definition, the undirected Traveling Salesman Problem can be described as a graph G = (V, E). V stands for vertices of the TSP graph, with V = {1, 2, 3,..., n}. The number n in the V set is the total vertex number. E contains all edges that connect vertices of the TSP graph. An edge connects two vertices. So (i, j) with (i, j = 1, 2, 3,..., n) are used to describe edge position. Distance between vertices is defined as d_{ij}. The weight of each edge is equal to d_{ij}. Searching the optimization of the Traveling Salesman Problem is equal to working out the minimum weight sum of edges that connect all vertices as a ring [20].

To start analysis of the robot navigation map, each tag on the walls are turned into a vertex. Assume that the object volume is small so that objects with RFID tags around are simplified as vertices with no volume. The location of these vertices are at the center of the objects.

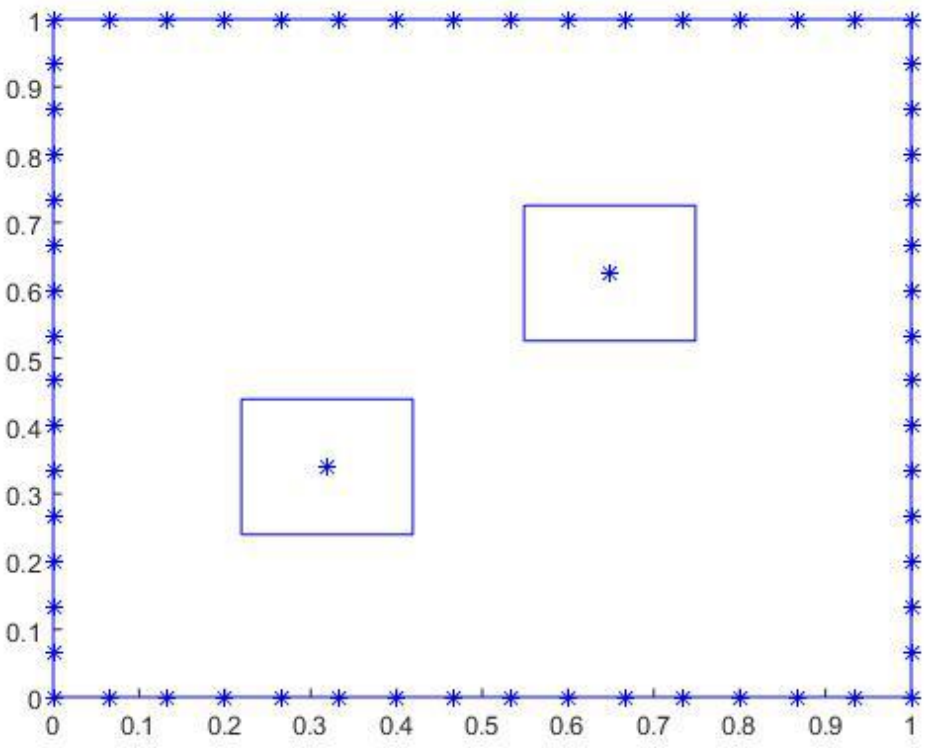

Figure 3.1 Simple Objects.RFID Tags Represented as TSP Vertices

3.2 Traveling Salesman Problem with Neighborhoods

The optimized path is discussed about the Traveling Salesman Problem. However, in practical application, separated vertices on a map may be divided into different zones, which leads to a derived problem called the Traveling Salesman Problem with Neighborhoods. Each zone has a center node connected to the main map, so that vertices in the zone map are like neighborhoods to the center node. When arriving at a center node on the main map, the zone map will be shown. In order to visit a vertex in a particular zone, the salesman must visit the center node first.

The Traveling Salesman Problem with Neighborhoods (TSPN) is an extension of the Traveling Salesman Problem. To define the Traveling Salesman Problem with Neighborhoods in graph theory language, parameters that graph G=(V, E), vertex set V, edge set E and weight set d_{ij} are the same as the definition of the Traveling Salesman Problem. Set Q is used to describe neighborhoods of each vertex. Neighborhoods of the TSPN are numbered with (i, k). Number i = 1, 2,..., n represent the vertex number on the Traveling Salesman Problem main map. Number k = 1, 2,..., m means the vertex number on the neighbor graph [21].

The tactic of robot navigation turns whole objects into center nodes the same as RFID tags on the wall. However, as mentioned before, RFID tags are also on objects as well. Separated small maps are built for each object in the room. Every map contains locations of the RFID tags for one, and those tags also generate a TSP circle. A sub map connects to the main map with its object center. Tags on different objects do not interfere with each other.

When a robot begins its navigation from the starting point near the wall, it follows the algorithm instruction to visit each tag. When the algorithm works out the time to leave the wall, the robot turns its direction and moves to the center of the chosen object. When the distance between the robot and the center of the object is smaller than a certain value, the main map is switched to the object map in the robot operating system. The robot begins the TSP route on the sub map. The same algorithm that keeps running on the main map continues calculating the optimal route of the TSP on the sub map. After finishing moving around the object, the robot operating system switches the object map to the main map again. The robot continues moving on the main map as the algorithm suggested.

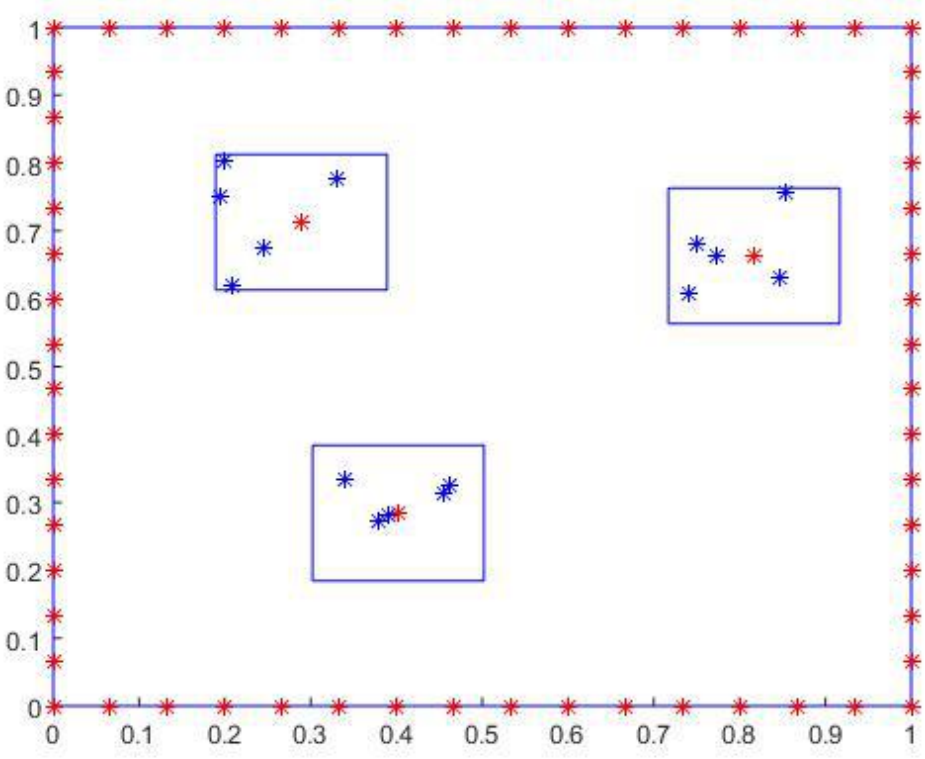

Figure 3.2 Random vertices generated within object zone

3.3 Matlab Simulation

Before the hardware test on the robot, algorithm performance is needed to simulate on software. Matlab provides a suitable software platform to run simulation.

Matlab version 2014 and above provides tools and functions to solve Traveling Salesman Problem [23]. Sharing the same disadvantage of all other algorithms for Traveling Salesman Problem, the complexity of all algorithms remains high. While input data amount grows large, time cost increases quickly.

First, Matlab functions are used to create and plot a test map of a room. Assume that the shape of the room is a rectangle. In most common conditions, room shape is approximately similar to a rectangle. The edge of the map is the location of four walls and obstacles are in the middle of the room. Several objects are placed in the room randomly. Objects represents shelves, boxes or furniture in a retail environment. To simply map building and simulation, object shape is set to rectangle. In the middle of each object rectangle, star dots are the center location of objects.

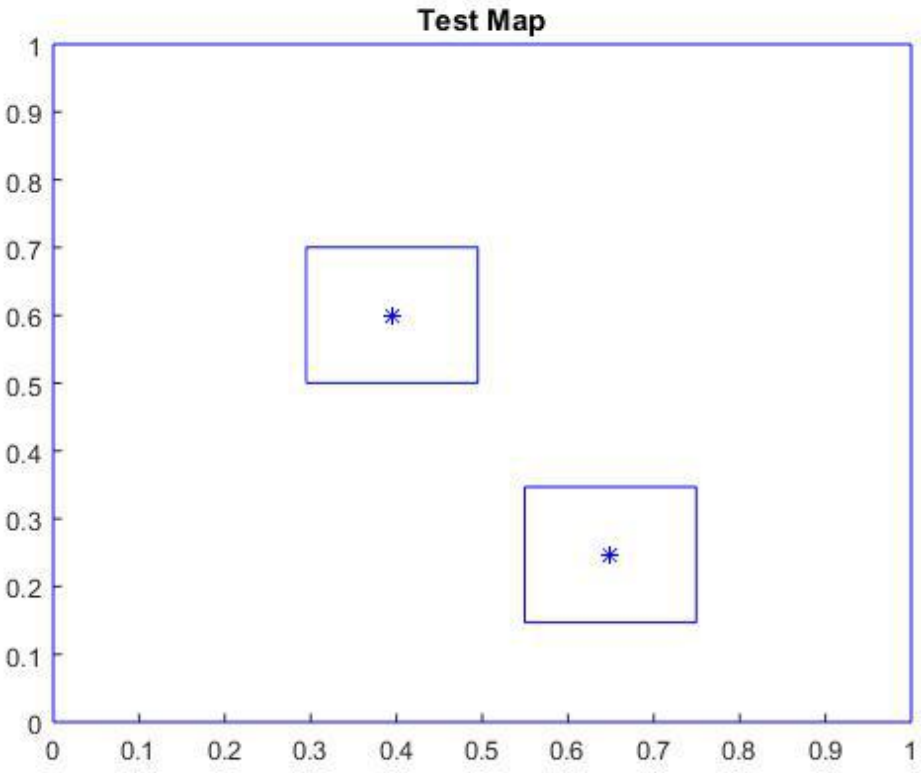

Figure 3.3 A test map with two obstacles in the room

In the initial step, the number of vertices of each map is calculated. Each vertex is marked with a corresponding number for later steps. Vertices are set on the map with its x, y-coordinate values. A new matrix is created to store x, y-coordinates of all vertices.

To work out optimization of the Traveling Salesman Problem, weights of all edges are needed. With collection of all vertex coordinate data on the map, the distance between each pair of vertices can be calculated.

Each two vertices on the map are chosen as a pair. All pairs are stored in a new matrix without repeating. N is the number of total vertices on the Traveling Salesman Problem map. The number of all vertex pairs is M = C(N, 2) = N!/2!(N - 2)! = N(N - 1)/2. In order to store all vertex pairs, the values of N and M are the row and column counts of the matrix.

With sorted vertex pair order matrix, the distance of each pair of vertices can be calculated with the following formula. All vertices of the Traveling Salesman Problem graph are generated in a Cartesian coordinate system. To calculate the distance between two vertices P1 and P2, with x, y-coordinates P1(x1, y1), P2(x2, y2), the following formula is used.

$$D = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

3.3.1 Solution with equality constraints

Matlab 2014 (or higher version) provides the function *intlinprog* as a linear solver. Depending on the linear solver function, optimization of Traveling Salesman Problem is turned into an optimal answer of a set of linear equations and inequalities.

The definition of

$$[\mathrm{x}, \mathrm{fval}, \mathrm{exitflag}, \mathrm{output}] = \mathrm{intlinprog}(\mathrm{f}, \mathrm{intcon}, \mathrm{A}, \mathrm{b}, \mathrm{Aeq}, \mathrm{beq}, \mathrm{lb}, \mathrm{ub}, \mathrm{options})$$

is equal to

$$\min_x f^T x \text{ subject to } \begin{cases} x\,(\text{intcon}) \text{ are integers} \\ A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub \end{cases}$$

Variable fval = f'*x, which is the va ue of optimization. Vector f stands for coefficient vector of

l

optimal result. Vector x is the optimization value that the linear solver needs to work out. Variable exitfalg is used to report function execution condition after function ends. Optimization process is stored in structure output. Elements in vector x with the constraint intcon must be integer. Described by Matlab matrix language, matrix A is a linear inequality constraint matrix, and vector b is a linear inequality constraint vector. Matrix Aeq is a linear equality constraint matrix, and vector beq is a linear equality constraint vector. Vector lb and up stand for lower and upper bound respectively. If the optimization requirements do not contain the lower and upper bounds constraints to the results, vector lb and up are set to empty [22].

In order to work out optimization of the Traveling Salesman Problem with a linear solver function, it is necessary to establish constraints.

Observing optimization of the Traveling Salesman Problem, the shape of optimal x is a ring which includes all vertices. To complete a ring, edges must connect all vertices end to end. A ring has no beginning and end node. In a ring, every vertex includes the endpoint of two edges. According to the requirement of the Traveling Salesman Problem, no repeated trails exists in a ring. Therefore each node can only be endpoints of two edges, no more and no less.

