Path finding and flow control for virtual reality augmented evacuation system for large indoor areas

Feng, Feng

2006

Feng, F. (2006). Path finding and flow control for virtual reality augmented evacuation system for large indoor areas. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/39353>

<https://doi.org/10.32657/10356/39353>

**Path Finding and Flow Control
for Virtual Reality Augmented Evacuation System
for Large Indoor Areas**

Feng Feng

School of Civil & Environmental Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Master of Engineering

2006

Acknowledgements

This thesis is to systematically summarize my work during my two-year studies in Nanyang Technological University. And it is also a presentation of all the effort of me and my supervisor, Dr. Chen Po-Han, Assistant Professor of School of Civil & Environmental Engineering, Nanyang Technological University. I'm deeply indebted to his generous help. His liberal and encouraging guidance style allows me to enjoy the happiness so much from the research work.

Thanks to Dr. Duh Been-Lirn, Assistant Professor of School of Mechanical and Aerospace Engineering, Nanyang Technological University. And thanks to Dr. Hsu Wen Jing, Associate Professor of School of Computer Engineering, Nanyang Technological University. They gave me valuable suggestions on algorithm development and system integration.

Thanks to the friendly academic environment of the School of CEE and the scholarship provided by NTU.

Thanks to my parents and dear wife. Their encouragement always blesses me to get through the challenges in my life.

Abstract

Abstract

From Euler's seven-bridge problem to modern robot motion planning techniques, path finding has been a very important part of network analysis in operations research. And finding the shortest path between two nodes in a network is a classical problem in a number of research areas. Although some general one-source-to-one-target shortest path algorithms have already been developed, such as Dijkstra's shortest path algorithm, in some cases, some alternative algorithms with lower computation complexity may be preferred due to the properties of real tasks.

The algorithms in this thesis are developed for a real-time evacuation system which is on the platform of a 3D Virtual Reality augmented environment, for which some fast shortest path algorithm is to be developed. In this order, this thesis will analyze some common inside-building topologies first, and then developed a fast shortest path algorithm for the topologies, which frequently appear in indoor areas, such as campus and commercial buildings. With the much lower computation complexity than Dijkstra's shortest path algorithm, this algorithm can be a useful tool for real-time optimal path finding in Euclidean networks. (Euclidean networks refer to the networks whose edge weights are assigned based on their Euclidean lengths).

Some flow control techniques have also been discussed; and a fast evacuation flow control algorithm has been developed as well. Compared to previous evacuation flow control algorithms, this one is more suitable for real-time systems due to its fast computing speed. A test bed evacuation system has been built in order to implement this algorithm.

Optimal path finding and flow control algorithms can provide very practical utilities for an evacuation system, and make the system a good tool for emergency rescuing tasks. The research in this thesis will give a deep insight in this aspect.

Table of Contents

Acknowledgements i

Abstract ii

Table of Contents iii

List of Figures v i

List of Tables ix

List of Acronyms and Abbreviations x

Chapter 1: Introduction 1

1.1 Background.....1

1.1.1 Preliminaries of VR and AR.....1

1.1.2 Large Indoor Area Evacuation Aid System Augmented by Virtual Reality.....3

1.2 Problem Statement.....4

1.3 Research Objectives.....6

1.4 Organization of the Report.....6

Chapter 2: Literature Review 8

2.1 Introduction.....8

2.2 Previous Works in VR and AR.....8

2.2.1 Introduction.....8

2.2.2 Indoor Position Tracking.....9

2.2.3 System Components of General VR Systems.....11

2.2.4 Problems Raised by VR Systems.....14

2.3 Free Space Conversion.....15

2.3.1 Introduction.....15

2.3.2 Exact Cell Decomposition.....16

2.3.3 Road Map Methods.....19

2.3.3.1 Visibility Graphs.....19

2.3.3.2 Voronoi Diagram.....22

2.4 Dijkstra’s Shortest Path Algorithm.....25

2.5 Shortest Route Computation in Distributed Systems.....29

2.6 Interval Routing Scheme.....34

2.7 Flow Control Methods.....37

Chapter 3: Research Methodology 40

Table of Contents

3.1 Flow of Research Work.....	40
3.2 Preliminary Stage.....	41
3.3 System Development Stage.....	41
3.4 Final Stage.....	42
Chapter 4: Shortest Path Finding Algorithm	43
4.1 Introduction.....	43
4.2 Spatial Map Diagrams of Common Indoor Structures.....	45
4.2.1 2D Grids.....	45
4.2.2 3D Grids.....	47
4.2.3 Trees and Rings.....	50
4.2.4 Other Structures.....	51
4.3 Dynamic Shortest Path Algorithm for 2D Grids.....	53
4.3.1 IRS for 2D Grids – Optimal 1-LIRS.....	53
4.3.2 Static Algorithm.....	55
4.3.3 Dynamic Algorithm.....	55
4.4 Dynamic Shortest Path Algorithm for 3D Grids.....	58
4.4.1 IRS for 3D Grids – Optimal 1-LIRS.....	58
4.4.2 Static Algorithm.....	59
4.4.3 Dynamic Algorithm.....	59
4.5 Dynamic Shortest Path Algorithm for trees.....	60
4.5.1 IRS for Trees – Optimal 1-CIRS.....	60
4.5.2 Dynamic Algorithm.....	61
4.6 Dynamic Shortest Path Algorithm for Rings.....	63
4.6.1 IRS for Rings – Optimal 1-CIRS.....	63
4.6.2 Dynamic Algorithm.....	66
4.7 Computation Complexity of 1-IRS Dynamic Shortest Path Algorithm	68
4.7.1 Preliminaries on Computation Complexity.....	68
4.7.1.1 Space Complexity.....	68
4.7.1.2 Time Complexity.....	69
4.7.1.3 Asymptotic Notation (O)	70
4.7.2 Time Complexity of 1-IRS Dynamic Shortest Path Algorithms.....	70
4.8 Performance Comparison of 1-IRS Dynamic Algorithms and Dijkstra's Algorithm.....	71
4.8.1 Introduction.....	71
4.8.2 Performance Comparison of 1-IRS and Dijkstra's Algorithms on 2D Grids.....	72
4.8.3 Performance Comparison of 1-IRS and Dijkstra's Algorithms on Trees.....	76
4.8.4 Performance Comparison of 1-IRS and Dijkstra's Algorithms on Rings.....	79
4.9 Dynamic Shortest Path Algorithm for Complex Networks.....	82
4.9.1 Element Classes and Non-Classified Segments.....	82
4.9.2 General Procedure Description.....	84
4.9.3 Performance Considerations.....	89
4.10 2-Layer 1-IRS Dynamic Shortest Path Algorithm.....	91
4.10.1 Introduction.....	91
4.10.2 2-Layer 1-IRS Dynamic Shortest Path Algorithm.....	92

Table of Contents

4.10.3 An Example of 2-Layer 1-IRS Dynamic Shortest Path Algorithm.....	93
4.10 Summary.....	96
Chapter 5: Flow Control Algorithm	98
5.1 Introduction.....	98
5.2 Problem Description.....	98
5.2.1 Introduction.....	98
5.2.2 Preliminary Definitions.....	100
5.2.2.1 Nodes and Arcs.....	100
5.2.2.2 Traversal Time (TT)	101
5.2.2.3 Dynamic Capacity (DC)	101
5.2.2.4 Wide Doors and Narrow Doors (WD and ND)	102
5.2.2.5 Single ND Problem.....	103
5.2.2.6 Single WD Problem.....	104
5.2.2.6 Multiple-ND Problem.....	105
5.3 The Flow Control Algorithm for the Multiple-ND Problem.....	107
5.3.1 A Generalization for Multiple-ND Problem.....	107
5.3.2 The Multiple-ND Flow Control Algorithm.....	109
5.4 K-Limited Flow Control Algorithm.....	110
5.5 Summary.....	113
Chapter 6 Virtual Reality Augmented Evacuation System for Large	
Indoor Areas	114
6.1 Flow Control Application to Takashimaya Shopping Center.....	114
6.2 Virtual Reality Augmented Evacuation System Application.....	118
6.2.1 Introduction.....	118
6.2.2 System Components.....	118
6.2.3 N1-B3 Floor Plan and Its Network Conversion.....	121
6.2.4 System Demonstration.....	125
6.3 Summary.....	129
Chapter 7: Conclusions and Future Work	130
7.1 Conclusions.....	130
7.2 Academic Contributions of this Research.....	132
7.3 Future Research.....	134
References	135
Publications	146
Appendix: Algorithms in C++ Style Pseudo-Codes	147

List of Figures

List of Figures

Figure 2.1 Components of VR system.....	11
Figure 2.2 HMDs in VR.....	12
Figure 2.3 Sample Spatial Map.....	13
Figure 2.4 Exact Cell Decomposition.....	17
Figure 2.5 A Free Path in a Channel.....	17
Figure 2.6 Trapezoidal Decomposition.....	18
Figure 2.7 Decomposition and a Path for the Spatial Map in Figure 2.3.....	18
Figure 2.8 A Path Calculated with Visibility Graph.....	20
Figure 2.9 Reduced Visibility Graph.....	20
Figure 2.10 Reduced Visibility Graph for Spatial Map.....	21
Figure 2.11 A Voronoi Digram.....	22
Figure 2.12 A Voronoi Diagram for Spatial Map.....	23
Figure 2.13 (a) Example for Dijkstra's Algorithm.....	26
Figure 2.13 (b) Input for the Second Step.....	27
Figure 2.13 (c) Input for the Third Step.....	27
Figure 2.13 (d) Input for the Fourth Step.....	28
Figure 2.13 (e) Input for the Fifth Step.....	28
Figure 2.13 (f) Final Result.....	29
Figure 2.14 The Complete Graph for a Distributed System.....	31
Figure 2.15 An Example Complete Intersection Graph.....	33
Figure 2.16 An Example of 1-CIRS	35
Figure 3.1 Research Working Flow Chart.....	40
Figure 4.1 Spatial Map Diagram for a 2D Grid Structure.....	45
Figure 4.2 The Simplified Spatial Diagram for a 2D Grid.....	46
Figure 4.3 Spatial Map Diagram for a 3D Grid Structure.....	48
Figure 4.4 Simplified Spatial Map Diagram for a 3D Grid.....	49
Figure 4.5 Single Floor Layout for a Tree.....	51