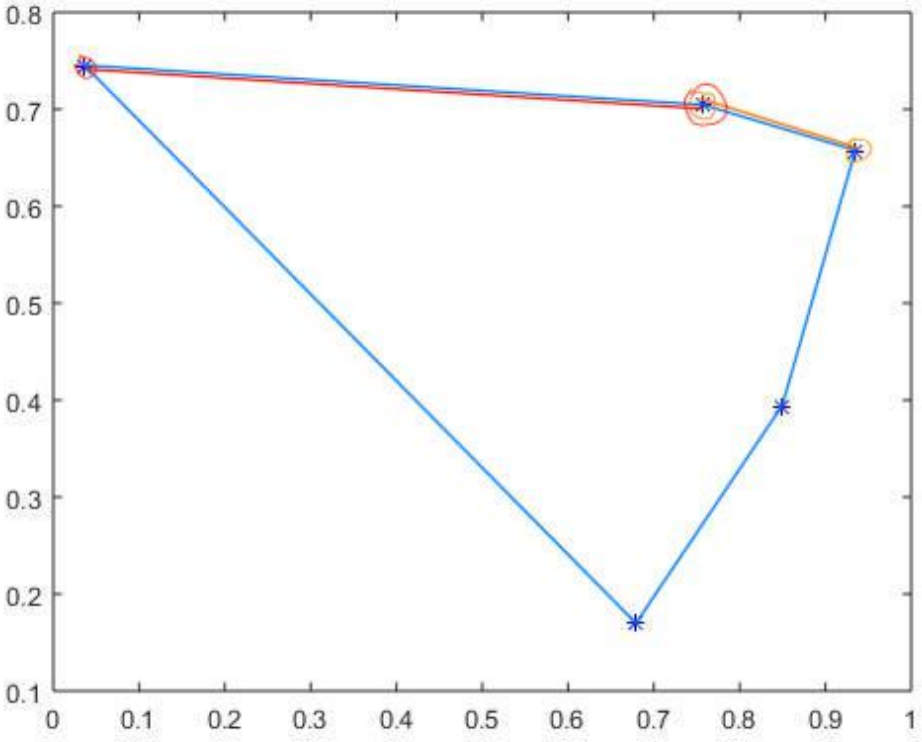

Figure 3.4 A vertex as endpoints of two edges

Create a new matrix Aeq. Set rows of Aeq equal to the number of vertices, and column of Aeq equal to vertex pair number. The algorithm traverses all vertices. When the algorithm is visiting vertex i, if node pair j contains vertex i, element (i, j) in matrix Aeq value is set to 1. Otherwise, the elements value is set to 0.

$$Aeq_{n\times m} = \begin{cases} 1, \text{if vertex i is an endpoint of either side of an edge} \\ 0, \text{otherwise} \end{cases}$$

The value n is the total number of vertices. Set i = 1, 2, 3,..., n. Value m is the total number of vertex pair, which is equal to n(n - 1)/2. Set j = 1, 2, 3,..., m.

From the first node to the last node in order, every node is able to connect with all other nodes. The algorithm is responsible to select out suitable node pair connections in order to construct optimization of TSP. In optimization of TSP, every vertex appears twice as the endpoint of two edges.

Create new matrix Beq. Set matrix Beq with the length equal to the number of nodes. Set all element values of vector Beq equal to 2. This is equal to the number of times that the vertex appears on the chosen edges that construct the optimal ring.

$$Aeq \cdot x = beq$$

Vector x represents the optimization of TSP, with vector length equal to the number vertex pairs. If a vertex pair is chosen, the element in x with the same vertex number is set to 1. Otherwise, elements are set to 0. X includes all node pairs needed to construct the optimized path, regardless of the order of nodes.

There are only two kinds of value - 1 or 0 - stored in vector x. It is clear that all elements in vector x are integers. Vector intcon is set from 1 to m(the number of vertex pair), which includes all vector x elements. Vector up is set to 1, which means the upper bound of optimization x is 1. Vector lb is set to 0, which means the lower bound of optimization x is 0.

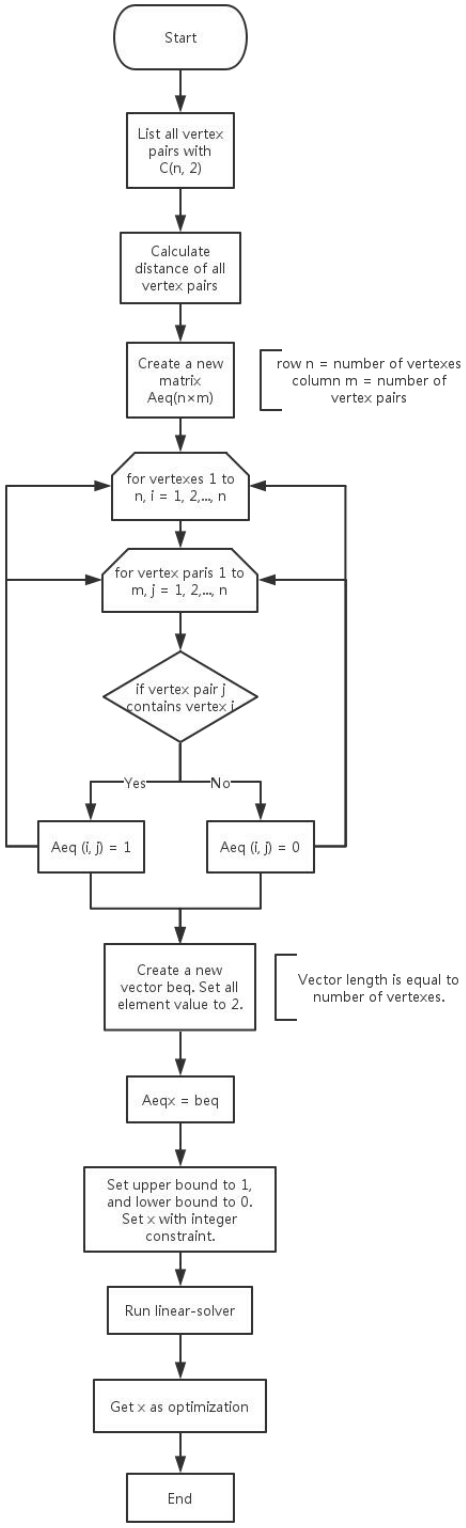

Figure 3.5 Flow chart of the optimization with equality constraints

3.3.2 Eliminate Sub Tours

However, vector x through a linear solver may not be a ring which includes all nodes. Otherwise, it can contain several rings on a graph, which also meets requirements of the equality constraint $Aeq \cdot x = beq$.

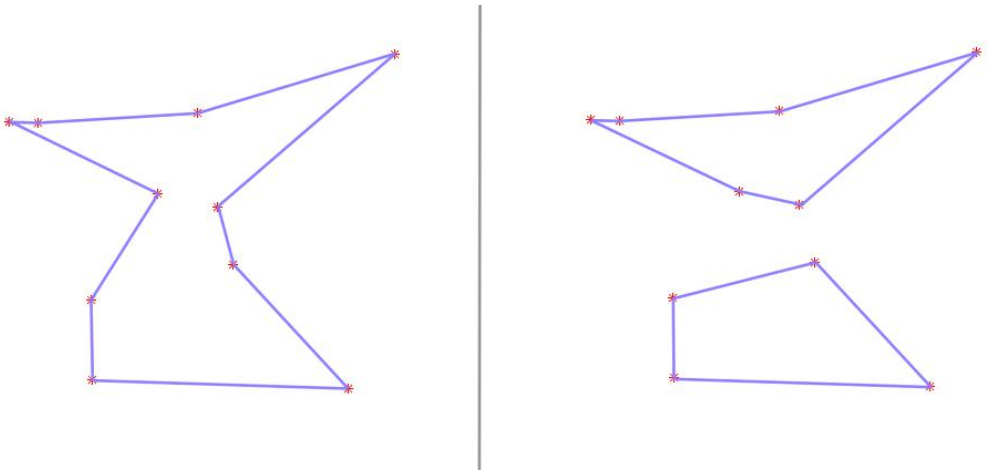

Figure 3.6 Sub tours exist on the TSP graph

Sub rings must be broken and roconstructed to a new main ring. To take further steps, it is necessary to know vertices and edges that build sub tours. The number of sub tours is also need to be known. To finish this step, Matlab provides detectSubtours function.

To reduce vector length, all vertex pairs in current optimization vector x are selected out and stored in a new matrix.

In order to avoid repeated detecting and dead loop, two more lists are created. One list is used to store the vertex pair check status as an unchecked list. If a vertex pair has been checked, the value on the unchecked list is set to 0. Otherwise, it is set to 1. Another list is used to store

checked data. The algorithm only needs to detect vertex pairs whose value remains at 1 on the unchecked list.

Before starting the sub tour search, the value of all elements on the unchecked data list is 1. When a vertex pair has been checked, the value will be set to 0 on the unchecked data list. If there is only one sub tour, the optimization of Traveling Salesman Problem may be used. Otherwise, the number of sub tours is larger than one. It is clear that there is one sub tour at least. At the beginning, the sub tour counter value is set to 1.

Begin detecting at the first vertex pair of all optimal vertex pairs. The index vertex is the first element of the starting pair. The next vertex which needs to be visited is the second element of the starting pair. While the detecting algorithm is running, the visited vertices are stored to the checked list, and the value of the checked vertex pairs is set to 0 on the unchecked list.

For every vertex i, there are two vertex pairs in the optimization that includes the vertex i (i = 1, 2, 3,..., n). One pair is the current pair that the algorithm is visiting. Another pair is the pair that the algorithm will visit. Following the position of vertex i, the algorithm will move from the current vertex pair to the pair that has not been arrived at yet. The new visited vertex pair will be set as a new start pair. In the vertex pair, the new next vertex is the element which is not equal to vertex i. During the detecting loop, the data of the starting vertex pairs and the next vertices all keep being updated. Visited vertices are added to the checked list. At the same time, the unchecked list deletes the visited vertex pairs.

When the value of the next vertex is the same as the index vertex value, vertices of a sub tour have been visited completely. One will be added to the value of the sub tour counter. The sequence number of vertices that construct a sub tour will be stored in a cell in order.

Set the start point at the first non empty element from the unchecked data list. Begin a new loop of tour detecting, until there are no more elements on the unchecked list.

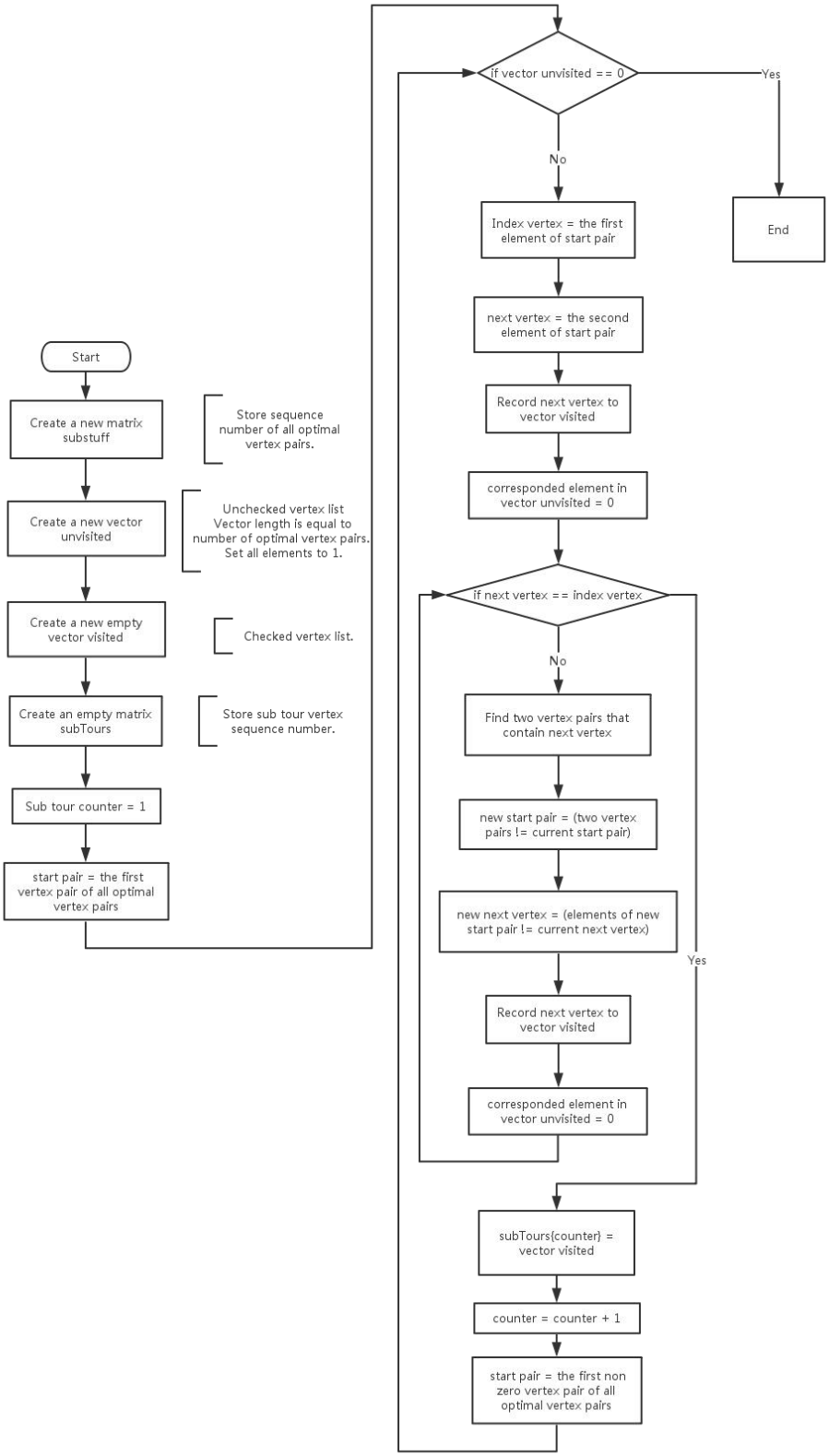

Figure 3.7 Flow chart of the detectSubtours function

The number of sub tours is determined with the help of the function detectSubtours. The function detectSubtours returns a matrix which stores the vertices that construct sub tours in order. The vertex number in each sub tour can be countered. In order to break the sub tours and construct a main ring which includes all vertices of the TSP, more constraints are needed.

A sub tour is a ring. As a ring, the number of vertices and edges is equal. When a ring is broken, the number of its edges must be smaller than the number of its vertices.

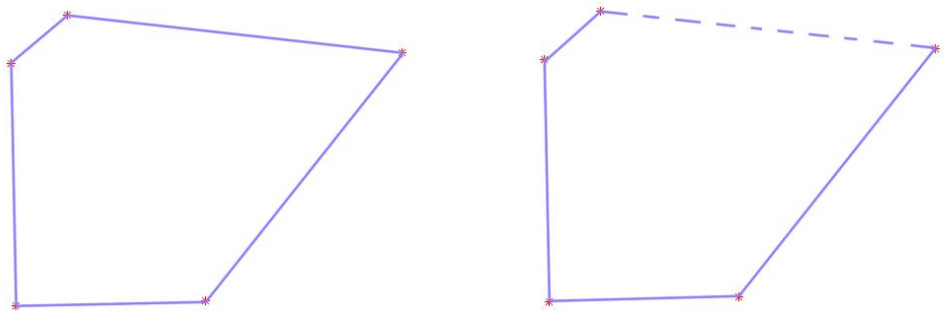

Figure 3.8 A closed circular route and an open route

A new matrix A is created, with the row value equal to the number of sub tours and the column value equal to the number of all vertex pairs (n (n - 1)/2). The vertex pairs of each sub tour are selected out and stored in matrix A.

A new vector B is created, with length equal to the sub tour number. In order to break the circle, elements of vector B are set with the value equal to (vertex number - 1) corresponding to the sequence number of the sub tours.

$A \cdot x < b$

With the constraint $A \cdot x < b$, and old constraints $Aeq \cdot x = beq$, $lb = 0$ and $\text{up} = 1$, the linear-solver e$_{\text{x}}$ecutes agai$_{\text{n}}$.

The optimization x needs to be examined. It is possible that the sub tour number are reduced but still not equal to 1. If the function detectSubTours detects that more than one ring exists on the TSP map, the whole process will be executed again. To avoid dead loop between various sub tours, the space of matrix A and vector B will be extended after each loop. Old constraints are saved. The empty space from the expansion is used to store new constraints. Until there is only one main tour that includes all vertices on the map, the sub tour detecting loops stop and the algorithm exits.

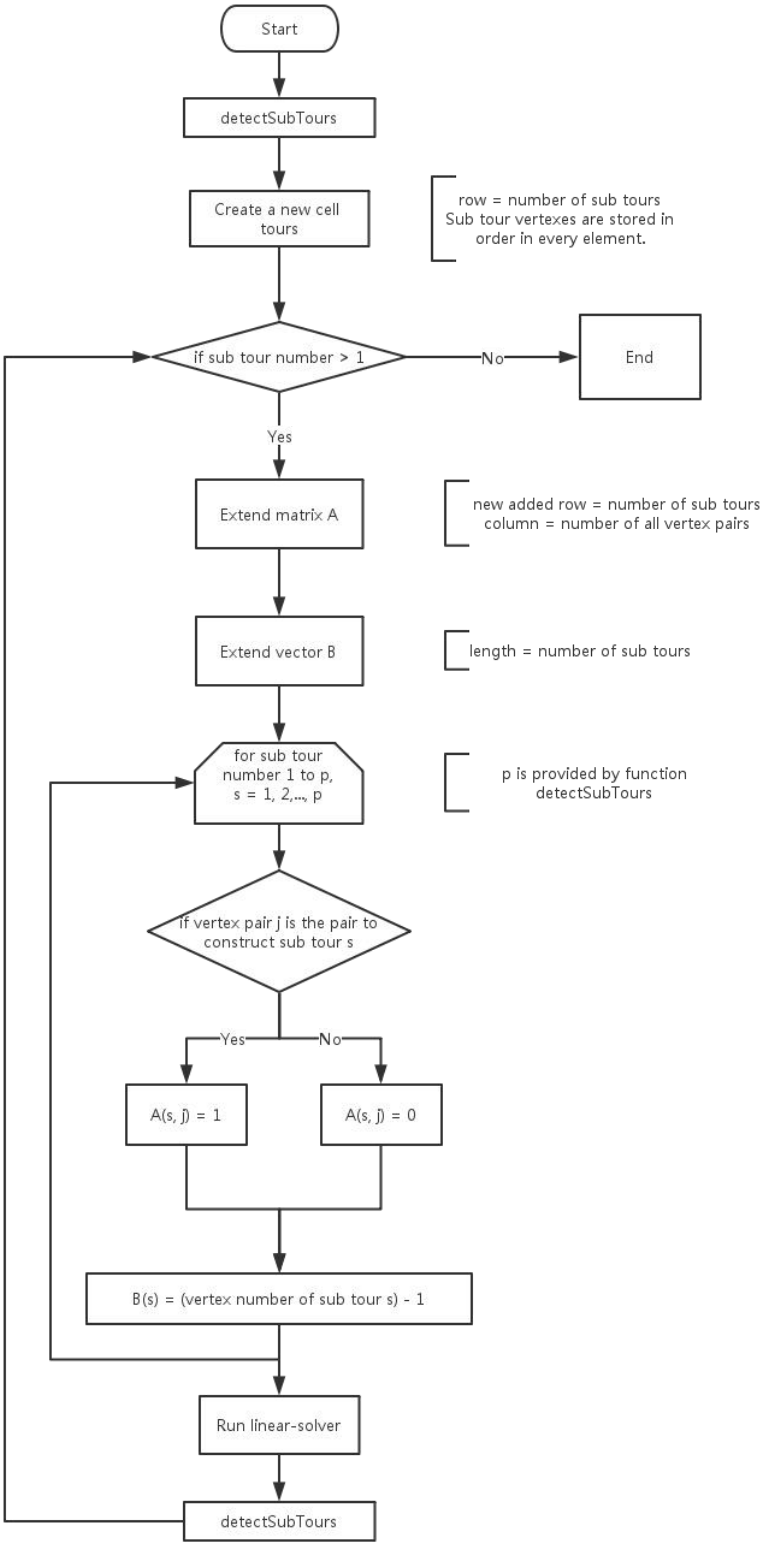

Figure 3.9 Flow chart of the optimization with inequality constraints

4. PERFORMANCE ANALYSIS

4.1 Performance of the Traveling Salesman Problem

4.1.1 Performance with various objects

Simply set the objects on the map as nodes. Star nodes represent the location of the object center. The number of objects is determined by the input parameter.

When there is only one object on the map, the vertex distribution is shown as below in the simulation. The x, y-coordinates value of the object is generated randomly by Matlab.

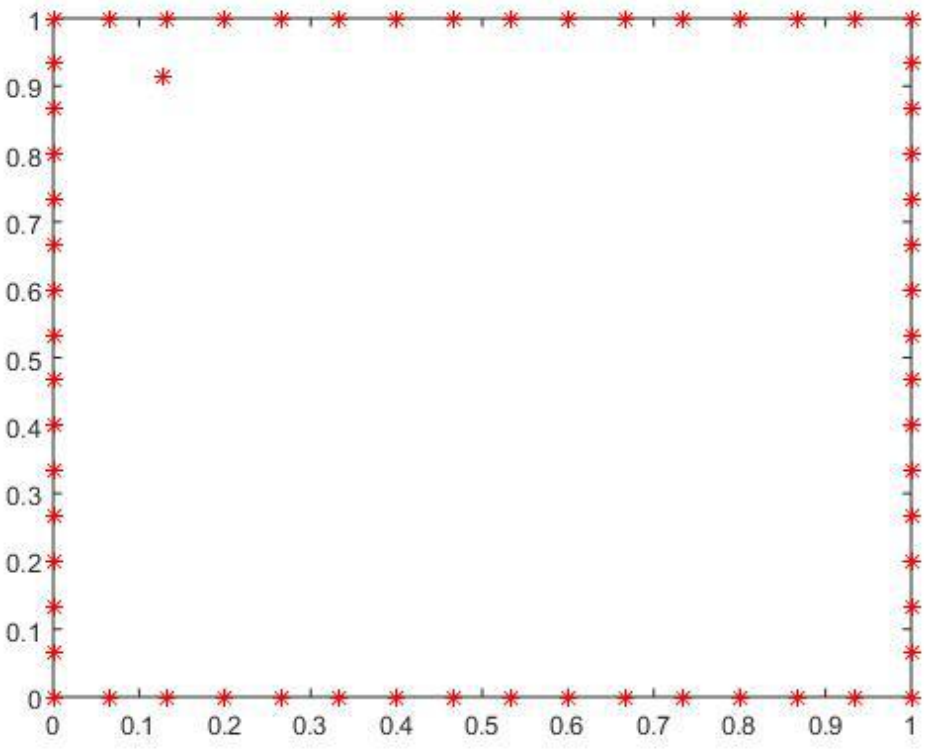

Figure 4.1 The vertex distribution map with one object

RFID tags on the wall occupy most of the percentage of the vertices of the TSP simulation graph. When there is only one object on the map, it is easy to decide the shortest path that connects all vertices on the map. To construct the TSP optimization, the algorithm chooses two edges with the shortest distance from the wall tags to the object location. Vertices on the wall connect other vertices near both of their sides, as most parts of the four wall circumference.

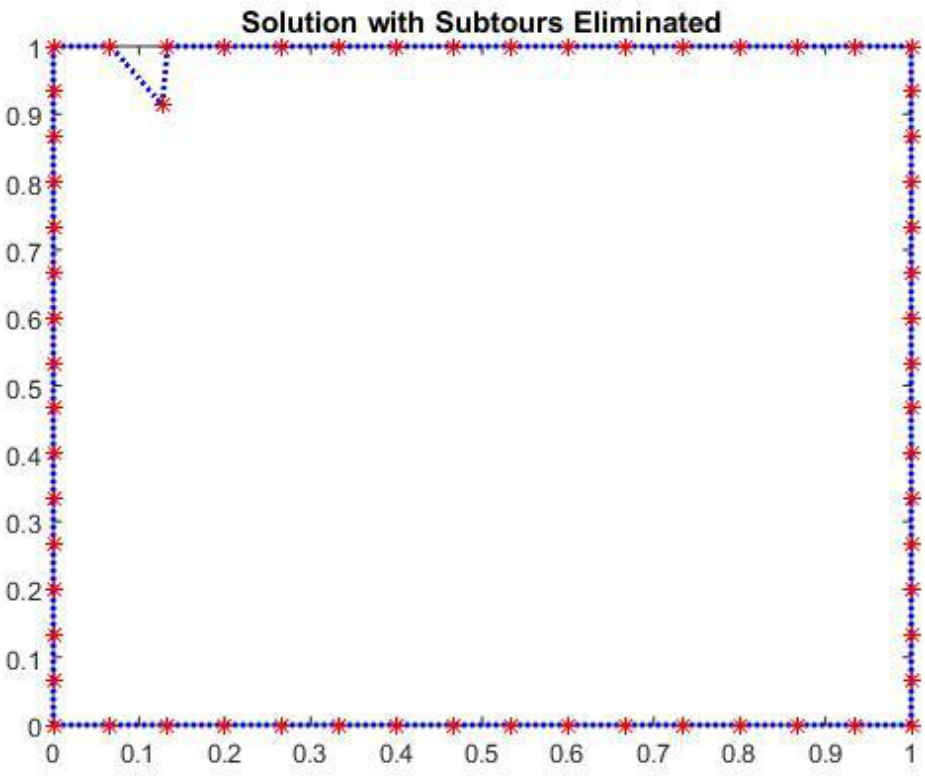

Figure 4.2 The TSP optimization with one object

While the number of objects keeps increasing, the complexity of the TSP graphs increases exponentially. When the complexity reflects on the performance of the TSP algorithm, sub tours appear more frequently in the simulation. To eliminate sub tours in the optimization, the algorithm needs to add more constraints and execute extra loops. In order to obtain reliable optimization, the time cost increases for algorithm execution.

Set the object number to 5. The object x, y-coordinate values are generated randomly. According to the map, there are various positions for the objects. It is possible to find the optimal path for one object by simply looking at the graph. However, when there are 5 or more objects on the

map, choosing the one with the shortest edge for one of the vertices can lead to missing the global minimum of the optimization.

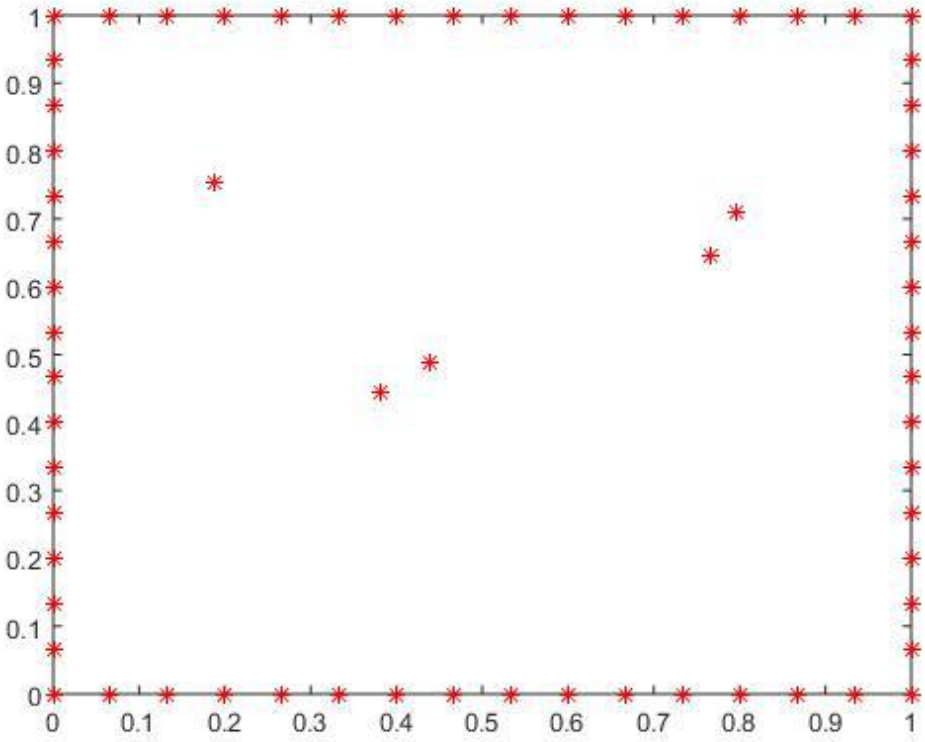

Figure 4.3 The vertex distribution map with five objects

When the algorithm handled the input data with 5 objects, sub tours appear in the early versions of the optimization.

To observe the sub tour edges that the algorithm chose, the three vertices at the middle of the map were isolated. The three vertices constructed a closed ring as a triangle, where one sub tour occurs. The two other vertices which represented objects connected with vertices on the wall, as another sub tour existed on the TSP simulation map.

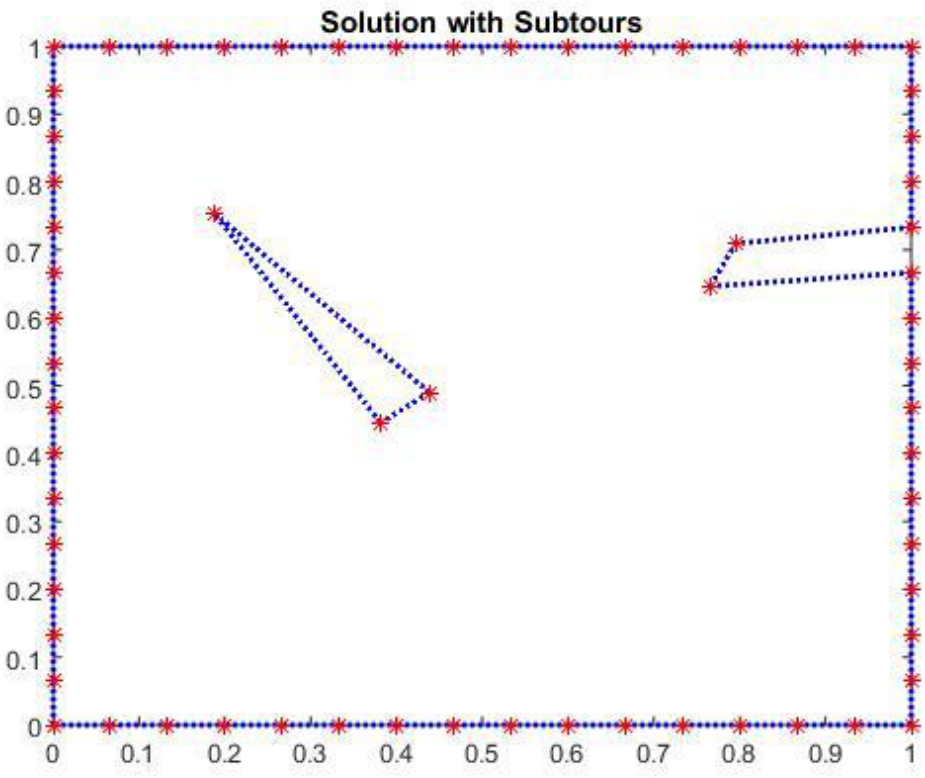

Figure 4.4 The TSP optimization of five objects at step 1

Two sub tours existed on the TSP graph. Two new constraints were added as inequality constraints. In order to construct a main ring, the edges of the sub tours would be moved away and the shape of the sub tours would be incomplete in the further optimization process of the algorithm.