List of Figures

Figure 4.6 Tree Structure of a Unit in a Block.....	51
Figure 4.7 A Ring Structure Example.....	52
Figure 4.8 1-LIRS for a 2D Grid.....	54
Figure 4.9 Dynamic S and T in 2D Grid.....	57
Figure 4.10 1-LIRS for a 3D Grid Structure.....	58
Figure 4.11 1-CIRS on A Tree T.....	61
Figure 4.12 A Tree cut be Dynamic S.....	62
Figure 4.13 A Tree cut be Dynamic T.....	62
Figure 4.14 Nodes and Arcs Labeling for Rings.....	64
Figure 4.15 A Ring Labeling Example.....	65
Figure 4.16 Static T and Dynamic S.....	66
Figure 4.17 Dynamic S and T.....	67
Figure 4.18 4 x 4 Grid and Source-Target Pair Generated by the Program.....	73
Figure 4.19 Relationship between Computing Time and Grid Size of Dijkstra's Algorithm on $n \times n$ 2D Grids.....	74
Figure 4.20 Relationship between Computing Time and Grid Size of 1-IRS Dynamic Algorithm on $n \times n$ 2D Grids.....	75
Figure 4.21 A Tree Generated by the Program.....	77
Figure 4.22 Relationship between Computing Time and Tree Size of Dijkstra's Algorithm on $n \times n$ Trees.....	79
Figure 4.23 Relationship between Computing Time and Tree Size of 1-IRS Algorithm on $n \times n$ Trees.....	79
Figure 4.24 Relationship between Computing Time and Ring Size of Dijkstra's Algorithm on n -node Rings.....	82
Figure 4.25 Relationship between Computing Time and Ring Size of 1-IRS Algorithm on n -node Rings.....	82
Figure 4.26 CSHG of a Complicated Structure.....	84
Figure 4.27 Star Form of an EC.....	85
Figure 4.28 2-Partite Form of an EC.....	86
Figure 4.29 The Flowchart for Multi-EC Algorithm.....	87

List of Figures

Figure 4.30 The PCCG for a Network.....	87
Figure 4.31 The CCG for a Network.....	88
Figure 4.32 Virtual Path for a Network.....	88
Figure 4.33 Real vs. Virtual Forms of an EC.....	90
Figure 4.34 2-layer 1-IRS Procedure.....	93
Figure 4.35 CSHG of the Example Network.....	94
Figure 4.36 2-Layer 1-IRS Labeling.....	95
Figure 5.1 Building Structure in Some Previous Research (Source: [84]).....	99
Figure 6.1 Takashimaya Floor Plan.....	114
Figure 6.2 Takashimaya Network.....	115
Figure 6.3 Input Dialog Box for Path Finding.....	115
Figure 6.4 Output Window for Path Finding.....	116
Figure 6.5 Input Dialog Box for Non-k-limited Flow Control.....	116
Figure 6.6 Output Message Box for Non-k-limited Flow Control.....	117
Figure 6.7 Output Window for Non-k-limited Flow Control.....	117
Figure 6.8 System Equipment	119
Figure 6.9 AR System.....	119
Figure 6.10 Floor Layout of N1-B3.....	122
Figure 6.11 Network of N1-B3.....	122
Figure 6.12 Region Division of N1-B3.....	123
Figure 6.13 the Cartesian System in OpenGL.....	125
Figure 6.14 Input Dialog for Testbed System.....	126
Figure 6.15 System Output Window.....	127
Figure 6.16 System Output Window with White Background.....	128
Figure 6.17 Bird's Eye View.....	128

List of Tables

List of Tables

Table 4.1 Experiment Data of Dijkstra’s Algorithm on n x n 2D Grids.....74

Table 4.2 Experiment Data of 1-IRS Dynamic Algorithm on n x n 2D Grids.....75

Table 4.3 Experiment Data of Dijkstra’s Algorithm on n x n trees.....78

Table 4.4 Experiment Data of 1-IRS Static Algorithm on n x n trees.....78

Table 4.5 Experiment Data of Dijkstra’s Algorithm on n-node Rings.....81

Table 4.6 Experiment Data of 1-IRS Algorithm on n-node Rings.....81

Table 6.1 Hardware List.....120

Table 6.2 Regions Mapping of N1-B3.....124

List of Acronyms and Abbreviations

List of Acronyms and Abbreviations

AR	Augmented Reality
BN	Boundary Node
CCG	Complete Connectivity Graph
CIG	Complete Intersection Graph
CIRS	Circular Interval Routing Scheme
CS	Classified Segment
CSHG	Classified Segment Hidden Graph
DC	Dynamic Capacity
DRM	Dead-Reckoning Module
EC	Element Class
ECID	Element Class Identity
EG	Evacuation Group
EPE	Ekahau Positioning Engine
GN	Gating Node
GPS	Global Positioning System
HMD	Head Mounted Display
IRS	Interval Routing Scheme
LIRS	Linear Interval Routing Scheme
NCS	Non-Classified Segment
ND	Narrow Door
ODS	Optimal Door Subset
OGN	Optimal Gating Node
PCCG	Primary Complete Connectivity Graph
PDA	Personal Digital Assitant
RSI	Received Signal Intensity
SO	Staircase Occupancy
TT	Traversal Time

List of Acronyms and Abbreviations

TTT	Total Traversal Time
VR	Virtual Reality
WD	Wide Door

Chapter 1 Introduction

1.1 Background

This research is to find fast real-time algorithms on optimal path finding and flow control for Euclidean networks, especially networks converted from large indoor areas. As the target environments would be present to the people in the form of 3D virtual reality (VR) in order to serve as an aid for rescue tasks in case of emergency within large indoor areas, a systematic analysis on building topologies and network abstractions and an overview on VR hardware, the efficiency of software and the real-time characteristics of the evacuation tasks are required. The workability of traditional optimal path and flow control algorithms must be reviewed under the new criteria set by the properties of the VR augmented emergency system.

1.1.1 Preliminaries of VR

In a Virtual Reality (VR) environment, all the objects appearing on the display device are not real. A person can interact with the virtual objects through specific input devices. A typical VR system is an F1 Car Race computer game. The 3D objects, such as racing cars, roads, grass and sky appearing on the monitor are not physically real but created with some development software. A player drives his own virtual racing car to compete with other virtual cars in a virtual racing court. The player controls his car with the keyboard, or mouse or joystick connected to the computer. In this case, the VR system has no purpose to make any change to the physical world. The environment is not physically real and neither is the whole race. In addition to its entertaining function in forms of computer games, 3D VR has been widely used in the construction industry. At the designing stage of a building, the effect graphs produced with some commercial 3D VR software like 3D Max present its appearance and put it into a virtual environment in order to give a vivid effect to the people. In project

Chapter1 Introduction

management, 4D VR works as a powerful interactive tool for project managers. Time is added as another dimension in addition to the geometric 3D. Project managers can choose the time and the system can show the constructed building components in a 3D form.

Sometimes we need to work in a real environment to fulfill a task. But with some virtual information added to some convenient display devices, the task can be done much more efficiently. The virtual information therefore is called augmenting information. And the real environment with augmenting information is called Augmented Reality (AR). AR has a long history of use in industry assistant systems. A well-known instance is the technical assistant AR system. A technician wearing a Head Mounted Display (HMD) unit can easily handle unfamiliar equipment with the information given on HMD, and his hands are working on real equipments in the meanwhile.

A VR or AR system is usually designed for specific tasks. As VR presents a virtual environment and no physical objects exist, it is usually for pure presentation or recreation purposes. But AR always serves physical purposes. That is why the evacuation aid system, into which we are going to add routing and flow control algorithms, is essentially an AR system.

In the case of earthquake, toxic gas explosion or fire, smoke or darkness due to the power outage, may cause the real world invisible to people. So the emergency evacuation aid system should include the 3D environment model, which is a fundamental part of a VR system. And in this emergency evacuation aid system, path finding and/or routing functions could be necessary.

Our evacuation system has integrated multiple advanced techniques in location tracking, orientation tracking, and ubiquitous computation. The presentation of 3D view and the background computation for path finding and flow control are conducted

Chapter1 Introduction

in a real time mode. This is supposed to be an effective AR assistant application for user's decision in context of emergency especially in an unfamiliar environment.

1.1.2 Large Indoor Area Evacuation Aid System Augmented by Virtual Reality

There has been some software studying the evacuation profile of a simple multi-storey building. They accept simple input information and simulate the whole evacuation process in a time ranging from several minutes to a couple of hours. The system we have integrated is significantly different from most of the existing evacuation software systems. The differences are as follows:

- i) The system is going to be implemented in a VR environment, whose appearance follows the real world. All the graphic, symbolic or numeric information presented to the users are displayed on HMD. All path information is presented as a series of connected straight lines of different colors in a 3D VR environment. This intuitive presentation solution is more effective than those given on traditional monitors for people in real emergency scenes.
- ii) Some input of the system like location and orientation information comes from the trackers equipped on the mobile users so that the scenes on HMD can be updated in real time. The automatic location collecting function significantly decreases input work.
- iii) The system updates scenes on HMD in real time according to the location and orientation of mobile users. The system can start to do the path calculation at anytime as the mobile user requests. It is required that the path calculation be completed in a time as short as possible, because it is a real time tool to aid people to escape from a dangerous area as fast as possible.

Chapter1 Introduction

- iv) The system deals with a large indoor area. This area may include a complicated multi-storey building, a large indoor area with multiple obstacles, or a combination of several structures above. Generally the input for a path finding algorithm is a network. The more complicated the topology of the indoor area, the larger the input network data structure to the path finding algorithm. So the method for the conversion of the indoor space to an abstract network must ensure the network size is suitable for real time calculation.
- v) Users of the system are supposed to be rescue team members. When they are evacuating a large group of people, how to assign evacuees to different exits to minimize the evacuation time is a big challenge. With a fast flow control algorithm provided, the system is able to give effective flow control suggestions in a short computing time. Existing evacuation software's flow control algorithms are not effective enough for our system in terms of their computing speed.

The features of our emergency evacuation system raised problems in several aspects, which are discussed below.

1.2 Problem Statement

How to find a new effective shortest path calculation method for common indoor topologies is one of the main topics discussed in this report. But detailed knowledge about the context, in which the path finding algorithm is going to work, can help in defining the requirements and limitations of the algorithm.

The VR engine of the system would consume a large amount of computation resource of the computer, such as RAM and CPU time. Other functions of the system such as path finding or flow assignment computation will compete with the VR engine for these resources. So it is required that the algorithms have low space and time

Chapter1 Introduction

complexity. Our Large Indoor Area Evacuation Aid System augmented by VR presents problems from 4 aspects:

- i) How to convert a large indoor area into an abstract network structure, which can facilitate the computation of the path. The size and the topology complexity of the network may seriously affect the calculation speed.
- ii) Develop a shortest path finding algorithm, which has a fast calculating speed and the calculation time is affordable in case of emergency in real time. This algorithm is supposed to run faster for the topology of large indoor areas than Dijkstra's shortest path algorithm.
- iii) Find an efficient flow control algorithm. This algorithm can work well in interactive VR environments in real time.
- iv) The accuracy level of the calculation should be suitable for those of the VR location and orientation trackers. A higher accuracy level may need longer calculation time and higher space complexity. How to find a good trade-off point between the level of accuracy and the computing time with limits presented by a VR system is an art rather than just an engineering problem.

With the problems stated above, the research objectives of this research are listed in the following sections.

Chapter1 Introduction

1.3 Research Objectives

The aim of this research is to find systematic methods for optimal path finding and flow control. The research objectives can be identified as follows:

1. To analyze the structures of common indoor areas, such as campus buildings, commercial buildings or apartment buildings. Find a method to convert the complicated VR environment into a network suitable for the path finding or flow computations.
2. To develop a fast optimal path finding algorithm suitable for real time systems. Its computation time should be affordable in the situations of emergency evacuation.
3. To develop a fast flow control algorithm suitable for real time systems. The computing time should be affordable in situation of emergency evacuation.
4. To test the new real time flow control function in a real time VR system, and evaluate its performance in terms of computing time.