However, one more loop of inequality constraints does not ensure successful sub tour elimination at the end of the loop. The old sub tours were eliminated by constraints, while the new sub tours can be created with the linear solver.

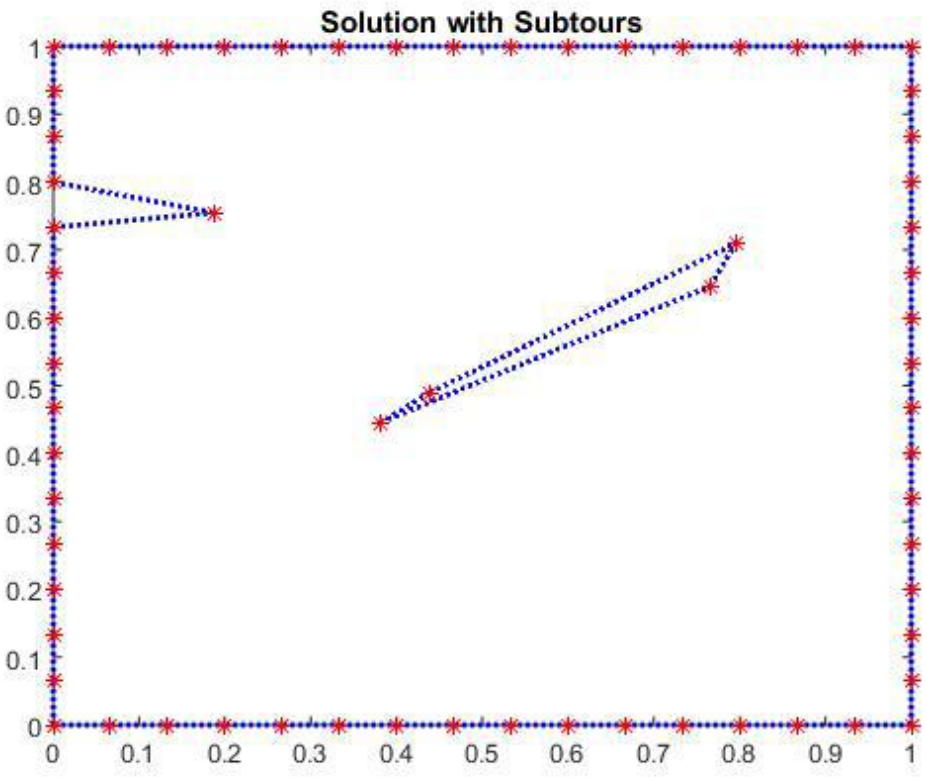

Figure 4.5 the TSP optimization of five objects at step 2

According to the optimization figure above, the algorithm chose to connect the four vertices in the middle part of the map as a sub tour. The vertex at the uppermost left part of the map is connected with the vertices around the wall, where another sub tour was found.

More constraints are added to the matrix and vector to optimize the result of the Traveling Salesman Problem. With the constraints stored, the algorithm is able to avoid the sub tours that have been already identified.

The figure below is the TSP optimization with the five objects in the middle part of the map. The sub tours were constructed twice by the linear-solver. With the two loops of sub tour elimination,

the algorithm works out an acceptable optimization of the Traveling Salesman Problem. However, when the amount of input data is large, more loops need to be executed for the unknown sub tours.

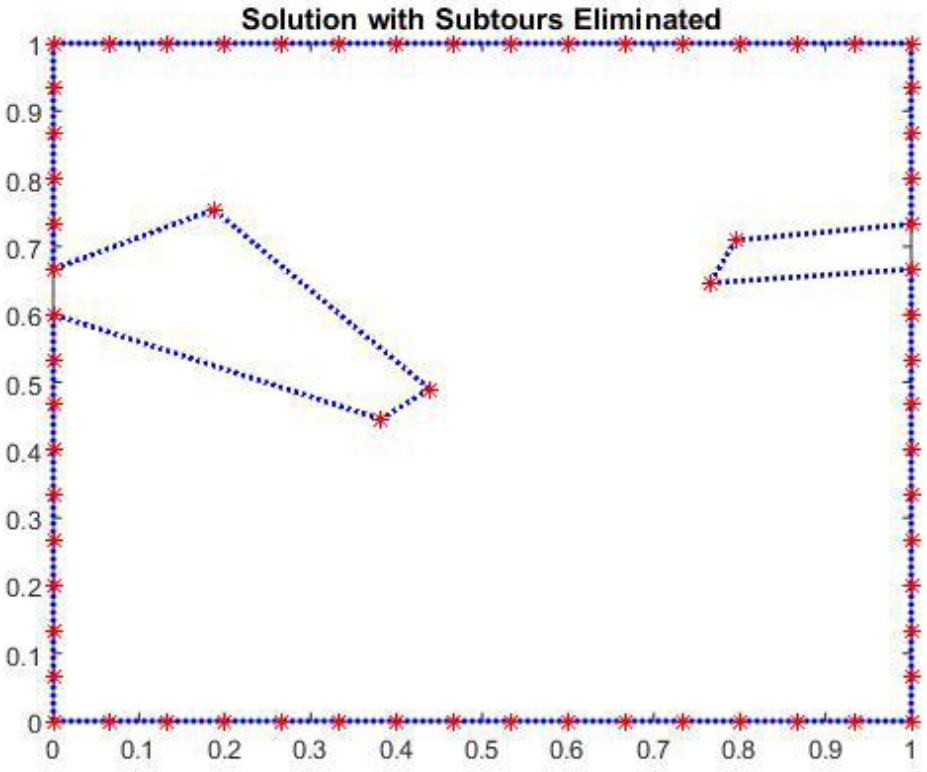

Figure 4.6 The optimization of the TSP with five objects

As mentioned before, when the number of vertices keeps growing, the increased time consumption can not be ignored. In this instance, the location of the object can also effect the performance of the algorithm.

When there are 10 objects on the map, the influence of object location can be observed. Comparing with the case where the object number is smaller than 10, the number of loops for

sub tour elimination increases, which reflects in an increase of time consumption. However, the number of loops that the algorithm needs for sub tour elimination is obviously different with various vertex distribution locations on the map.

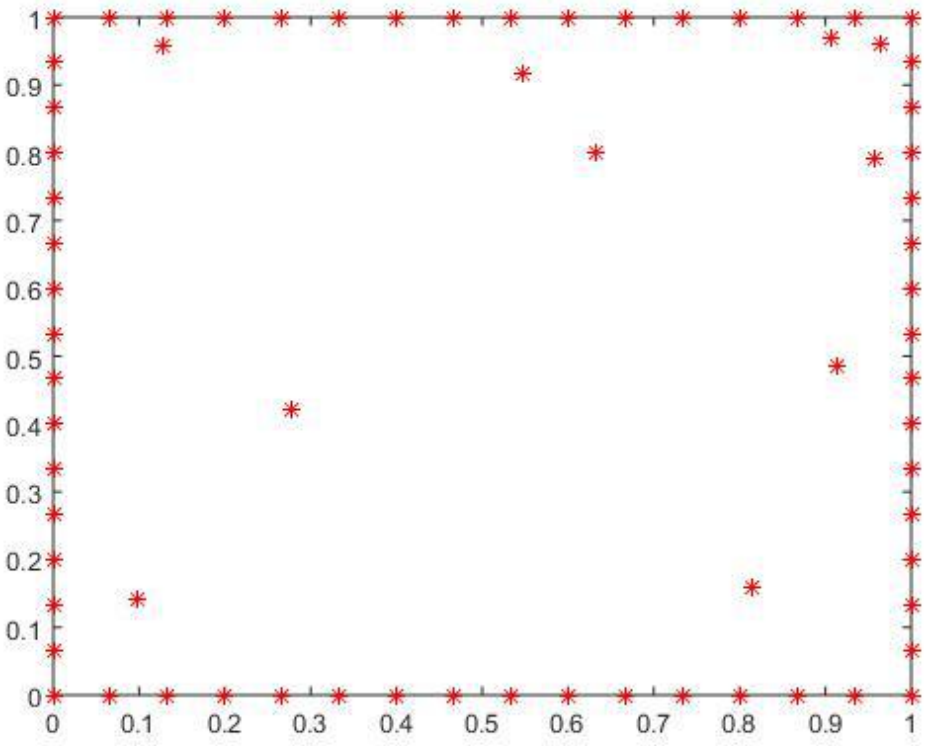

Figure 4.7 The TSP original map with ten objects in good case

The vertex distribution in the figure above can be seen as a sample of good cases. There are 10 objects located on the map. However, no sub tours appeared during the algorithm execution.

By observing the location of the vertices, it is obvious that most vertices are generated near the edge of the map. Assume that the difference of edge distance between object vertices is clear, so that the linear solver can work out the optimization with less time cost.

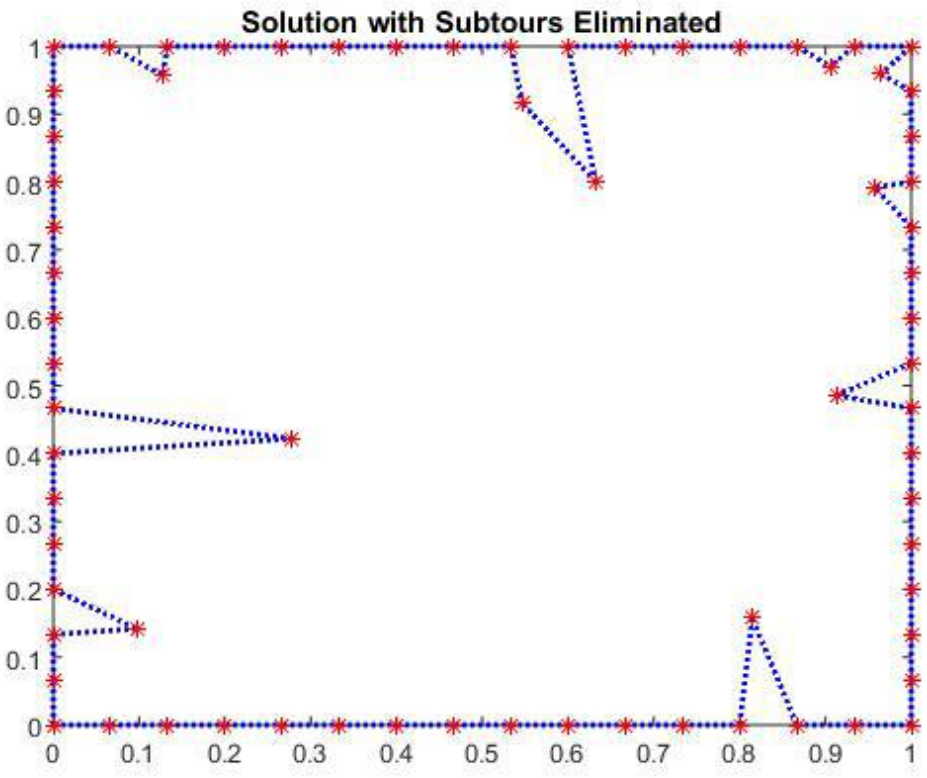

Figure 4.8. The TSP optimization with ten objects in good case

As good location of the vertices can reduce the time cost of the algorithm, there are also bad locations of vertex distribution, which lead to more loops to eliminate sub tours. The function linear solver is provided by Matlab. When bad cases occur from input data, the quality of the linear solver decides the time cost of the algorithm execution.

One sample of bad cases are shown below. All x, y coordinate values of 10 object vertices are generated randomly. However, there are vertices whose edge distance between each other is approximately similar. Assume this fact can lead to the result that more sub tours appear through the linear solver.

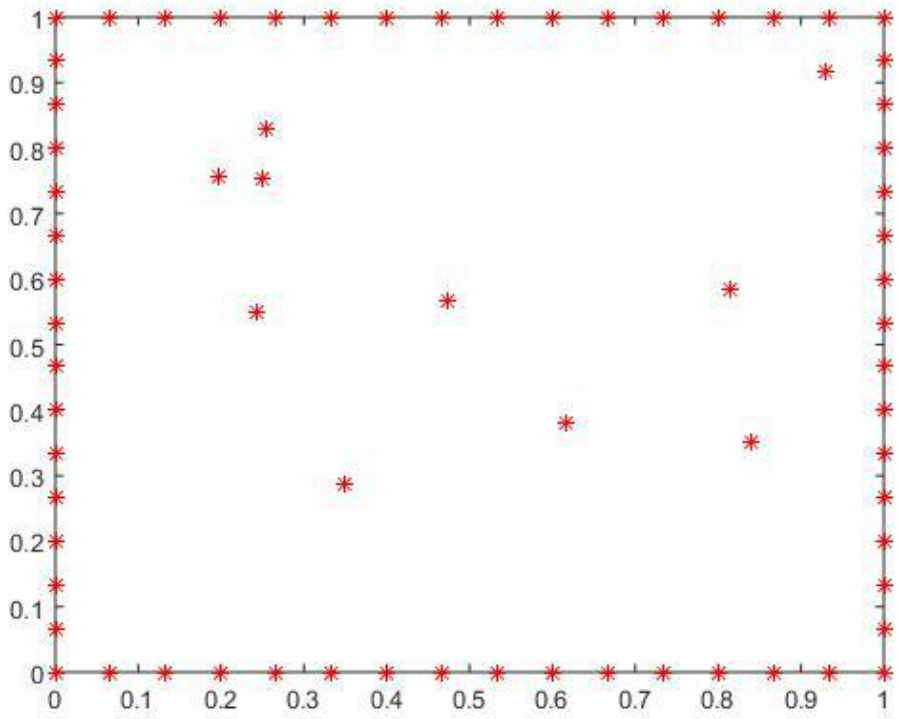

Figure 4.9 The TSP original map with ten objects in bad case

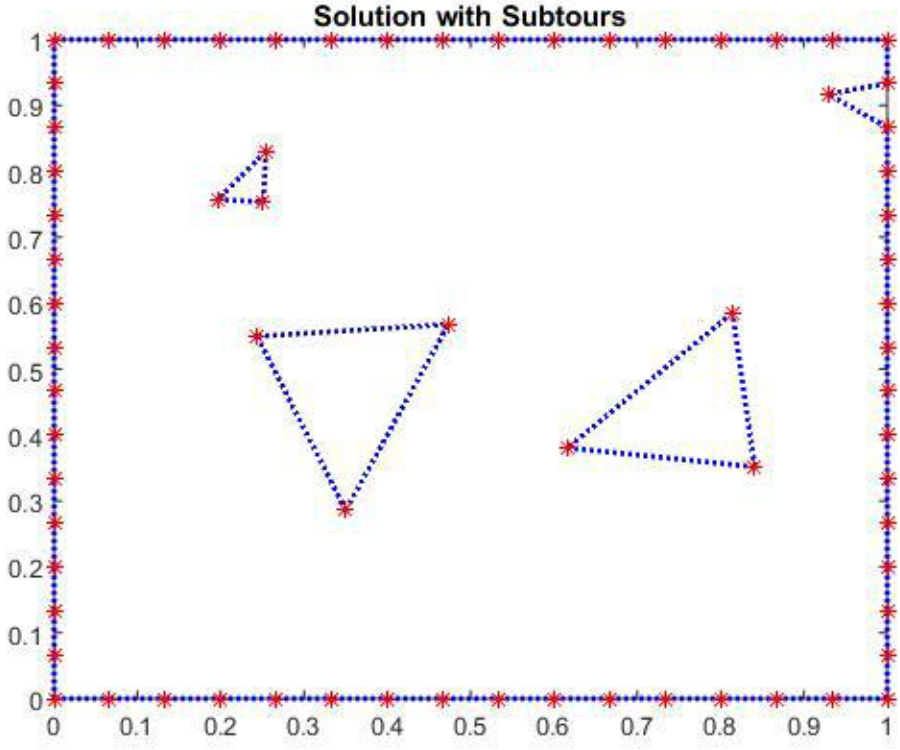

Figure 4.10 The TSP optimization in bad case at step 1

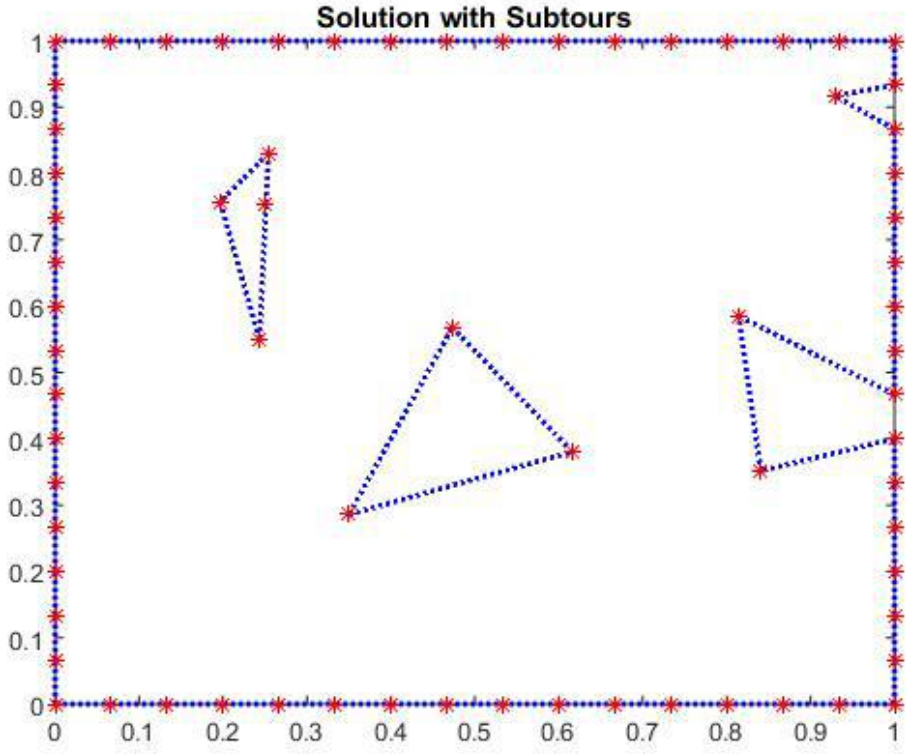

Figure 4.11 The TSP optimization in bad case at step 2

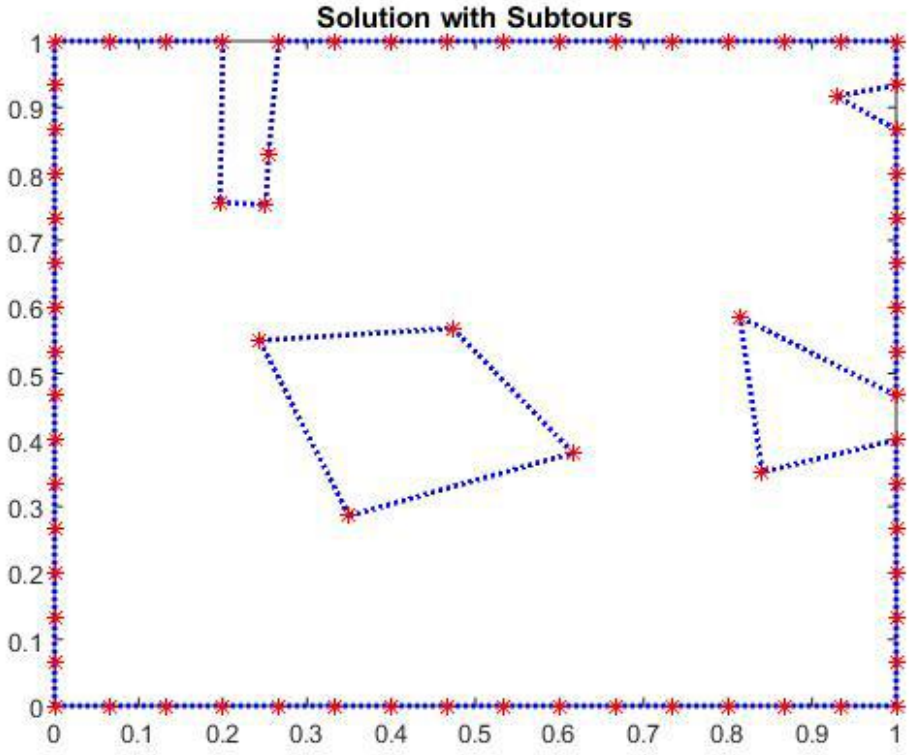

Figure 4.12 The TSP optimization in bad case at step 3

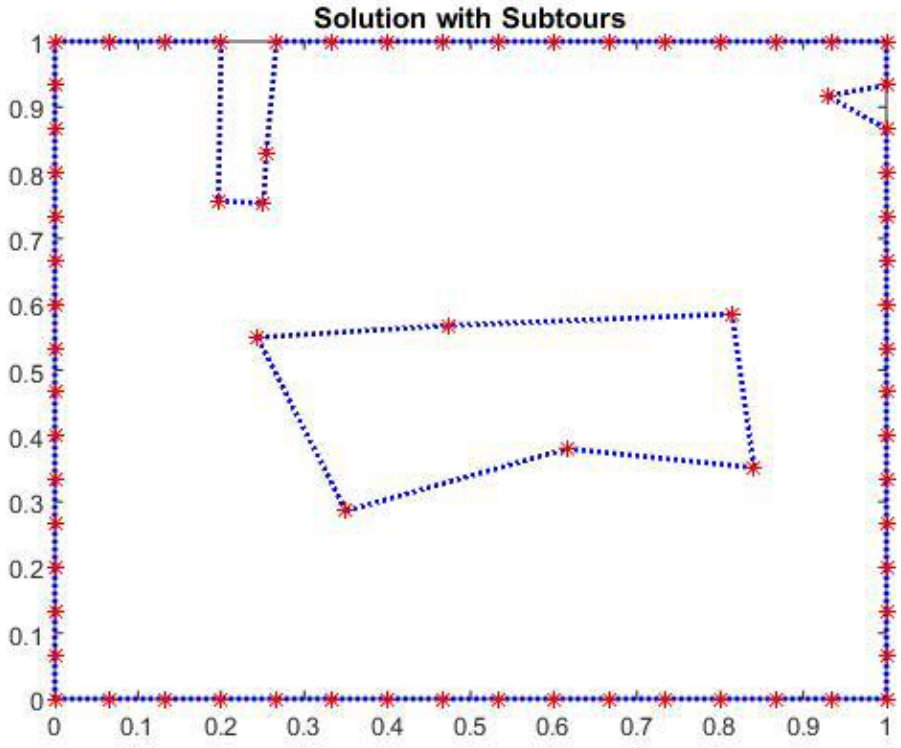

Figure 4.13 The TSP optimization in bad case at step 4

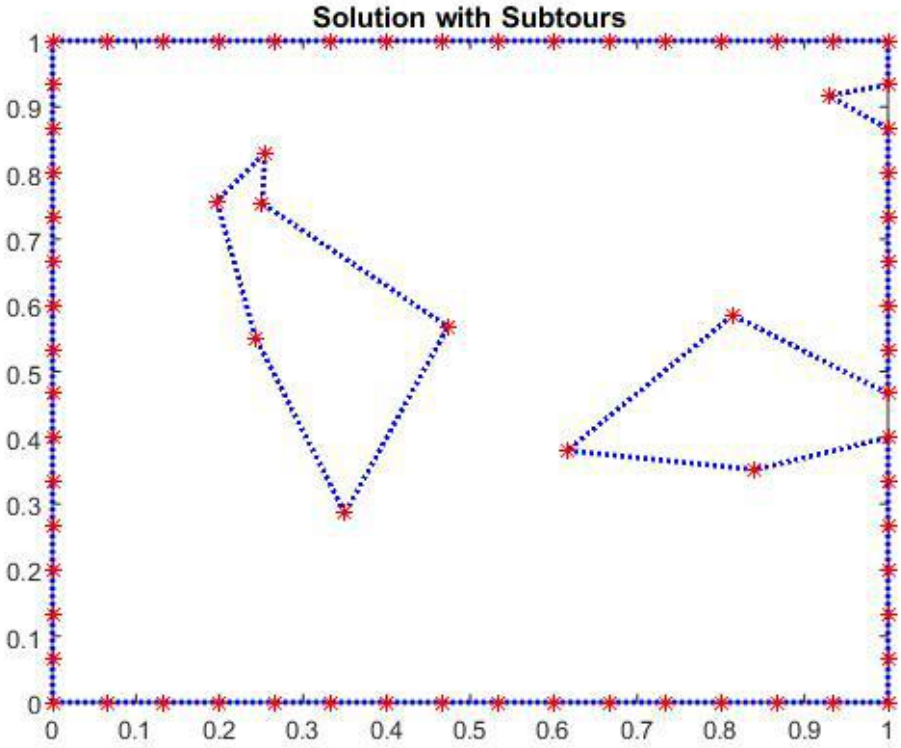

Figure 4.14 The TSP optimization in bad case at step 5

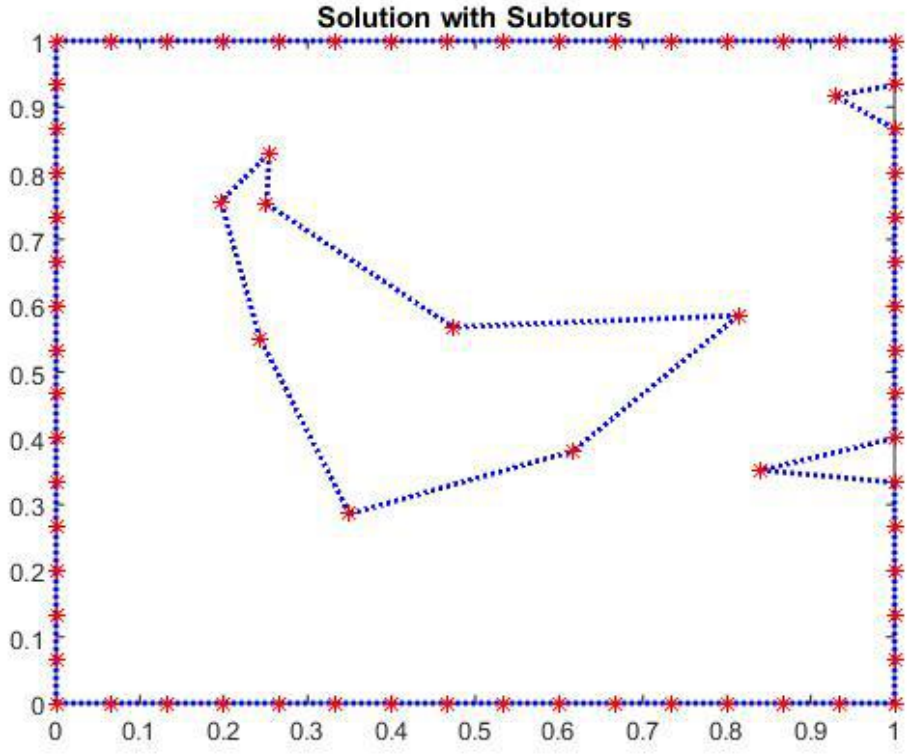

Figure 4.15 The TSP optimization in bad case at step 6

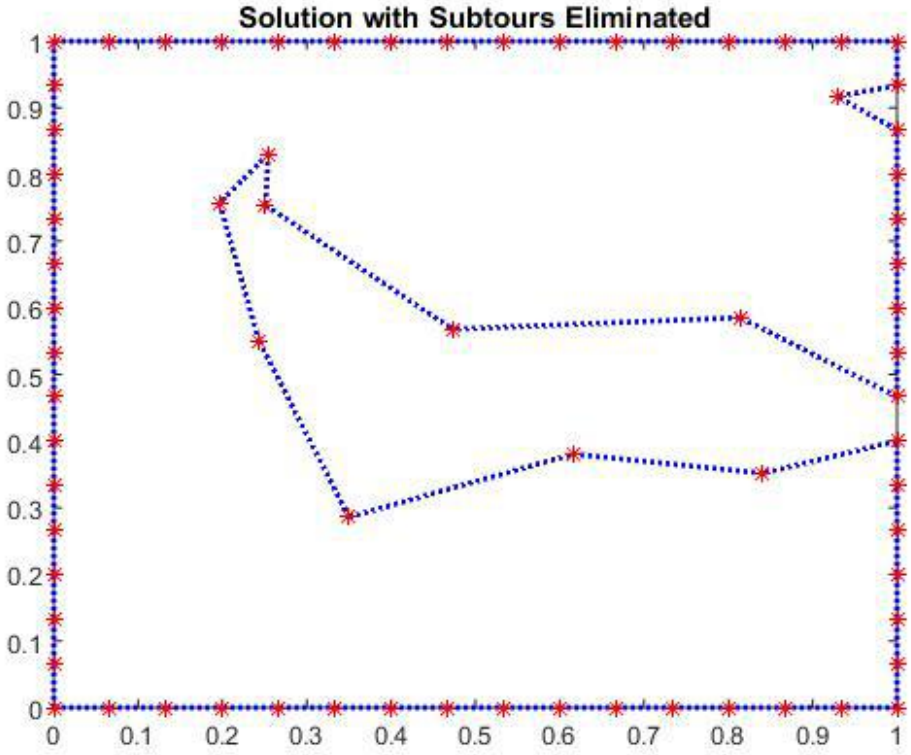

Figure 4.16 The TSP optimization eliminated sub tours in bad case

4.1.2 Time Consumption of Optimization

The location of the vertex distribution decides the time cost of the TSP algorithm execution. However, the vertex position is randomly generated. Random means that the vertex distribution can be good or bad according to probability. According to the times of algorithm execution, the time cost of various number of objects on the map can be observed with data statistics.