1.4 Organization of the Report

This research describes the development of a shortest path finding algorithm and a flow control method, which are for the topologies of large indoor areas presented in the form of VR for real time emergency evacuation tasks. The organization of this report is as follows:

Chapter 1 presents an introduction to this research.

Chapter 2 reviews previous works in VR and AR, previous evacuation software systems, previous research in space decomposition, graph theory, shortest path

Chapter1 Introduction

algorithms and some optimal routing techniques, and previous research in flow control.

Chapter 3 describes the methodology used in this research

Chapter 4 discusses previous path finding algorithms in detail. How some large indoor area layouts are abstracted to be certain network structures, which are the input to path finding algorithms, is discussed as well. Dynamic shortest path finding methods with interval routing techniques for these structures are described.

Chapter 5 describes the optimal flow control problems. A fast linear optimal flow control algorithm is developed and described in detail.

Chapter 6 demonstrates a sample program first using a network based on the topology of level 1 of Takashimaya indoor shopping centre. In this program, the flow control algorithm is implemented without any location or orientation trackers integrated. All input is given manually.

Chapter 6 gives an overview of a real-time testbed emergency evacuation system, which is the platform for all the real time algorithms implemented. The detailed information on hardware integration and the algorithm implementation are given in this chapter. This testbed system is for a project financially supported by Nanyang Technological University.

Chapter 7 summarizes current findings and outlines possible future research work for similar emergency evacuation systems.

Chapter 2 Literature Review

2.1 Introduction

To develop efficient algorithms to solve practical problems in a real time system, the first step is to find what limitations of the system raises by its software and hardware. So the first part of this chapter will give a brief review on previous virtual reality (VR) works.

All network algorithms need an input network to get output, but a 3D VR environment is not a network originally. So how to abstract a 3D VR environment to be a network, whose size and data structure are suitable to the objectives of the algorithms for a real time system, is the second problem we will deal with in our algorithm development. Some existing methods to convert real space to network structures are reviewed in the second part of this chapter.

Then relevant previous research on optimal route finding and flow control methods are reviewed. When existing optimal route finding and flow control methods are applied directly to our system, some problems will rise. These problems are identified and considered as the starting point of the algorithm development stage.

2.2 Previous Work in VR and AR

2.2.1 Introduction

Augmented reality (AR) has developed solution for many complex practical problems. During the past 20 years, new technologies, such as GPS, indoor location tracking system and wearable equipments, make AR more and more effective in its application to real-world domain. In technical assistant system, AR implemented by see-through

Chapter2 Literature Review

eye-pieces has become a very popular tool. It is the same in a lot of other fields like medical therapy, mechanical control, tourism and exhibition, project process control and military information systems.

The objective of this research is to explore an AR (implemented as VR due to the task character) enhanced evacuation aid system for large indoor environment. This system has integrated advanced techniques in location tracking, orientation tracking, and ubiquitous computation. Therefore, interface presentation on HMD (Head Mounted Display) and the background computation for shortest path finding and flow control are conducted in a real time mode.

2.2.2 Indoor Position Tracking

Many researchers have addressed the development of outdoor location-aware mobile and wearable systems. Some have relied on modifying the environment explored. For example, in some research, building information is labeled with bar codes to provide location-specific information on a hand-held computer equipped with a bar code scanner [1]. Others have combined Global Positioning System (GPS) and orientation trackers to produce map-based contextual displays, to provide audio navigation assistance to blind users, or to annotate the world with overlaid textual labels or multimedia information [2-7]. These projects presage the goal of anchoring a world-wide spatial hypertext of information to physical locations and objects.

Indoor augmented reality systems have to turn to different tracking techniques due to the incapability and inaccuracy of GPS in indoor environment. Many approaches to indoor position tracking require that the user's environment be equipped with sensors, beacons, or visual fiducials [8-12]. Tethered position and orientation tracking systems have attained high accuracy for up to room-sized areas using magnetic, ultrasonic, and optical technologies, including dense arrays of ceiling-mounted optical beacons [13, 14]. Alternatively, sparsely placed infrared beacons can support tether-less

Chapter2 Literature Review

position-only tracking over and entire building at much lower accuracy [10, 11]

But for indoor wide area tracking, ultrasonic tracking and GPS are not applicable, both of which require no obstruction between the sensor and the transmitter. An alternative tracking method was tested, which used a Point Research PintMan Dead-Reckoning Module (DRM) integrated with a pedometer [15].

The location trackers suitable for large indoor areas like Ekahau Positioning Engine™ (EPE) have appeared in the market [16]. It is a software-based location technique developed by Ekahau, Inc. Unlike the competing positioning technologies, Ekahau does not apply propagation or triangulation methods that suffer from radio wave multipathing, scattering, and attenuation effects. Instead, the Ekahau Site Calibration™ method is used for collecting radio network sample points from different site locations. Each sample point contains received signal intensity (RSI) and the related map coordinates, stored in an area-specific Positioning Model for accurate tracking [16]. NTU campus has been covered by an IEEE 802.11 Wi-Fi wireless LAN, which is the only location tracking hardware required for EPE to provide location information; with an average accuracy of 1 meter, EPE can provides a very convenient and cost-effective tracking method. And the system can be tested in our campus directly. That's the reason Ekahau Wi-Fi location tracker is used in our system.

2.2.3 System Components of General VR Systems

The components of a classic virtual reality system [20] are shown in Figure 2.1.

Figure 2.1 Components of VR system

An evacuation system is designed to augment the real world instead of block real world information, so it should be called augmented reality system. But its implementation is essentially a virtual reality system, so the model in Figure 2.1 can be directly used to describe the VR part of a large indoor area evacuation system.

i) I/O Devices

The I/O devices for mobile users are Head Mounted Displays (HMD) shown in Figure 2.2. Each mobile user holds a mobile computer or Personal Digital Assitant (PDA) with a HMD plugged in. For a single mobile user (without indoor users or any other mobile users), the computer or PDA will hold all software including 3D virtual reality model, VR engine, object information database and task components. Our testbed system is in this simple form.

Chapter2 Literature Review

ii) VR Engine

VR engine is a key component of any VR system, which reads its input devices, accesses task-dependent databases, performs the required real-time computations to update the state of the virtual world, and feeds the results to the output displays.

Figure 2.2 HMDs in VR

The mobile user is equipped with location and orientation trackers, from which VR engine reads user's location and orientation and updates the virtual world displayed on the mobile user's HMD.

Typical orientation trackers in the markets like InertiaCube2 are mounted over user's head [17]. With location and orientation trackers, VR engine can get X-Y coordinate information and user's yaw, pitch and roll angles. Then VR engine will do the

Chapter2 Literature Review

computation based on the 3D model of the environment and update mobile user's HMD. Then the VR engine will trace and present relevant information of the updated environment from a database.

iii) Software and Database

The software implementation for a VR augmented evacuation system is usually divided into 3 stages. The first stage is environment modeling stage. As the working environment of the system will be normalized into nodes and links, the environment should be modeled as a collection of objects. So a 3D or 2D object oriented graphic developing environment is needed. There are a variety of available commercial VR development software like WorldToolKit can do this work very efficiently [8]. As the 3D environment model of our testbed system is relatively simple, it has been built with OpenGL library in VC++, which is the basic language implementing all the algorithms in the system.

The 3D VR environment will be built in the form of a collection of spatial maps, which will accurately model the building geometry (walls, doors, passageways etc.) [15]. A sample spatial map is shown in Figure 2.3

Figure 2.3 Sample Spatial Map

Chapter2 Literature Review

Information about all objects in the spatial map is stored in the form of relational database. Real-time modification is allowed. When the mobile user makes a request for the information about an object, VR engine can search the database and transfer back correct information. Although this part has not been included into our system due to time and cost limitations, it is a very important part for a mature VR system.

iv) Tasks

The users of our evacuation system are supposed to be rescue teams, whose main tasks are to find people in dangerous areas and take them to safe areas in an efficient way; so shortest path finding and flow control functions have been integrated into the system. With these functions, mobile rescue team members can find people in danger and lead them to safe areas in a short time.

v) User/Trainee

Some training should be conducted to let users be familiar with the interactive VR interface. As time latency and register errors are unavoidable due to trackers' inadequate accuracy level and computation delays, the users must take some time to get comfortable with the VR environment and get familiar with how to activate path finding and flow control calculation by the pointing equipments.

2.2.4 Problems Raised by VR Systems

As the VR engine processes the real time location and orientation data from trackers, and updates the view on HMD dynamically, the computing resources like CPU and RAM would be highly occupied. Additional computation tasks like path finding or flow control may affect the performance of the VR engine, when they are competing for the computing resources. When the input network for route finding or flow control computation is very large in terms of node or arc number, the effect may be very

Chapter2 Literature Review

serious. On the other hand with shortage in computing resource, the computing speed of additional computation tasks may become so slow that it can not meet the time limit the evacuation tasks need.

With respects to above problems raised by the VR system and by the task characters, fast optimal path finding and flow control algorithms for common large indoor topologies are going to be developed. The computation complexity of the new algorithms should be low enough that it can be easily integrated with the VR system and can enable the VR system to fulfill the evacuation tasks in time.

2.3 Free Space Conversion

2.3.1 Introduction

Most large indoor areas consist of one or more than one building connected by some passageways or doors. A path between any two points in the large indoor area should be within the free space in this area. The free space refers to all the space within this area, which allows people to walk through. All the passageways, staircases and any empty space confined by doors, internal and external walls of the building may be deemed as free space. As for our path finding problem, passageways and staircases are our concerns; and it is assumed that our algorithm only deals with paths within passageway and staircase areas. In robot motion planning, path finding is a traditional problem and this free space is called work space or configuration space for the movement of a robot. In later discussion, the free space is symbolized by C_{free} .

As the platform for any path finding computation, the free space in real large indoor areas or VR interface has to be converted to certain data structure, which can be recognized by the computer. In robot motion planning, free space is usually converted to a network, which is a convenient data structure for any further path finding computation. And with the development of modern computation techniques, network

Chapter2 Literature Review

analysis has become a power tool for a variety of problems. For our research, network analysis is used and the free space of a large indoor area will be converted into a network G , with V as its node set and E edge set. Therefore, the network is represented by the sign $G(V, E)$.

For the same free space, there are a number of methods to do the network conversion. And the resulting networks are different due to the methods. To choose a method suitable for our path finding problem, literatures on free space conversion are reviewed as follows.

2.3.2 Exact Cell Decomposition

One approach to do the network conversion is exact cell decomposition. The principle of this approach is to first decompose the free space C_{free} into a collection of non-overlapping regions, called cells, whose union is exactly C_{free} (or its closure). Next, the connectivity graph which represents the adjacency relation among the cells is constructed and searched. The outcome of the path finding computation based on this network conversion is actually a sequence of cells, called a channel, connecting the cell containing the initial configuration to the cell containing the goal configuration. A path is finally extracted from this sequence.