Set the number of objects from 1 to 10. Then generate the x, y - coordinates of the objects with random value. Set the algorithm execution times to 50. The matrices and vectors are created to store the x, y-coordinate values of vertices, the sequence order of optimization vertices and the time consumption.

The complexity of the algorithm itself decides the growing trend of time cost according to the change of input data amount. At the same time, the hardware of the PC which supports Matlab simulation decides the actual time cost.

The simulation is running on a laptop.

Logo: Dell

CPU: Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 1.60GHz

RAM: 4.00GB

System: Windows 7 64 bits

Table 4.1 Time consumption of various vertices distribution

/s	1	2	3	4	5	6	7	8	9	10
min	0.047	0.046	0.062	0.047	0.046	0.062	0.062	0.062	0.047	0.063
max	0.14	0.312	2.356	2.528	4.009	17.02	21.512	30.202	93.63	37.15
average	0.0687	0.0742	0.1452	0.2505	0.5424	1.9251	1.6192	4.6962	5.4601	5.3457
median	0.063	0.063	0.078	0.078	0.164	0.694	0.733	0.9285	1.7705	1.7315

As the formula shown above, with the object number from 1 to 10, the minimum value of time cost keeps low. After 50 times of execution, there is a possibility of a good case occurring. Random nodes can be located at a good place. Facing good cases, the algorithm performs well, and the execution time is low.

However, the max value of time cost shows the difference between various numbers of objects set on a map. When objects are more than six, time cost increases significantly. The time difference shows a bad case of vertex distribution. Along with the number of object vertices, they can both increase the complexity of the Traveling Salesman Problem optimization.

To calculate the average time cost, both minimum and maximum value are considered. The average value of time cost shows a trend with the object number increasing. However, minimum value and maximum value are both regarded as extreme values. Extreme values can affect average value of time cost in statistic. Median value is selected according to the amount of data. When an extreme value exists in a formula, a median value can stand for the trend of time cost.

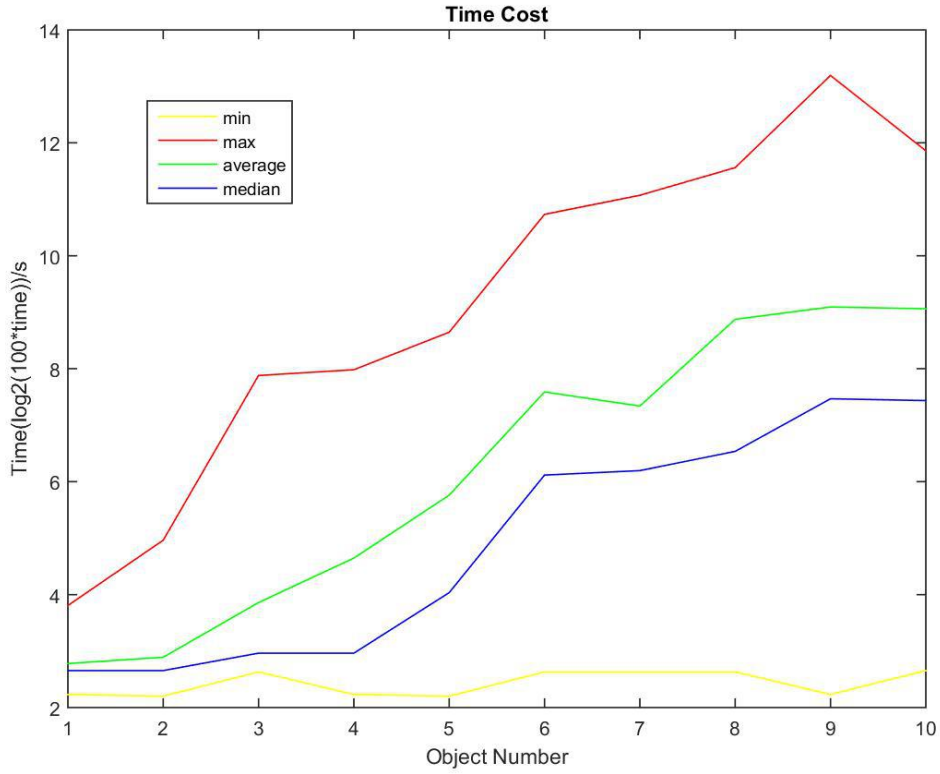

Figure 4.17 Time cost of the TSP optimization

When the object number increases from 1 to 10, time cost has increased more than 20 times.

4.1.3 Local minimum and global minimum of the TSP Optimization

The distribution of the RFID tags is decided by the layout of the room. Comparing with the condition that all vertices are generated randomly in simulation, the distribution is still different.

With the linear solver, the algorithm reaches optimization of the Traveling Salesman Problem. When people see the vertex distribution on the map, the edges along the wall are chosen with eye view. When the same vertex distribution is given on the map, the algorithm calculates the optimal result and people select the route with their own judgment. Comparing with the TSP

optimization chosen by the algorithm and people on both sides, there are differences in a few details.

Set the object vertex number to 10 and create a new map on Matlab. According to the vertex distribution on the figure below, the vertex relation is complex.

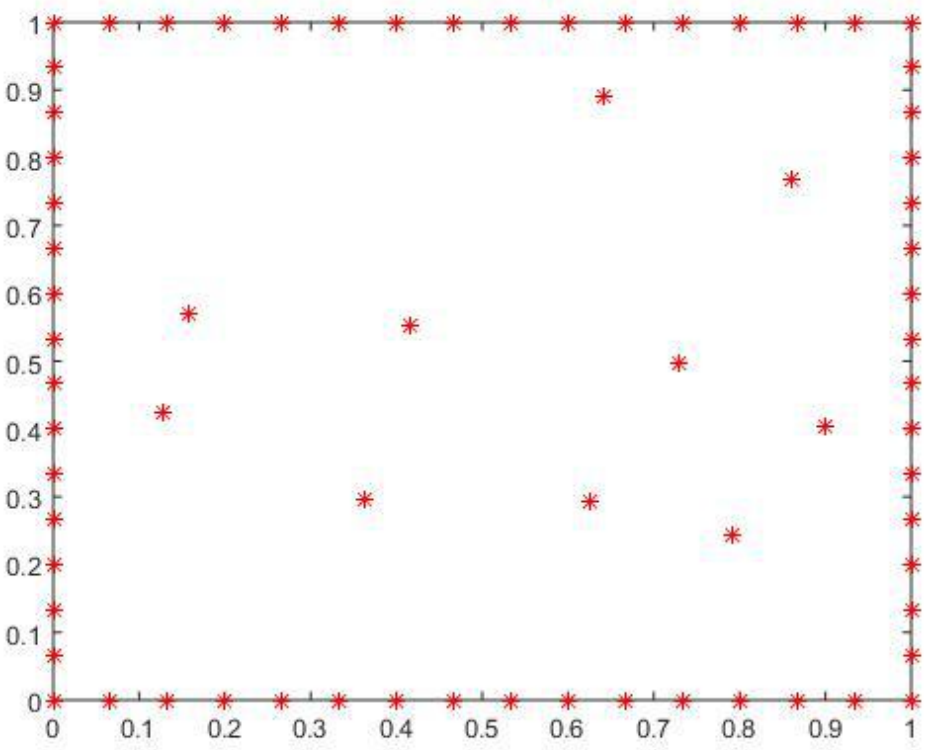

Figure 4.18 The original map of the TSP Vertex distribution

Five participants agreed to draw the shortest route that connect all vertices on the map. The participants chose the graph edges without calculations. To save time and ensure consistency of the original map, all figures are sent through the Internet. The original feedback figures are shown below.

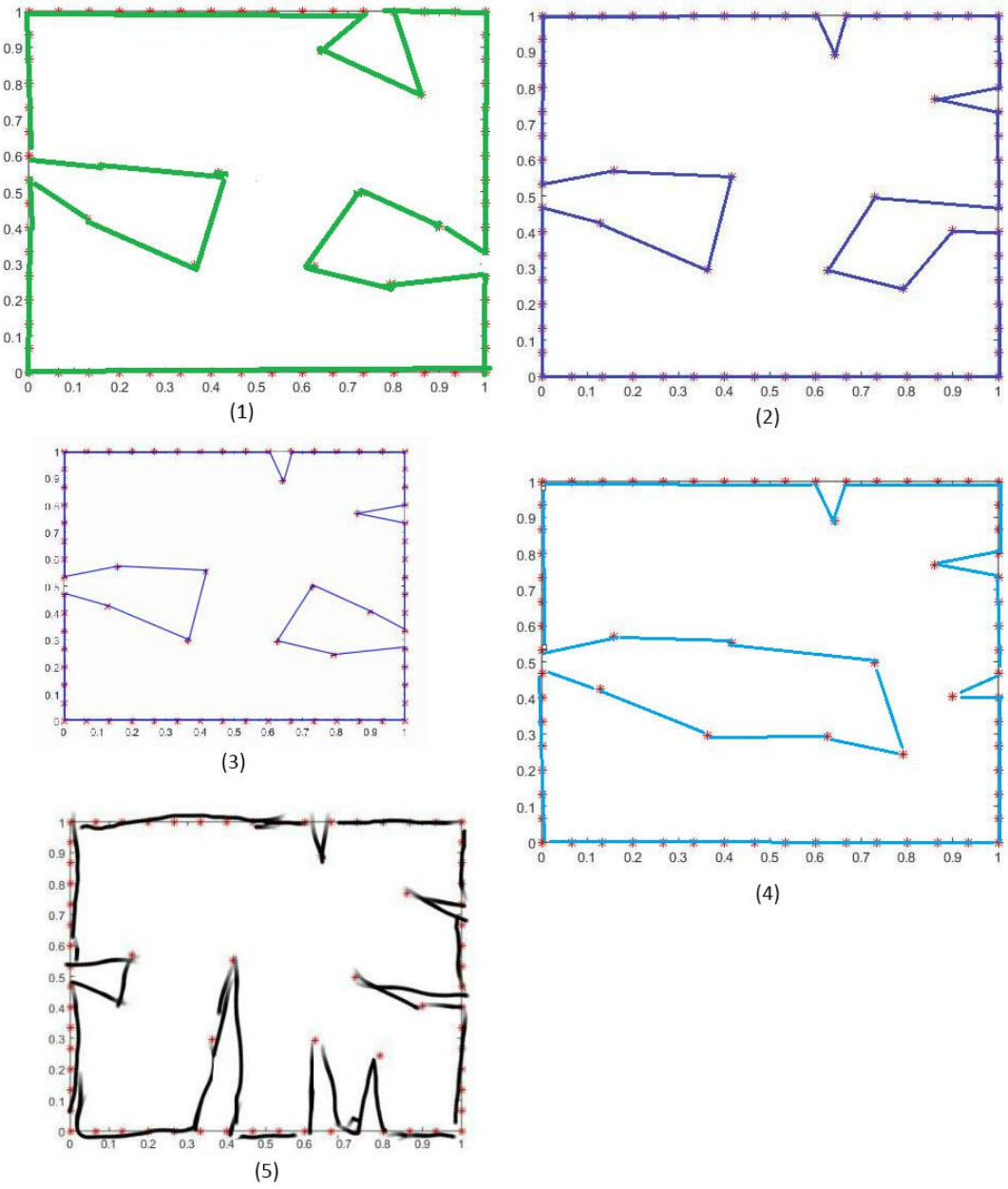

Figure 4.19 The optimal routes chosen by five participants

With the Matlab process, all feedback figures are redrawn and displayed as the figures below. The total distance of each figure is calculated. According to the distance value formula shown,

the result from the Matlab algorithm is optimal, while other routes are approaching the optimization of the TSP.

Table 4.2 Distance of the different TSP optimal routes

(1)	(2)	(3)	(4)	(5)	algorithm
6.4790	6.3026	6.2631	6.3062	7.2871	6.2326

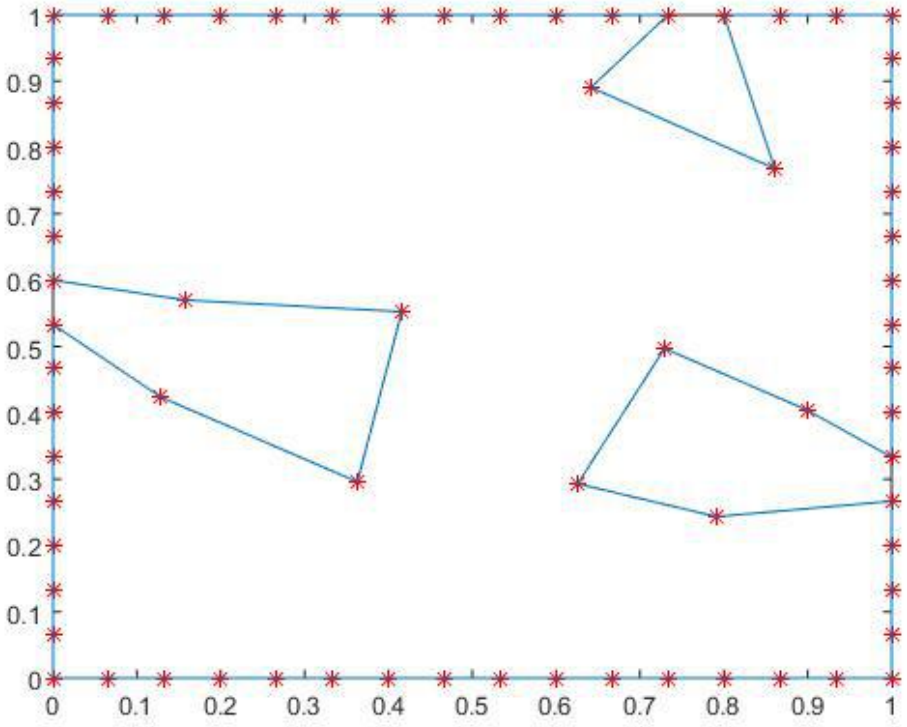

Figure 4.20 The redrawn optimal route by participant 1

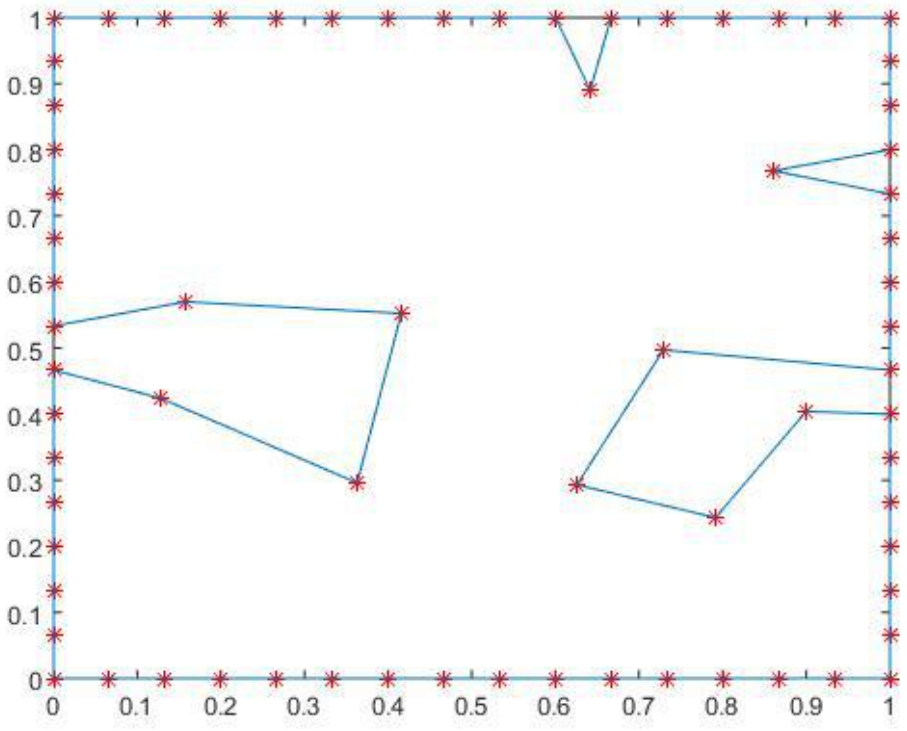

Figure 4.21 The redrawn optimal route by participant 2

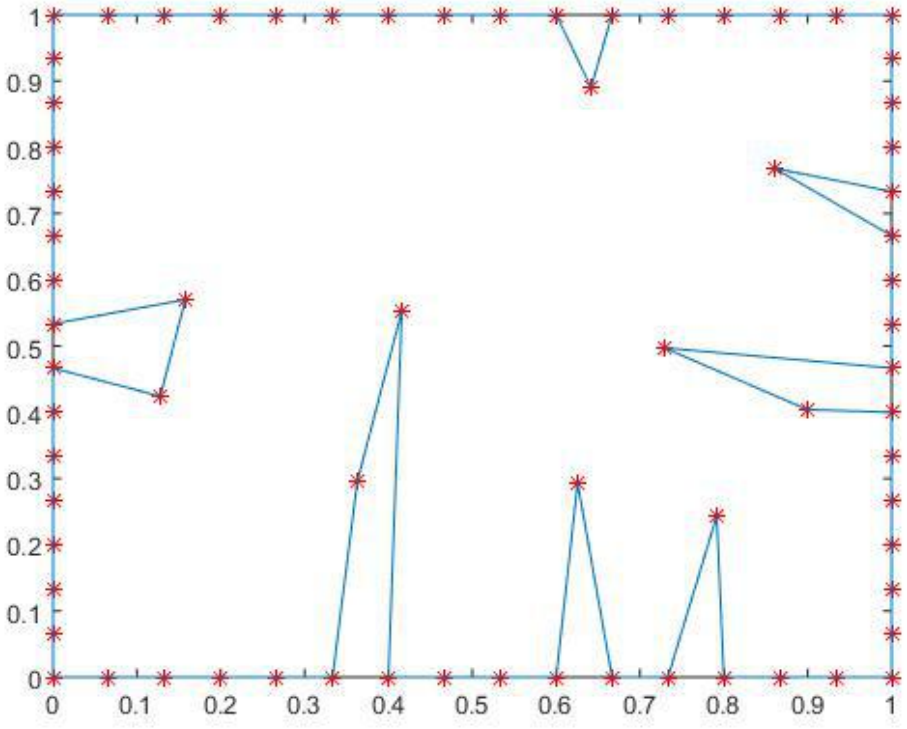

Figure 4.22 The redrawn optimal route by participant 3

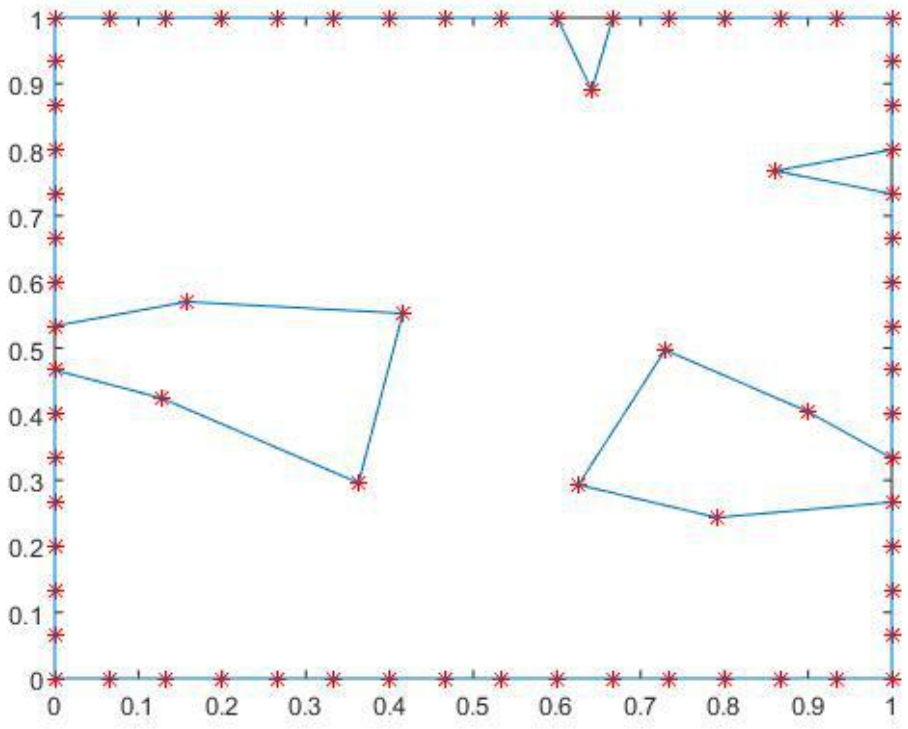

Figure 4.23 The redrawn optimal route by participant 4

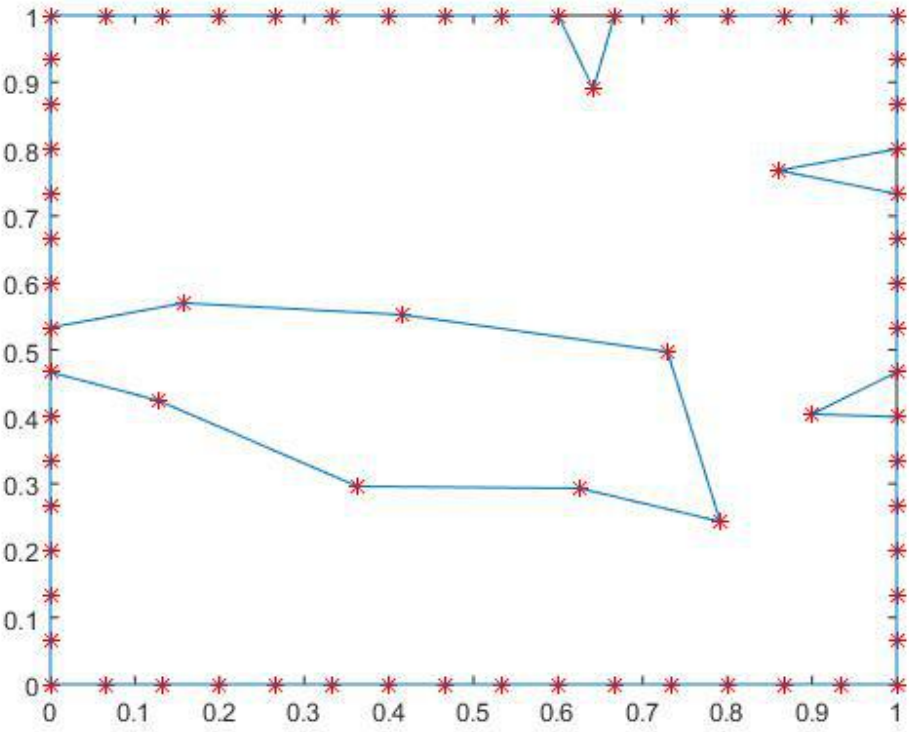

Figure 4.24 The redrawn optimal route by participant 5

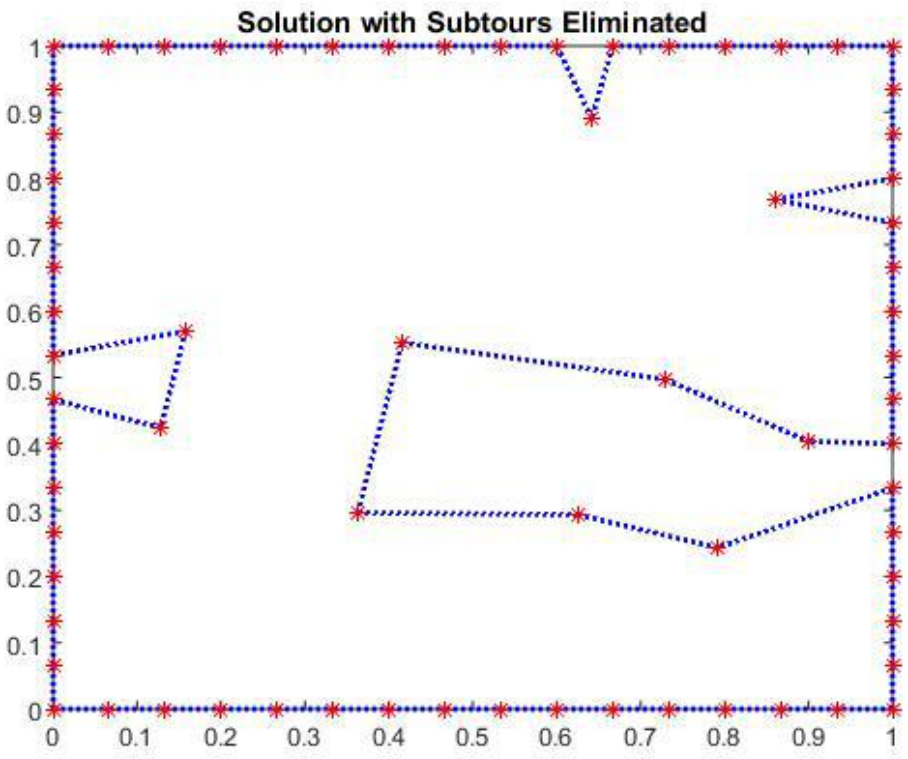

Figure 4.25 The TSP optimization by Matlab

The TSP optimization by Matlab is shown above. Comparing the optimal routes drawn by people and the algorithm, the algorithm chose the edges through calculation, while people chose the edges by dividing vertices on separated map parts.

Observing feedback figures, when vertices are located near the wall edge, people tend to connect vertices with the nearest wall. According to the feedback figures, the vertices in the blue circles are judged as vertices near the wall.

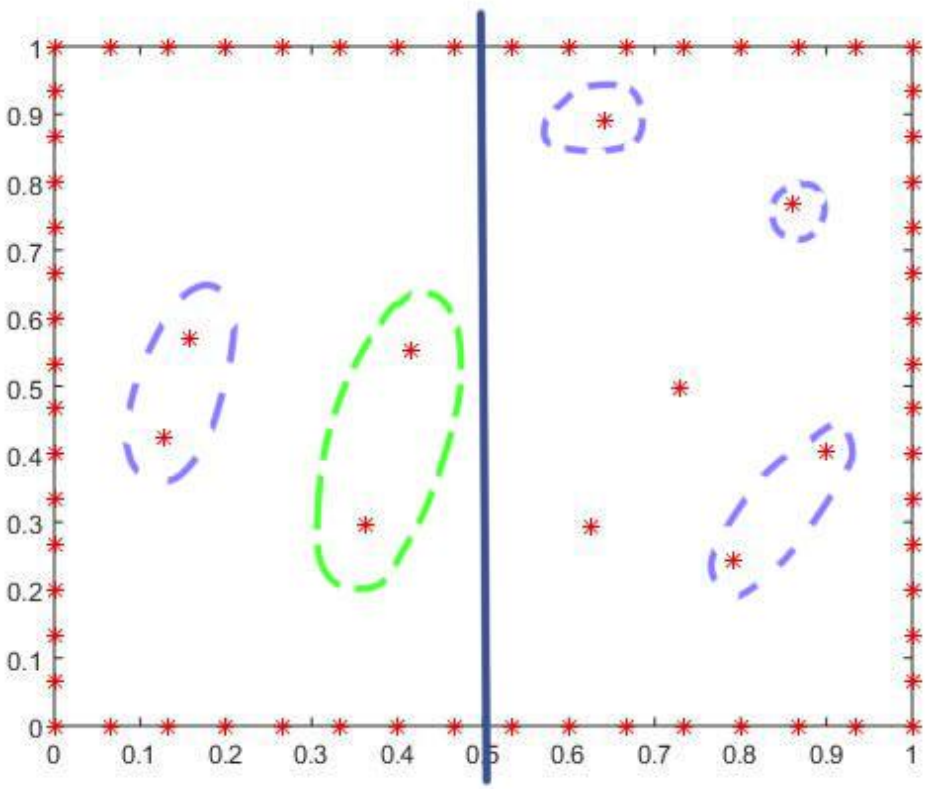

Figure 4.26 Object vertices in different parts

When the vertices are in the middle part of the map, it is hard to make a decision based on eye view only. The vertices in the green circle are at the lower left part of the map. In people's way of thinking, those two vertices are first considered to connect with the edges at the left part of the map. According to all routes drawn by participants, no choice was made to connect the two vertices with the wall on the right part of the map.