The small cells, which a decomposition of a free space C_{free} gets, can take different shapes. Triangular and trapezoidal decomposition are usually used to solve robot motion planning in a free space with obstacles presented. Some triangular and trapezoidal decomposition methods have been developed [21-25]. Some other decomposition methods may not need all cells in the same shape [26, 27], one of whose example is shown in Figure 2.4. The free space is decomposed into a collection of convex polygonal cells whose interiors don't intersect. Each cell is labeled by a distinct integer and the associated connectivity graph is displayed with each node placed in the corresponding cell.

Chapter2 Literature Review

Figure 2.4 Exact Cell Decomposition

q_{init} and q_{goal} is the source node and target node respectively and black areas are obstacles in Figure 2.4. Once a channel is constructed, a path in free space can easily be generated as the polygonal line joining q_{init} and q_{goal} through the midpoints of the portions of boundary shared by every two successive cells in the channel. The path is shown in Figure 2.5.

Figure 2.5 A Free Path in a Channel

With the trapezoidal decomposition method, the same free space is decomposed as in Figure 2.6 [25]. A vertical line parallel to the y-axis is swept across C_{free} from left to right. At every vertex X of any obstacle, a maximum of two vertical line segments are erected. They connect X to the edges of the obstacles immediately above and below

X.

Figure 2.6 Trapezoidal Decomposition

With this method, the special map in Figure 2.3 can be decomposed as in Figure 2.7.

Figure 2.7 Decomposition and a Path for the Spatial Map in Figure 2.3

The decomposition for the free space and a path between two arbitrary points are shown in the Figure 2.7. We can see that the path for the free space in the spatial map does not suit very well to people’s walking habit. In the passageway of a building, person usually takes a path paralleling to the walls on two sides. But the in Figure 2.7, the path does not follow people’s behavior very well.

2.3.3 Road Map Methods

The definition of road map methods was given, whose general idea is to capture the connectivity of the robot's free space in the form of a network of one-dimensional curves [27]. One-dimensional curves are generally straight lines. There are a variety of methods to construct the roadmap. Methods based on different principles have been proposed, producing various sorts of roadmaps, called "visibility graphs", "Voronoi diagrams", etc.

2.3.3.1 Visibility Graphs

Visibility graph method essentially applies to two-dimensional free spaces with polygon obstacles presented. The road map is a graph called a "visibility graph" which is constructed by connecting every pair of vertices in free space C_{free} 's boundary by a straight segment, if this segment does not traverse the interior of any obstacle. The boundary B of C_{free} is the union of the outer boundary of the free space and the boundaries of all obstacles within the free space.

This method is one of the earliest path planning methods and it has been widely used to implement path planning for mobile robots. The visibility graph for a free space and two points within it is showed in Figure 2.8. With a set of disjoint polygonal obstacles and two points within the free space, a path between these two points can be calculated; and the resulting path is a sequence of straight arcs. Two points are the starting and ending nodes of this arc sequence.

The visibility graph was introduced for planning robot motions [28]. An $O(n^2 \log n)$ algorithm [29] is presented with n as the total number of the nodes in the network in the visibility graph. Some other fast optimal algorithms accept a set of disjoint obstacles, a starting point and an ending point as the input [30-32]. They construct the visibility graph in real time and calculate the shortest path in the form of a sequence

Chapter2 Literature Review

of straight arcs.

In some cases, one does not need all the visibility edges. One only needs the visibility edges that define a common tangent. An algorithm computes this reduced visibility graph in time $O(n + c^2 \log n)$, where c is the number of obstacles [33]. The reduced visibility graph for the same free space as in Figure 2.8 is shown in Figure 2.9.

Figure 2.8 A Path Calculated with Visibility Graph

Figure 2.9 Reduced Visibility Graph

Chapter2 Literature Review

The arcs in Figure 2.9 are reduced and the final path is the same as in Figure 2.8. The reduced visibility graph of the spatial map in Figure 2.3 is shown in Figure 2.10.

Figure 2.10 Reduced Visibility Graph for Spatial Map

For the starting and ending point in Figure 2.10, there are 16 arcs and 10 nodes in the network.

In graph computing, the computation complexity of an algorithm in terms of mathematic sign O is usually proportional to the arc and node numbers in the network. In the process of network conversion, a network with more arcs and/or more nodes may present more accurate information about the topology of the free space; and therefore the resulting path may be more accurate in terms of its Euclidean length (Euclidean length is the real geometric length of an arc or a path). But more nodes and arcs mean more computation time and in real time system, a trade-off point have to be found between accuracy and efficiency.

Another roadmap method named Voronoi diagrams significantly decreases the network size and the computation result is affordable in terms of its accuracy.

2.3.3.2 Voronoi Diagram

Voronoi diagrams are attributed to a series of research in [34-42]. The Voronoi diagram of C_{free} is the set of points whose minimal distance to the boundaries (including inner and outer boundaries) of the free space is achieved with more than one point in the boundaries. The Voronoi diagram may consist of straight segments and parabolic arc. In Figure 2.11, a Voronoi diagram is constructed for a bounded polygonal free space.

Figure 2.11 A Voronoi Diagram

In Figure 2.11, a straight segment is the set of configurations that are closest to the same pair of edges or the same pair of vertices. For example, the straight segment S_1 is closest to edges E_3 and E_7 . A parabolic arc is the set of configurations that are closest to the same pair consisting of an edge and a vertex. For example, the arc S_2 is closest to edge E_8 and vertex X_5 .

Chapter2 Literature Review

A Voronoi diagram of the free space in Figure 2.3 is shown in Figure 2.12. The Voronoi diagram only has 10 nodes and 11 arcs. The network size is significantly reduced, but the accuracy of the paths calculated with Voronoi diagram is affordable for most of real time tasks.

If the four arcs, which are not parallel to any boundary of the free space, are deleted, the resulting network only has 6 nodes and 7 arcs. If the starting ending points are given in real time, the calculation of the path will be much faster due to this small modification of the Voronoi diagram. And the accuracy level of the computing outcome is still affordable for our VR evacuation tasks. The size of the networks got with visibility graph or exact cell decomposition methods are generally much bigger than Voronoi diagram. In real time system, original or modified Voronoi diagram is a better choice due to its fast computation speed.

Figure 2.12 A Voronoi Diagram for Spatial Map

Additionally, the modified Voronoi diagram (we call it *spatial map diagram* later on) for the spatial map of the indoor environment gives a path, within which, most of the arcs is a straight line parallel to the walls on two sides. This suits people's walking habit very well. The output paths from exact cell decomposition methods and visibility graph methods do not have this characteristic. It is another advantage of spatial map diagram.

Chapter2 Literature Review

Another advantage of spatial map diagram is that the updating of the network is much easier than exact cell decomposition methods or visibility graph. When emergency taking place, some passage way may be damaged so serious that it can not be passable. Then some updating on the network structure is needed. If the network conversion is based on visibility graph method or exact cell decomposition, the change to one segment passage way often causes the changes to multiple arcs of the overall network. But with spatial map diagram, one segment of passage way corresponds to one arc. So the change on one segment of passage way only causes the change on one arc in the network. This advantage may make the real time system very efficient in the situations where significant changes happen to the real world environment.

When it is decided that spatial map diagram is going to be used as the input to the routing algorithm we are going to develop in this research, we have to use it to construct networks for a variety of inside-building structures. With the analysis on these structures, new algorithm is described in Chapter 4.

After the method to convert large inside-building structure is chosen, we can move on to review previous researches on optimal routing and shortest path finding. Shortest path finding has been well studied in a variety of research areas such as operation, transportation, electronic circuit and robot motion planning. One algorithm due to Dijkstra has been accepted as the most efficient shortest path algorithm to general network topologies [44]. With certain topologies, especially high regular topologies, other algorithms are developed in order to have lower computation complexity and therefore higher computation speed than Dijkstra's algorithm. Let's start to review these researches.

Chapter2 Literature Review

2.4 Dijkstra's Shortest Path Algorithm

The idea behind Dijkstra's shortest path algorithm [57, 58] for a weighted network is straightforward. And as for its low computation complexity, it is used as the basic algorithm for path finding techniques. Most of graph theory problems and correspondent researches, which involves path finding, refer to Dijkstra's algorithm. We can only list a few of them because of the huge number of this kind of research. Some examples can be found in [47-56]. For some networks of special topologies, other algorithms, not involving Dijkstra's algorithm, are used. But they are not that popular due to its narrow application scope.

The essence of Dijkstra's shortest path algorithm is that it fans out from the origin, successively identifying the shortest path to each nodes of the network in the ascending order of their shortest distances from the origin, thereby solving the problem when the destination node is reached.

Here we use a popular way to describe this algorithm as follows [58]:

(1) Objective of nth iteration: Find the nth nearest node to the origin (to be repeated for $n = 1, 2 \dots$ until the nth nearest node is the destination).

(2) Input for nth iteration: $n - 1$ nearest nodes to the origin (solved for at the previous iterations), including their shortest path and distance from the origin. (These nodes, plus the origin, will be called solved nodes; the others are unsolved nodes.)

(3) Candidates for nth nearest node: Each solved node that is directly connected by a link to one or more unsolved nodes provides one candidate – the unsolved node with the shortest connecting link. (Ties provide additional candidates.)

Chapter2 Literature Review

(4) *Calculation of nth nearest node*: For each such solved node and its candidate, add the distance between them and the distance of the shortest path from the origin to this solved node. The candidate with the smallest such total distance is the nth nearest node (ties provide additional solved nodes), and its shortest path is the one generating this distance.

To illustrate this algorithm, an example is given in Figure 2.13. A network is given with O as the original point and T as the target point. The number on each arc is the weight of the arc. (The weight is the length of the arc in this case.) To find the shortest path from O to T, we start from O.

In the first step, scan O’s adjacency list and find that among three nodes A, B and C, A is the nearest node to O. So A is the candidate of O. As O is the root node of the algorithm, no other candidate node appears at this stage. So A is labeled solved. The input for the second step is shown in Figure 2.13 (b).

Figure 2.13 (a) Example for Dijkstra’s Algorithm

In the second step, O’s candidate is C with distance 4 and A’s candidate is B with distance to O as $2+2=4$. It is a tie between two candidates, so label both of them as solved. Then the input for the third step is as in Figure 2.13 (c)

Figure 2.13 (b) Input for the Second Step

Figure 2.13 (c) Input for the Third Step

In the third step, A's candidate is D with distance $2+7=9$; B's candidate is E with distance $2+2+3=7$; C's candidate is E with $4+4=8$. So add E. And remember E's predecessor is B not C. Then the input for the fourth step is as in Figure 2.13 (d)

Figure 2.13 (d) Input for the Fourth Step

In the fourth step, A’s candidate is D with distance $2+7=9$; B’s candidate is D with distance $2+2+4=8$; and E’s candidate is D with distance $2+2+3+1=8$. So label D as solved and choose B or E as its predecessor. If E is chosen, the input for the fifth step is as in Figure 2.13 (e).

Figure 2.13 (e) Input for the Fifth Step

In the fifth step, D’s candidate is T with $2+2+3+1+5=13$ as its distance; E’s candidate is T with $2+2+3+7=14$ as its distance. No other solved nodes have candidates. So T is

labeled solved and the shortest path from O to T is OABEDT with the length 13. It is shown in Figure 2.13 (f).

Figure 2.13 (f) Final Result

2.5 Shortest Path Computation in Distributed Systems

A distributed system is composed of a set of autonomous subsystems and a central computing server. The subsystems may overlap at some places called intersections. To divide a large network into several intersected subnetworks may be due to specific reasons. In one research, a large network is divided into subnetworks due to the shortage of information for the central computing server [59]. Each subnetwork is an autonomous subsystem with a computer, which can give route information of the nodes within the subsystem. Subsystems are connected by the intersection nodes, which may belong to more than one subsystem and therefore can serve as a gate from one subsystem to another. The central computing server is to provide route information that crosses subsystems. If the starting and ending nodes exist in different subsystems, the central computing server is to calculate the globally shortest path with the information on intersection nodes provided by all subsystems. The input network for the central computing server is constructed according to the information collected from all subsystems. This input network is called *complete intersection graph* (CIG).

Chapter2 Literature Review

The above algorithm is developed for the some specific problems [60, 61]. The reason to divide the whole network to be several intersected subsystems is due to the unavailability of the overall network information for the central server. In the algorithm we have developed in Chapter 4 requires the large network to be divided into multiple connected small networks for different reasons. In Chapter 4, this reason is discussed in detail.

Some algorithm use a form of intersection graph [62, 63], which provides less information than the complete intersection graph, to apply Azevedo's heuristic method for calculation and ranking a set of top k ($k > 1$) best solutions [52]. The path computed by this algorithm is not guaranteed to be globally optimal because the intersection graph on which it is calculated does not include all the necessary information.

As we are concerned about globally optimal path in this research, a close look at the CIG algorithm is preferred [59]. The computation process of an example large network, which is shown in Figure 2.14, is analyzed below.

Figure 2.14 The Complete Graph for a Distributed System

The complete large network in Figure 2.14 is divided into four subsystems. Class 1 and Class 2 intersect at node a. Class1 and Class 4 intersect at node e. Class 2 and Class 3 intersect at nodes c and b. Class 2 and Class 4 intersect at node d. Class 3 and Class 4 intersect at node f. Note that there are two arcs connecting c and b. These two arcs belong to Class 2 and Class 3 respectively, and have different lengths. In the construction of the complete intersection graph, only the arc with minimum length is retained in this case.

If the starting and the ending nodes are assigned and are in different subsystems, the computing can be started. The whole process of the CIG algorithm can be divided into 3 stages; they are:

Stage1 subsystem calculation: In the class where the starting node exists, compute the route from the starting node to all nodes, where the class intersects with other classes.

Chapter2 Literature Review

Within each of other classes, routes are calculated between intersection nodes. Notice that all calculations within a subsystem use Dijkstra's algorithm.

Stage2 construction of complete intersection graph: In this stage, with all the route information provided by the subsystems, a complete intersection graph is constructed.

Stage3 global shortest path computation: The complete intersection graph is input to the central computing server to calculate the global shortest path with Dijkstra's algorithm.

The complete intersection graph for Figure 2.14 for with the starting and ending nodes designated is shown in Figure 2.15.

Original large network can be directly input to a computer for Dijkstra's algorithm without division, whose computation time on one computer is denoted by T_1 ; and it is supposed to be faster than the above algorithm if the 3 stages are all implemented on only one computer.

Let T_1 denote the time of the computation in *Stage1* if all subsystems' computation is implemented on only one computer. T_2 denotes the time needed to construct the complete intersection graph. And T_3 denotes the time needed in *Stage3*.

Figure 2.15 An Example Complete Intersection Graph

As *Stage1* and *Stage3* are both based on Dijkstra's algorithm, and the network areas covered by *Stage1* and *Stage3* are overlapped very little. So the network size obtained by adding all the network areas covered by *Stage1* and *Stage3* is very close to the size of the whole large network. As Dijkstra's algorithm's computation time is roughly proportional to the network size, $T_1 + T_3$ should be very close to T_t .

As the total computing time for the CIG algorithm is $T_1+T_2+T_3$ and T_1+T_3 is close to T_t , so its computation time implemented on one computer is longer than pure Dijkstra's algorithm implemented directly to the whole large network. The time difference is roughly due to T_2 , i.e. construction of the complete intersection graph lowers the computation efficiency of the CIG algorithm.

If within each subnetwork, some fast algorithm with much higher computing speed than Dijkstra's algorithm is applied instead, the computing time saved on T_1 may counteract the negative effect caused by T_2 . In some cases, especially to some special topologies of the network, T_1 may be proportionally much longer than T_2 and T_3 . In

Chapter2 Literature Review

this case, the application of some fast algorithm to the subsystems in stead of Dijkstra's algorithm may significantly decrease the total time of the computation that the new computation time may be even shorter than T_1 .

As for the network we are dealing with, its topological characteristics are analyzed in Chapter 4. With some manipulation, a large inside-building network can be converted to be a collection of connected highly regular structures. Within each highly regular structure, a fast path finding method called **Interval Routing Scheme** can be applied. If each of this structure is treated as a subnetwork similar to the one in CIG and in stage1 Dijkstra's algorithm is replaced with interval routing scheme, in some cases, the total computing time may be significantly shorter than T_1 . In next section, literatures in interval routing scheme are reviewed; and some preliminary concepts in interval routing techniques are introduced.

2.6 Interval Routing Scheme

Interval Routing is a way of implementing routing schemes on arbitrary networks. A routing scheme is a strategy that assigns to every source-target pair the path that a message from the source to the target should take. The concept of routing scheme was originally developed for computer network. For example, one possible routing scheme is to store a complete routing table in each of the n vertices of the network, specifying for each target the next edge in some shortest path over which the message must be forwarded. But with the number of the nodes increases, the size of the routing tables stored at each node would increase as well.

Another way of implementing routing schemes, called interval routing, has been discussed in a number of research articles, which has been summarized in [65]. Actually interval routing scheme is based on representing the routing table stored at each node in a compact manner. In this method, each node is assigned a distinct label from the set $\{1, 2 \dots n\}$. Arcs are bi-directional and are labeled with one or several

Chapter2 Literature Review

subintervals of the interval $[1 \dots n]$ so that for any node v the intervals associated with outgoing edges from v are pair-wise disjoint and their union covers $[1 \dots n]$. When a message with destination v arrives at node u , the message is forwarded on the unique outgoing edge labeled with an interval containing the label of v .

In most cases, $[1 \dots n]$ is the cyclic interval, i.e., all subintervals are understood to wrap around. Such a scheme is called a circular interval routing scheme (CIRS). Variants of the scheme include linear interval routing schemes (LIRS), in which $[1 \dots n]$ is viewed as a linear interval; k -interval routing schemes, in which edges can be labeled with at most k -intervals (k -IRS or k -LIRS). An interval routing scheme for which all messages are routed along shortest paths is called an optimal scheme.

An example of a circular interval routing scheme is showed in Figure 2.16.

Figure 2.16 An Example of 1-CIRS

In Figure 2.16, nodes are first labeled with numbers from 1 to 7. Each arc is given an interval for each direction. If we are trying to find the path from node 5 to node 2, first scan all intervals on all arcs outward from node 5. We find that the number 2 exists in

Chapter2 Literature Review

the circular interval [6, 4], (the circular interval [6, 4] includes {6, 7, 1, 2, 3, 4}) so arc from node 5 to node 4 is chosen. With similar methods, the arc from node 4 to node 3 is chosen. At node 3, there are 3 intervals, among which [1, 2] includes number 2. Therefore the path is constructed as 5-4-3-2. At each node, the next arc to follow is determined by whether the destination node's label is bracketed by the interval of that arc.

Some highly regular networks such as 2D grids (alias meshes), 3D grids admit optimal 1-LIRS [64]. Trees and rings admit optimal 1-CIRS [66]. The efficiency of an optimal interval routing scheme is measured in terms of its compactness – the maximum number of intervals constituting the label of an edge. In terms of compactness, 1-LIRS or 1-IRS is the most efficient interval routing scheme.

Interval routing techniques were introduced by Santoro and Khatib discussing 1-LIRS [67]. Leeuwen and Tan subsequently generalized this scheme to more than one interval per edge [64]. The interval routing method has been implemented in the INMOS C104 and RCube routers [68, 69]. A short survey was presented by Leeuwen and Tan [70]. Since then, a lot of work has been done.

A detailed discussion of the impact of the node re-labeling on routing information complexity is presented [71]. Some special topologies like lithium graphs and weak-lithium graphs are observed and discussed in terms of their adaptability to IRS techniques [72].

Other characteristics of interval routing schemes are explored in a variety of researches [73-77]. As a fast routing technique, interval routing schemes have been developed thoroughly applied to computer network routers. But its application has not extends to other fields, like Euclidean path finding. In Chapter 4, interval routing schemes are applied to some common inside-building topologies in order to explore its possible usage in transportation operations. Part of the work in Chapter 4 has been

Chapter2 Literature Review

published [87, 88].

The last section of literature review is about flow control problem. The requirements for the flow control algorithm are similar to the shortest path finding algorithm. Low computation complexity and fast computing speed are of the first consideration.

2.7 Flow Control Methods

As a VR augmented system is designed as a tool for the rescue team in the context of emergency, it is very useful to provide evacuee assignment suggestions to the rescue team when they are trying to assign a large population to different exits from a dangerous location.

When one location, which in most cases can be assumed to be a room, of a building is on fire, people in or near that room will be evacuated through the exits on that floor as soon as possible. Suppose the rescue team have been there in a very short time after the breakout of the fire and the first task for them is to find where the exits are on the floor and how many evacuees would be assigned to each exit.

To solve this problem, some literature and previous work in indoor emergency evacuation is to be reviewed; and a fast flow control algorithm has been developed and discussed in Chapter 5.

To evacuate a large population of people from one location to several exits through a number of paths is an old network problem. Extensive research in flow control has addressed this problem. Hillier and Lieberman presents a classic algorithm [78]. If the network has multiple source nodes, a new source node is added and new arcs with the new source node at one end and each old source node as the other ends are added. If the network have multiple target nodes, a new target node are added and new arcs with the new target node as one end and each old target nodes as the other ends are

Chapter2 Literature Review

added. The capacities of new arcs are assumed infinite. So the network can always be transformed to be a network with one source node and one target node.

The idea behind the algorithm is to find augmenting paths one by one and augment the flow on them to its capacity until no augmenting path can be found. With this algorithm, the same network may receive several different maximum flow schemes depending on the order of augmenting paths the algorithm picks. If we give each arc in the network a weight, which can be deemed as the cost of moving one unit of flow through the arc, then we get a criterion to evaluate all the maximum flow schemes. For example, if we define the weight of an arc to be its Euclidean length and if we need a minimum cost maximum flow scheme, the schemes preferred will be the one, in which the augmenting paths are picked in ascendant order of its Euclidean length.