According to the TSP optimization by Matlab, the algorithm chose the opposite. People make decisions from one vertex to another vertex, the same as when selecting a local minimum. However, the sum of the local minimum distance in the TSP may not be equal to the global

optimization. The algorithm is capable of the necessary calculation to solve the TSP for the global circumstance.

4.2 Performance of the Traveling Salesman Problem with Neighbors

After accomplishing the simulation with vertices that represent tags on the wall and the object center, the next stage is to examine the algorithm performance for objects with their sub maps.

Assume that the objects do not take a large area on the map. Each object is small comparing to the space of the lab. The distance from the tags on the object edge to the center of the object is minimal, so that the distance from the wall to the center of the object can turn into the distance from the wall to the object edge approximately.

In the RFID lab, when a robot navigates the environment, the objects may be tables, shelves, and wardrobes. The shape of the objects can be a rectangle, a circle or an irregular polygon. However, in order to simplify map building, set the shape of all the objects as rectangular.

Random vertices are generated within the boundary of the rectangle. In software simulation, assume that there are five RFID tags on each object and every tag is at a different place. Set object number N = 1

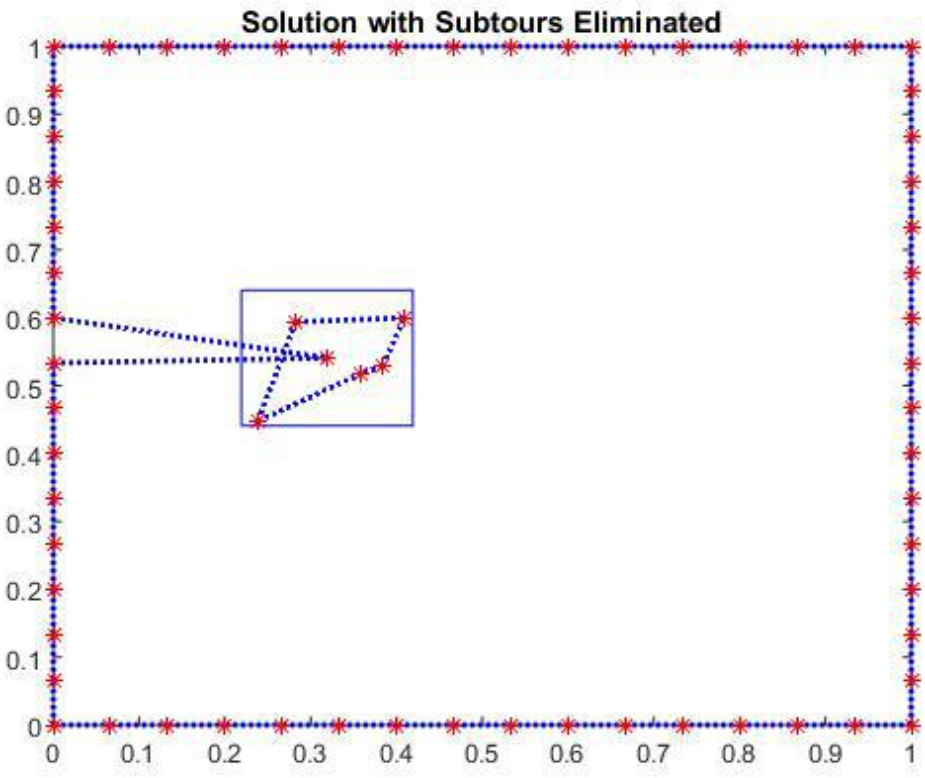

Figure 4.27 The Optimization of the TSP with Neighborhoods with one object

According to the figure above, the TSP optimization is constructed on the main map. The sub map for the RIFD tags on the object is in the area of the rectangle. The algorithm reaches the TSP optimal route on the sub map as the same process on the main map.

Set object number N = 5.

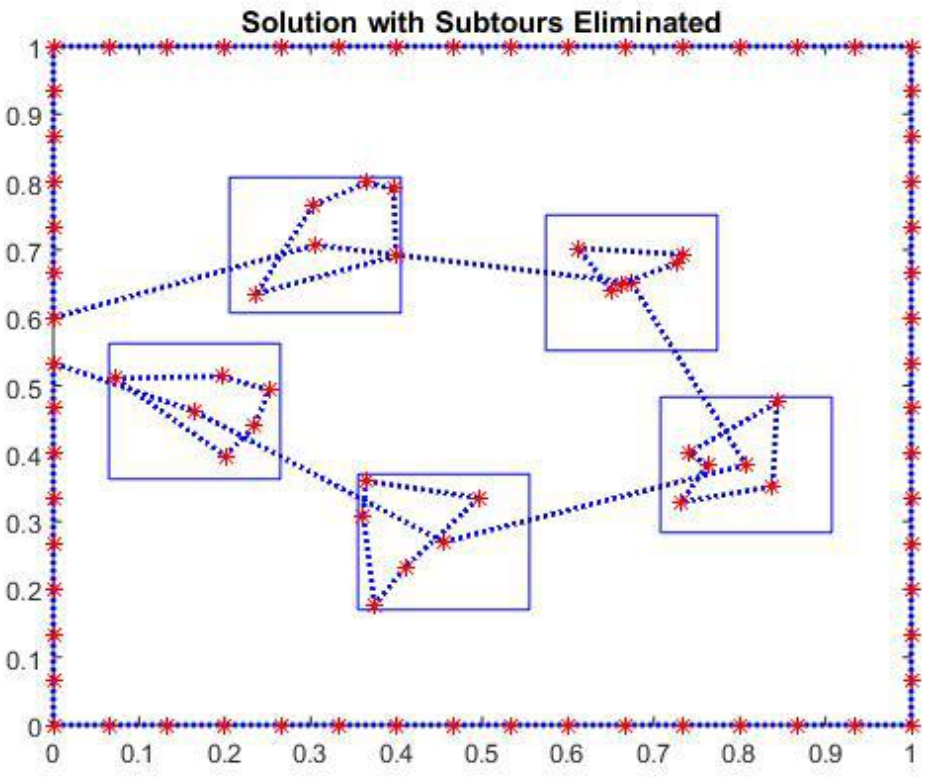

Figure 4.28 The Optimization of the TSP with Neighborhoods with five objects

Comparing with the optimization with one object on the map, the figure above is more complicated. In the Traveling Salesman Problem with Neighborhoods, one more object added to the map means one more sub map that the algorithm needs to handle. In every sub map, when the tag number increases, so does the complexity of the TSP graph. The object vertex number decides the sub map number. The number of tags on the objects decides the vertices on every sub map. In the process of solving the Traveling Salesman Problem with Neighborhoods, both factors can affect the performance of the algorithm.

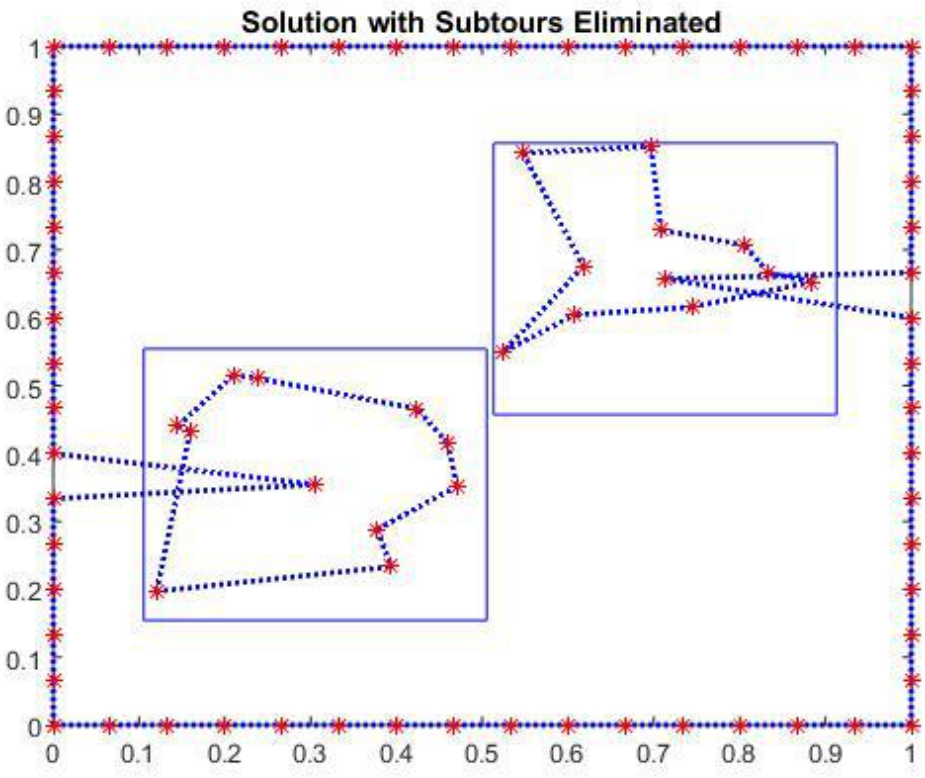

Figure 4.29 The Optimization of the TSP with Neighborhoods with two big objects

While the tags on the wall and the tags on the objects are in two maps, the TSP algorithm can not calculate the distance between nodes that are not on a same map. When the object size becomes larger, just as the figure shown above, the shortest distance from the wall to the center of the objects can not be equal to the real shortest distance from the wall to the tags on the edge of the objects.

However, the map can be divided into various zones for the corresponding functions. Before moving to other zones, it is necessary for the robots to complete navigation in the current zone. TSP with Neighborhoods for robot navigation can be used in retail environments. The sales frequency for different commodities can vary. The frequency for robots to navigate zones is

different as well. When the object size becomes large, even though the optimization of TSP with Neighborhoods through the main map and the sub maps may not be optimal with only one map, the solution is still acceptable.

5. CONCLUSION

The RFID readers and tags provide hardware support for robot navigation. During robot navigation, the route choice of the robots turns into the Traveling Salesman Problem model. In simulation, the function linear solver is used for the optimal desired result reducing the route distance successfully. The cost of hardware and time increases parallel to the increase of the amount of input data. Since the Traveling Salesman Problem belongs to the NP-hard category, the complexity of the algorithm for optimization for the TSP can not be ignored.

Comparing the optimal routes chosen by the participants and by the algorithm, the route chosen by the algorithm wins out. As shown in the mathematical model of the application in a retail environment, robots navigate according to the optimization of the Traveling Salesman Problem, which improves efficiency and saves labor cost.

REFERENCES

[1] Thirumurugan, J., et al. "Line following robot for library inventory management system." *Emerging Trends in Robotics and Communication Technologies (INTERACT), 2010 International Conference on*. IEEE, 2010.

[2] Siegwart R, Nourbakhsh I R, Scaramuzza D. Introduction to autonomous mobile robots[M]. MIT press, 2011.

[3] Miah M S, Gueaieb W. An RFID-Based robot navigation system with a customized RFID tag architecture[C]//Microelectronics, 2007. ICM 2007. Internatonal Conference on. IEEE, 2007: 25-30.

[4] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning: Sensors and techniques," Journal of Robotic Systems, vol. 14, no. 4, pp. 231–249, April 1997.

[5] L. Ojeda, D. Cruz, G. Reina, and J. Borenstein, "Current-based slippage detection and odometry correction for mobile robots and planetary rovers," IEEE Transactions on Robotics, vol. 22, no. 2, pp. 366–378, April 2006.

[6] Lee Y C, Chae H, Kim S H. Applications of robot navigation based on artificial landmark in large scale public space[C]//Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on. IEEE, 2011: 721-726.

[7] Mardiyanto R, Anggoro J, Budiman F. 2D map creator for robot navigation by utilizing Kinect and rotary encoder[C]//Intelligent Technology and Its Applications (ISITIA), 2015 International Seminar on. IEEE, 2015: 81-84.

[8] Olszewski B, Fenton S, Tworek B, et al. RFID positioning robot: An indoor navigation system[C]//Electro/Information Technology (EIT), 2013 IEEE International Conference on. IEEE, 2013: 1-6.

[9] Miah M, Gueaieb W. Indoor robot navigation through intelligent processing of RFID signal measurements[C]//Autonomous and Intelligent Systems (AIS), 2010 International Conference on. IEEE, 2010: 1-6.

[10] Miah M, Gueaieb W. A stochastic approach of mobile robot navigation using customized rfid systems[C]//Signals, Circuits and Systems (SCS), 2009 3rd International Conference on. IEEE, 2009: 1-6.

[11] Kirkpatrick, S.; Gelatt Jr, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science 220 (4598): 671–680. Bibcode:1983Sci...220..671K. doi:10.1126/science.220.4598.671. JSTOR 1690046. PMID 17813860.

[12] Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". Journal of Optimization Theory and Applications 45: 41–51. doi:10.1007/BF00940812.

[13] X. Bian, and L. Mi, “Development on genetic algorithm theory and its applications,” Application Research of Computers, vol. 27, no. 7, pp. 2425–2429, 2010.

[14] Yu, Yingying, Yan Chen, and Taoying Li. "A new design of genetic algorithm for solving TSP." *Computational Sciences and Optimization (CSO), 2011 Fourth International Joint Conference on*. IEEE, 2011.

[15] Dorigo, Marco, and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem." Evolutionary Computation, IEEE Transactions on 1.1 (1997): 53-66.

[16] Zhao Y, Xiao Z, Kang J. Optimization design based on improved ant colony algorithm for PID parameters of BP neural network[C]//Informatics in Control, Automation and Robotics (CAR), 2010 2nd International Asia Conference on. IEEE, 2010, 3: 5-8.

[17] Kennedy, J.; Eberhart, R. (1995). "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. pp. 1942–1948.

[18] Wang X H, Li J J. Hybrid particle swarm optimization with simulated annealing[C]//Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on. IEEE, 2004, 4: 2402-2405.

[19] Samuel G G, Rajan A, Christober C. Hybrid Particle Swarm Optimization—Genetic algorithm and Particle Swarm Optimization—Evolutionary programming for long-term generation maintenance scheduling[C]//Renewable Energy and Sustainable Energy (ICRESE), 2013 International Conference on. IEEE, 2013: 227-232.

[20] Ye G, Rui X. An improved simulated annealing and genetic algorithm for TSP[C]//Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on. IEEE, 2013: 6-9.

[21] Vicencio K, Davis B, Gentilini I. Multi-goal path planning based on the generalized traveling salesman problem with neighborhoods[C]//Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. IEEE, 2014: 2985-2990.

[22] http://www.mathworks.com/help/optim/ug/intlinprog.html

[23] http://www.mathworks.com/help/optim/ug/travelling-salesman-problem.html