A maximum flow algorithm suggested by Ford and Fulkerson [79] and refined by Edmonds and Karp [80] chooses augmenting paths in the ascendant order of its path length. The path length defined here is not the Euclidean length but the depth from the root of the Breadth First (BFS) tree of the network to the target node. In each loop of the algorithm, a classic BFS algorithm [81] is executed to find the shortest path and then the path found is augmented. The arcs whose flows meet the capacities are deleted from the network at the end of this loop and the new network is built for the next loop. The algorithm ends when no shortest path can be found from the source node to the end node.

Taking Euclidean distance as the arc weight to calculate shortest paths, White developed another version of flow control algorithm [82]. And it is used in EVACNET+, which is an interactive computer program that allows the modeling of emergency building evacuations [83]. This model deals with a network consisting of a set of nodes connected by arcs. The nodes represent building components such as rooms, halls and stairs. The arcs represent the passageways between the objects represented by nodes. EVACNET+ determines an optimal plan to evacuate the

Chapter2 Literature Review

building in a minimum amount of time.

Another model to study the evacuation profile of a building, SAFE-R, makes some simplification on the network model used in EVACNET+ [84]. SAFE-R merges all source nodes lying on the same floor into one source node to avoid overlapping of paths, by which the time for calculating the shortest path in each loop is decreased.

From the perspective of implementation purposes, existing emergency evacuation software systems can be divided into two categories. One is mainly for simulation purpose. EXIT89 [89], EXODUS [90], EGRESS [91], SIMULEX [92], etc. are in this category. They provide no solution for optimization of overall evacuation process. EVACNET+ and its variation SAFE-R are in the other category, which can give optimal evacuation profiles through a simulation process. The network flow is optimized by a network flow transshipment algorithm due to White [82]. But White's algorithm is a dynamic programming, which means the optimal evacuation profile will be given after a number of stages of calculation. The computation at each stage has a very high complexity, especially when the network is very large in terms of the numbers of the nodes and arcs. The outcome of the earlier stage will be a part of the input for the later stage. The first step of this algorithm is to find out all paths from all source nodes to all exit nodes. Only this step can be very time consuming; and therefore this high-accuracy computation strategy is not suitable for our real-time evacuation tasks on a VR platform. Through some observation on common building structures, it is found that through some simplifications on the indoor structures of common buildings, a less accurate but more efficient algorithm can be applied, which is a reasonable solution for our real-time system.

In Chapter 4 and 5, the algorithms developed for this research are discussed in detail. Many basic concepts and helpful ideas come from the literature reviewed above.

Chapter 3 Research Methodology

3.1 Flow of Research Work

As the main objective of this research is to develop a fast shortest path finding algorithm and an effective flow control method for a project which is about a large indoor area evacuation system augmented by VR, its methodology is similar to that of a large-scale software development project. The overall working flow chart is as follows in Figure 3.1.

Figure 3.1 Research Working Flow Chart

Chapter 3 Research Methodology

The research work is divided into three stages: preliminary stage, development stage and final stage. Items in each stage in the flow chart are explained as follows.

3.2 Preliminary Stage

Research objectives are defined first. Then the scope of the research is identified, and relevant literature and previous work are reviewed. In this stage, tools for the development of the systems, hardware and development software, are reviewed in terms of the objectives of the research and also to the knowledge gained from previous work reviewed.

3.3 System Development Stage

After the preliminary stage, the research enters the development stage. Certain hardware and development software are chosen.

Though the VR environment is the platform to test the algorithms up to be developed, the platform can be developed in parallel to the algorithm development.

A brief performance comparison of the new shortest path finding algorithm and the Dijkstra's shortest path algorithm will be given in chapter 4.

When the 3D VR environment and all mathematic algorithms are completed, respective shortest path finding and flow control algorithms can be coded as functions. And some of theses functions will be integrated with the VR environment.

When most of coding work has been done, the VR environment will be equipped with the chosen hardware. Some coding may be needed to integrate the drivers for certain hardware to the VR platform. After all hardware mounted, a full functioned system is ready to be tested.

Chapter 3 Research Methodology

3.4 Final Stage

When the system is integrated, it will be tested against the objectives of the research. Efficiency of the algorithms developed will be tested in terms of its accuracy and computing speed. During the test, modifications to the algorithms and the software codes are allowed.

The research methodology is a critical reference for research schedule. Algorithm development is the core in our methodology flow chart and most of original academic findings of our research will be produced in this part. Currently the flow control algorithm has been integrated into the testbed system. Although not implemented as a real tool for our testbed evacuation system, the routing algorithms can also be integrated and developed into a real-time road guide for the system in some future work.

Chapter 4 Shortest Path Finding Algorithm

4.1 Introduction

In this chapter, some common indoor structures are analyzed first with their *spatial map diagrams*. They can present most indoor layouts in modern buildings, which follow some high regular structures. For these highly regular structures, Dijkstra's algorithm is unnecessary to be used. Instead, interval routing scheme (IRS), a technique dealing with static nodes in computer network routing, can be used. In Chapter 2, it has been demonstrated through an example that Dijkstra's algorithm fans out from the origin, successively identifying the shortest path to each of the nodes of the network in the ascending order of their shortest distances from the source, thereby solving the problem when the target node is reached.

The whole process of Dijkstra's algorithm is like a growing tree T . The root of the tree is the source node s and in each recursion, one node is added. The sequence of these addition operations is in order of each node's distance to s . When the target node t is added, the problem is solved. At last only a small portion of the nodes on this growing tree consists of the shortest path p from s to t . This means the recursions, in which the nodes not in the p are added to T , waste computing time and are not desirable.

When path finding follows the routing information stored in the routing table in each node, which is widely used in computer network routing, the path finding recursions only add useful nodes to p . This saves a lot of computing time in most situations, especially for networks with a highly regular topology like grid, tree or ring (these structures are called element classes). In this case, the size of the routing table for each arc from each node can be reduced to only two numbers, which represents a linear or circular interval of natural numbers. With the labels on the nodes and the interval on each arc, a path can be found without unnecessary node visiting and the

Chapter 4 Shortest Path Finding Algorithm

memory space required in computation by the routing table is reduced to the lowest level. This suits VR systems very well. This technique is called IRS as has been reviewed in Chapter 2.

As IRS needs the nodes labeled and arcs given intervals before it works for the path computation, it only can be applied to computer networks as a fast routing technique. For transportation path findings with dynamic sources and targets, it can not be applied directly. The algorithms for several highly regular structures, which frequently appear indoors, are developed in this chapter to extend the IRS's use from static to dynamic path finding.

Large indoor areas often consist of a collection of connected element classes. With IRS, system's dependence on Dijkstra's algorithm is lowered significantly. But in some cases, Dijkstra's algorithm is still needed, especially for complex networks. In this case, with the IRS in each element class, the large network can be simplified to be a small network, in which the numbers of nodes and arcs are significantly reduced. Then this simplified network can be input to Dijkstra's algorithm. This algorithm is illustrated in section 4.8.

The computation complexity comparison of Dijkstra's algorithm and dynamic IRS algorithm has been given in section 4.7.

After the dynamic IRS algorithm is developed and described, an extension of this algorithm is given, which is called 2-layer dynamic IRS algorithm. This algorithm is on the basis of 1-layer dynamic IRS and can be applied to more complex topologies than a single Element Class (EC). This algorithm is given in section 4.9.

4.2 Spatial Map Diagrams of Common Indoor Structures

To find an efficient solution to a problem, the first step should be an insightful analysis on the characters of the problem. Therefore, to find the efficient algorithms for indoor path finding, common indoor structures should be analyzed first bases on their spatial map diagrams.

4.2.1 2D Grids

Let look at a frequently appearing layouts in most flat buildings first. In Figure 4.1 (a) an (b), the plan spatial maps for two neighboring floors A and B are shown. The building spatial maps are drawn after the formal graduate apartment at Block 553, Upper Jurong, which is a 10-floor flat building.

(a) Spatial Map Diagram: Floor A

(b) Spatial Map Diagram: Floor B

Figure 4.1 Spatial Map Diagram for a 2D Grid Structure

Chapter 4 Shortest Path Finding Algorithm

The dashed lines are the spatial map diagrams for floors A and B. A is one storey above B. Floors A and B are connected by three staircases. Each staircase connects the all the floors of the building. In the spatial map diagram, the path between node A1 and B1 are a sequence of connected segments from A1 to staircase 1 to B1. If we straighten this segment sequence to make it one arc from A1 to B1, and do the same manipulations on the paths A2 to B2 and A3 to B3 add on all other floors, the spatial map diagram of the whole building is looked as in Figure 4.2.

Each vertical segment in Figure 4.2 is actually a sequence of connected segments, but this straightening manipulation simplified the path finding calculation significantly and does not affect the accuracy of calculation result. When a path is calculated and there are any this kind of vertical arcs included in the path, only a simply conversion from the arcs' calculation forms to its physical forms is needed in order to present the path found on VR displays.

Figure 4.2 the Simplified Spatial Diagram for a 2D Grid

Notice that floor C is the second floor, and the two entrances are trivial for the path finding calculation. The 10-floor building can be treated as a 9x3 2D grid, which is a perfect structure for 1-LIRS (1-Linear Interval Routing Scheme) [43]. If the path finding is between any static nodes like A1 to C3 or B3 to C1, Dijkstra's algorithm is

Chapter 4 Shortest Path Finding Algorithm

not needed. But if the dynamic source or/and target locations are involved, 1-LIRS can not directly applied to the structure and some algorithm has to be developed to embrace this dynamic path finding in order to extend the usage of IRS in computer network routing to dynamic Euclidean shortest path finding.

4.2.2 3D Grids

Some complex indoor structures can be converted to 3D Grids, an example for which is shown in Figure 4.3. The building spatial maps are drawn after N1-B2 and N1-B3 in NTU campus.

Chapter 4 Shortest Path Finding Algorithm

(a) Spatial Map Diagram: Floor A

(b) Spatial Map Diagram: Floor B

Figure 4.3 Spatial Map Diagram for a 3D Grid Structure

The dashed lines are the spatial map diagrams for floor A and B. A is one storey above B. Floors A and B are connected by three staircases. Each staircase connects all the floors of the building. In the spatial map diagram, the path between node A1 and B1 are a sequence of connected segments from A1 to staircase1 to B1. In if we straighten this segment sequence to make it one arc from A1 to B1, and do the same

Chapter 4 Shortest Path Finding Algorithm

manipulations on the paths A4 to B4, A5 to B5 and A8 to B8 add on all other floors. As for A2 to B2, A3 to B3, A6 to B6 and A7 to B7, the situation is a little bit complicated. But as Staircase5 is very close to A2 and B2, the path from A2 to B2 can be treated as a path as a sequence of A2-Z1-Staircase5-Y1-B2. And in the same way, the path from A6 to B6 is A6-Z1-Staircase5-Y1-B6. Similarly the path from A3 to B3 is A3-Z2-Staircase6-Y2-B3; the path from A7 to B7 is A7-Z2-Staircase6-Y2-B7. If we straighten all these segmental paths to be single segments, the spatial map diagram of floor A, B and C appears as in Figure 4.4.

Figure 4.4 Simplified Spatial Map Diagram for a 3D Grid

Notice that the length of arc A1-B1, A5-B5, A4-B4 and A8-B8 are the same and are equal to the total length of their path sequences. In similar way, the length of arc A2-B2, A3-B3, A6-B6 and A7-B7 are the same and are equal to the total length of their path sequences. But A2-B2, A3-B3, A6-B6 and A7-B7 are a little longer than A1-B1, A5-B5, A4-B4 and A8-B8. But the length difference (3-5m) can be ignore and the IRS can still be applied to the structure, due to the relatively large building size and low accuracy level of the location tracker (average 1m).

Chapter 4 Shortest Path Finding Algorithm

Also notice that A2-B2 and A6-B6 are separate arcs; but in fact, they overlapped. It does not affect the computing result but can make the computation much more efficient to make the simplifications as in Figure 4.4. The same manipulations are applied between floor B and C.

1-LIRS can be applied to the 3D grid structure in Figure 4.4. Similarly as the 2D case in section 4.2.1, if the path finding is between any static nodes, Dijkstra's algorithm is not needed. But if any dynamic node is involved, some algorithm has to be developed to extend the usage of IRS.

4.2.3 Trees and Rings

Some other structures can be found in the form of trees and rings. One example of a tree structure is a typical unit of most of blocks on Jurong West. It is a unit in a 12-story apartment building. All floors of the unit are connected by a staircase. On each floor of the unit there are 4 apartments. The simplified spatial map diagram for one floor is in Figure 4.5 and for the whole building in Figure 4.6. 1-IRS (1-Circular Interval Routing Scheme) can be applied to this tree structure [46].

Another 1-CIRS structure is ring [43]. Rings also frequently appear in inside-building topologies. Some on-campus residence halls of NTU take this form. One example is shown in Figure 4.7

As the building structures (or called topologies in another way) we are talking about here are all based on inside-building spatial map diagrams, when we mention building structures or topologies later in this report, they all refer to their topologies in their inside-building spatial map diagrams.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.5 Single Floor Layout for a Tree

Figure 4.6 Tree Structure of a Unit in a Block

Figure 4.7 A Ring Structure Example

4.2.4 Other Structures

In addition to grids, trees and rings, inside-building structures may take more complicated forms. Even in the same building, some floors may support 1-LIRS, but other floors may not support 1-LIRS. These more complicated topologies may support other interval routing schemes with worse compactness such as k -IRS, where $k > 1$. In this research, we only discuss 1-IRS indoor structures. There are three reasons why we do not include k -IRS ($k > 1$) structures in this research.

First, this research is the first one to try to extend IRS techniques, which has not been used for Euclidean path finding problems, to common inside-building structures. In traditional research fields about Euclidean path finding, like robot motion planning and transportation operation, no previous research includes the IRS technique. Therefore no such kind of research has made any inside-building topology analysis from the perspectives of IRS. This research is the first one. So all the knowledge about IRS and its applicable topologies come from network analysis on computer network routing. From the literatures in this field, we can find mature theoretical basis and methods for 1-IRS structure. As for k -IRS ($k > 0$), some arbitrary topologies have been analyzed, but they do not frequently appearing inside common buildings.

Chapter 4 Shortest Path Finding Algorithm

Second, k -IRS ($k > 0$) has much lower computation efficiency than 1-IRS. For most of indoor structures, k -IRS is not expected to be able to show much faster computing speed than traditional Dijkstra based algorithms. k -IRS or Dijkstra based algorithms, which one is better in terms of computing speed for an arbitrary complicated topology? To answer this question, a series of specific researches are needed, which are not the focus of this research.

Third, a lot of complicated indoor structures can be treated as a collection of connected 1-IRS structures; with this view, some divide-and-conquer idea like the one presented in complete intersection graph [59], can be applied with the help of the IRS. In some cases, the algorithm based on this idea can also show much better performance than traditional Dijkstra based algorithms. An algorithm based on this idea is illustrated later in this chapter.

4.3 Dynamic Shortest Path Algorithm for 2D Grids

4.3.1 IRS for 2D Grids – Optimal 1-LIRS

With the labeling method described by Leeuwen and Tan, any 2D grid allows an optimal 1-LIRS (1 Linear Interval Routing Scheme) [43]. An example for it is shown in Figure 4.8. Each static node in the grid is labeled with a number starting from 1. Each segment connecting two adjacent nodes represents two arcs in different directions. Each arc is labeled with a linear interval.

If the source and target points are just two static nodes, given the labeling information of these two points and follow the interval information on the arcs, a shortest path can be found very efficiently, because no useless nodes would be visited at all. For example, if we want to find the shortest path from node 7 to node 3. As node 3 is in $[1, 6]$, then node 4 is chosen. In the intervals of three arcs starting from node 4, $[1, 3]$ includes 3, then along corresponding arc, node 1 is selected. Then follow the same

Chapter 4 Shortest Path Finding Algorithm

procedure, node 2 and then node 3 is reached. The path is 7-4-1-2-3. No useless nodes are visited; this is the major advantage to Diskstra’s algorithm.

Figure 4.8 1-LIRS for a 2D Grid

*Dijkstra’s algorithm always builds a growing tree T without any consideration of the topology of the network. It looks for the node nearest to T and then adds it to the tree. This procedure starts from the source nodes and the recursive procedure will not end when the target node is reached and added to the tree. If Dijkstra’s algorithm is applied to the problem above, when node 3 is reached, node 1, 2, 4, 5, 6, 7, 8 and 9 have already visited and added to the tree. But the visits and additions of 5, 6, 8 and 9 are not necessagry. These visits on useless nodes are the main disadvantages of Dijkstra’s algorithm. Useless visits will significantly affect the performance of Dijkstra’s algorithm, especially when the network size is very large in terms of node number or/and arc number. **The strategy of adding routing information to the data structures of the nodes and arcs in the network will help to reduce***

Chapter 4 Shortest Path Finding Algorithm

useless arc and node visits significantly, especially for some special network structures. These structures include grids, trees and rings, etc.

To describe the IRS algorithm above, object oriented programming language will be used. Systematic definitions on the classes needed are shown below.

4.3.2 Static Algorithm

The classes we need to define a network $G(V, E)$ are Gridnode and Gridarc.

The class definitions and members are described in Appendix.

The main program *pathstatic()* returns an arc list, in which arc are connected head by tail. This arc list is the shortest path from node s to t , which can be sent as the input to the display processing program to present the path on HMD or other display devices for users.

4.3.3 Dynamic Algorithm

There are three cases with dynamic nodes involved; they are

- (1) Source node is dynamic, target is static;
- (2) Source node is static, target is dynamic
- (3) Source and target nodes are both dynamic

Here, we introduce some definitions needed in later discussion of the algorithm. We assume that any dynamic location is on a segment, which represent a piece of passageway or staircase. So it must be adjacent to two static nodes directly. We call these two static nodes *gating nodes* (GN) for this dynamic location. The shortest path from or to a dynamic node must pass one of the gating nodes of this dynamic node; and this gating node is called *optimal gating node* (OGN). So the problem here how to find the OGN of each dynamic node assigned as source or target node. If source S and

Chapter 4 Shortest Path Finding Algorithm

target T are both dynamic, which is in case 3 above, the shortest path from S to T is $S \rightarrow \text{OGN of } S \rightarrow \text{OGN of } T \rightarrow T$. The part OGN of S to OGN of T can be found easily with IRS information and the algorithm is described in section 4.3.2. Let's demonstrate the method how to find OGN with the 2D grid in Figure 4.9.

Case 1:

In this case, S is dynamic and T is static. Let S1 and S2 are GNs of S. If segment S1S2 is vertical, then cut the grid into two parts along a horizontal line passing S. T must be in one of the parts. OGN of S is the GN which is in the same part as T.

If segment S1S2 is horizontal, then cut the grid into two parts along a vertical line passing S. T must be in one of the parts. OGN of S is the GN which is in the same part as T. The path is $S \rightarrow \text{OGN of } S \rightarrow T$.

Case 2:

In this case, T is dynamic and S is static. Let T1 and T2 are GNs of T. If segment T1T2 is vertical, then cut the grid into two parts along a horizontal line passing T. S must be in one of the parts. OGN of T is the GN which is in the same part as S. If segment T1T2 is horizontal, then cut the grid into two parts along a vertical line passing T. S must be in one of the parts. OGN of T is the GN which is in the same part as S. The path is $S \rightarrow \text{OGN of } T \rightarrow T$.

Case 3:

As there are four OGNs in this case, the situation is a little more complicated. Let's illustrate it with a piece of a 2D grid in Figure 4.9. Let S be a dynamic location on a vertical segment. Let S1 be the upper GN and S2 be the lower GN ($S1.y < S2.y$ and $S1.x = S2.x$). If T is on a vertical segment, let T1 be the upper GN and T2 be the

Chapter 4 Shortest Path Finding Algorithm

lower GN ($T1.y < T2.y$ and $T1.x = T2.x$). If T is on a horizontal segment, let $T1$ be the left GN and $T2$ be the right GN ($T1.x < T2.x$ and $T1.y = T2.y$).

Cut the grid into two parts along a horizontal line passing S . If both of T_1 and T_2 are in the upper part like AB or CD , S_1 is the OGN of S . If T_1 and T_2 are both in the lower part like GH and IJ , S_2 is the OGN of S . If T_1 is in the upper part and T_2 is in the lower part like EF , then we have to compare $|SS_1|+|ET|$ with $|SS_2|+|TF|$. If $|SS_1|+|ET|$ is smaller, then the OGN of S is S_1 and the OGN of T is E ; otherwise, the OGN of S is S_2 , and the OGN of T is F .

This method can also be used to choose OGNs when S1S2 is horizontal ($S1.x < S2.x$ and $S1.y = S2.y$). In this case the cutting is vertical, and the similar procedures to find OGNs can be used.

Figure 4.9 Dynamic S and T in 2D Grid

The procedure can be described with the pseudo-code in Appendix.

Chapter 4 Shortest Path Finding Algorithm

The program *findpath2d* can find the shortest path between any two locations in a 2D grid with no useless node visits. In any case, *pathstatic()* runs for only one time, which significantly improves the algorithm’s efficiency.

4.4 Dynamic Shortest Path Algorithm for 3D Grids

4.4.1 IRS for 3D Grids – Optimal 1-LIRS

Similar to 2D grids, optimal 1-LIRS (1 Linear Interval Routing Scheme) is allowed for 3D grid. In Figure 4.10, the optimal 1-LIRS node labeling is implemented on the 3D grid.

Figure 4.10 1-LIRS for a 3D Grid Structure

The horizontal arcs are distributed to 3 levels; each vertical segment connects two levels. The arcs in one level are labeled with intervals in the same way as in 2D grids. A downward vertical arc’s interval includes all nodes below the arc. An upward vertical arc’s interval includes all nodes above the arc. With all the arcs and nodes properly labeled, the shortest path between any two static nodes in the grid can be

Chapter 4 Shortest Path Finding Algorithm

found fast and easily. To describe the IRS algorithm above, systematic definitions on the classes needed are shown below.

4.4.2 Static Algorithm

The definition for the node class in 3D grids is almost the same as 2D grids, but only with an additional coordinate member *z*. The 3D grid node class definition and the static algorithm are shown below with the C style pseudo code.

```
Class Gridnode {Arc [] arclist; int nodelb; int x; int y; int z}
/* Arclist is an array of the arcs starting from the node. Nodelb is the 1-LIRS label for the node. x and y
are the coordinates of the node in the grid*/
```

The static algorithm is the same as 2D grids, so the function *pathstatic()* can be used directly here.

4.4.3 Dynamic Algorithm

We notice that all vertical arcs represent staircases. As our location tracker only supports 2D location tracking [16], S or T assigned to any vertical segments will be arbitrarily moved to the arc's upper or lower static node. So we only discuss meaningful cases, where S and T's segments are both horizontal. Then we have two cases for the relative locations of S and T; they are:

- (i) S and T in the same *z* level and
- (ii) S and T in different *z* levels.

In case (i), *Findpath2d ()* can be directly applied, because OGNs of S and T can be found with their *x* and *y* knowledge only. In case (ii), except one situation, *Findpath2d ()* can be directly applied. This exception happens when the projection of dynamic node S's segment on T's level overlaps with T's segments. Let S1 and S2 be

Chapter 4 Shortest Path Finding Algorithm

S's two GNs, and T1 and T2 be T's two GNs. Assume S1's projection on T's level overlaps with T1, and S2's projections on T's level overlaps with T2. If $|SS1|+|TT1| \leq |SS2|+|TT2|$, then S1 and T1 are OGNs; otherwise, S2 and T2 are OGNs. The general algorithm for 3D grids is described with the pseudo codes in Appendix.

4.5 Dynamic Shortest Path Algorithm for trees

4.5.1 IRS for Trees – Optimal 1-CIRS

A tree has three important properties:

- (1) *it has no cycles and*
- (2) *when a tree is cut into two parts at any segment, each part is a new tree*
- (3) *there is only one single path between any two nodes on a tree, no matter if the nodes are dynamic or static. (A single node is trial tree.)*

Based on these properties, it has been proved that a tree allows optimal 1-CIRS (1 Circular Interval Routing Scheme) [14]. An example has been demonstrated in section 2.6. The 1-CIRS node labeling follows the depth first procedure. Arcs' interval labeling is based on property 2 and 3.

In Figure 4.11, the tree T can be seen as composed of two trees T1, T2 and the segment AB connecting T1 and T2. This view of T comes from property (2). According to property (1) and (3), the path from A to any node in T2, must pass segment AB. So the interval for the arc A-B must include labels of all nodes of T2. Vice versa, the interval for the arc B-A must include labels of all nodes of T1. If the nodes of T are labeled with the depth first procedure, the intervals on all arcs allow 1-CIRS. The algorithm for static path finding on a tree can be found in Appendix.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.11 1-CIRS on A Tree T

4.5.2 Dynamic Algorithm

There are three cases we need to discuss on the dynamic path finding algorithms for trees:

- (1) Source S is dynamic and Target T is Static;
- (2) Source S is static and Target T is dynamic; and
- (3) S and T are both dynamic.

In case 1, imagine that the tree Tr is cut into two sub-trees $Tr1$ and $Tr2$ at S and T is on $Tr2$. S 's GNs are $S1$ and $S2$, which exist in different sub trees. So one and only one among $S1$ and $S2$ exists on the same sub-tree with T . Let's assume $S2$ is on $Tr2$. So the interval of arc $S1$ - $S2$ must include $S2$ and T ; but the interval of arc $S2$ - $S1$ does not include T . In this case $S2$ is the OGN for S .

In case 2, imagine that the tree Tr is cut into two sub-trees $Tr1$ and $Tr2$ at T and S is on $Tr1$. T 's GNs are $T1$ and $T2$, which exist in different sub trees. So one and only one among $T1$ and $T2$ exists on the same sub-tree with S . Let's assume $T1$ is on $Tr1$. So the interval of arc $T2$ - $T1$ must include $T2$ and S ; but the interval of arc $T1$ - $T2$ does not include S . In this case $T1$ is the OGN for T .

Chapter 4 Shortest Path Finding Algorithm

In case 3, if S and T are on the same segment, the path is just a straight arc from S to T. If S and T are on different segments, first imagine that the tree Tr is cut into two sub-trees Tr_1 and Tr_2 at S and T is on Tr_2 . S's GNs are S_1 and S_2 , which exists in different sub trees. T's GNs are T_1 and T_2 , both of which exist on Tr_2 . Let's assume S_2 is on Tr_2 . The situation is illustrated in Figure 4.12. The interval of arc S_1 - S_2 must include S_2 , T_1 and T_2 ; but the interval of arc S_2 - S_1 does not include T_1 or T_2 . In this case S_2 is the OGN for S.

Figure 4.12 A Tree Cut by Dynamic S

After S's OGN is determined, T's OGN can be determined similarly. Imagine that the tree Tr is cut into two sub-trees Tr_3 and Tr_4 at T and S is on Tr_3 . T's GNs are T_1 and T_2 , which exists in different sub trees. S's GNs are S_1 and S_2 , both of which exist on Tr_3 . Let's assume T_1 is on Tr_3 . The situation is illustrated in Figure 4.13. The interval of arc T_2 - T_1 must include T_1 , S_1 and S_2 ; but the interval of arc T_1 - T_2 does not include S_1 or S_2 . In this case T_1 is the OGN for S.

Figure 4.13 A Tree Cut by Dynamic T

Chapter 4 Shortest Path Finding Algorithm

In case 3, we see that we only need to observe the intervals of the arcs between GNs in order to determine the OGNs for S and T. These arcs are S1-S2, S2-S1, T1-T2 and T2-T1. If S1-S2's interval includes T1 and T2, S2 is the OGN for S; otherwise, S1 is the OGN for S. Similarly, if T1-T2's interval includes S1 and S2, T2 is the OGN for T; otherwise, T1 is the OGN for T. The algorithm is described in Appendix.

4.6 Dynamic Shortest Path Algorithm for Rings

4.6.1 IRS for Rings – Optimal 1-CIRS

Many indoor structures are in circular form, which in calculation is often treated like a ring. A ring consists of a series of straight segments. Each node in the ring is adjacent to two segments; therefore, each node has two arcs starting from itself. The node and arc interval labeling method is based on “hops”, which is a concept used only in computer networks [43]. The distance from one node to another is measured with hops, not with Euclidean length. In our case, we don't concern about the hop numbers between nodes. Euclidean length is our concern. So we have to develop new node and arc interval labeling method based on IRS concepts and Euclidean distance instead of hop numbers.

Each node in the ring is adjacent to two segments; therefore, each node has two arcs starting from itself. A ring is shown in Figure 4.14. Dashed curves represent all segments not shown in the Figure.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.14 Nodes and Arcs Labeling for Rings

The labeling of nodes is simple. Start from any arbitrary node and choose an arbitrary direction (clockwise or counter-clockwise along the ring). Label all the nodes one by one from 1 to n . n is the total number of the nodes in the ring. As for arc interval labeling is little more complicated because Euclidean distance is used instead of hops.

Choose an arbitrary node E to in the ring in Figure 4.14 for example. Draw a dot line passing E , which cuts the ring into two halves. Another intersection point of the dot line and the ring is Q on segment MN . Make sure that the half $\{EG...PNQ\}$ is equal to $\{EF...OMQ\}$ in terms of Euclidean length. Then the shortest paths from E to any nodes in $\{EG...PNQ\}$ all take the clockwise direction; and the shortest paths from E to any nodes in $\{EF...OMQ\}$ all take the counter-clockwise direction. So the interval of the clockwise arc EG should include the labels of all nodes in $\{EG...PNQ\}$; and the interval of the counter-clockwise arc $E-F$ should include the labels of all nodes in $\{EF...OMQ\}$. Because the characteristics of the ring topology and our node labeling method, no matter how to choose E , the labels of all nodes on one of its two halves of the ring can be bracketed with only one circular interval. That means, for each of its two arcs to different directions, only a circular interval is needed. Accordingly,

Chapter 4 Shortest Path Finding Algorithm

1-LIRS can be applied not only to abstract rings in computer networks, but also to physical Euclidean rings. An example is shown in Fig 4.15.

Figure 4.15 A Ring Labeling Example

The ring's static algorithm, similar to that of trees is shown in Appendix.

Notice that for rings, the arclist array for each node only has two arcs. Each arc takes a direction of clockwise or counter-clockwise. So we can make a small change to the arc' class definition in order to facilitate the later dynamic algorithm. The class definitions for ring arcs and nodes are as follows:

```
Class Ringnode {Arc [] arclist; int nodelb;} // node definition
Class Ringarc {Ringnode stnode, ennode; float arclength; Boolean ifclockwise; int sint, eint;}
/* stnode and ennode are the start and end nodes of the arc. arclength is the Euclidean length of the arc;
sint and eint are respectively the first and second number of the linear interval label. Ifclockwise is a
sign indicating the arc's direction. */
```

Chapter 4 Shortest Path Finding Algorithm

4.6.2 Dynamic Algorithm

There are three cases when dynamic nodes are involved:

- (1) Source S is dynamic and target T is static;
- (2) S is static and T is dynamic; and
- (3) S and T are both dynamic.

In case 1, cut the ring with a line passing T into two equal halves. The cutting line intersects with segment EF at O. It is shown in Figure 4.16. AB is any segment whose nodes labels are both within the interval of arc T-G and therefore, AB is above the cutting line. If S is on the arc similar to AB, the OGN of S is A.

Figure 4.16 Static T and Dynamic S

C-D is any arc whose nodes labels are both within the interval of arc T-H and therefore, C-D is below the cutting line. If S is on the arc similar to C-D, the OGN of S is C. If S is on the segment of E-F, OGN of S can not be determined with only relative location considerations of GNs. The Euclidean length of the path from S to T

Chapter 4 Shortest Path Finding Algorithm

in counter-clockwise direction must be compared with that in clockwise direction. The smaller one is the direction of the shortest path from S to T. In case 2, the method is similar, but the cutting line is drawn from S instead of T. We illustrate the method for case 3 with the ring in Figure 4.17.

In the ring in Figure 4.17, S1 is the counter-clockwise GN of S; and S2 is the clockwise GN of S. T is the clockwise GN of T; and T2 is the counter-clockwise GN of T. For our convenience to illustrate the algorithm, draw a line passing S1 and T1; and draw another line passing through S2 and T2. If the interval of arc S1-S2 includes all of S2, T2 and T1, it means the length of the ring below the line S1-T1 is shorter than that above the line. So the clockwise path from S1 to T1 is shorter than the counter-clockwise path. In this case, S to T's shortest path should be clockwise; and therefore, S2 and T2 are OGNs for S and T respectively.

Figure 4.17 Dynamic S and T

If the interval of arc S2-S1 includes all of S1, T1 and T2, it means the length of the ring above the line S2-T2 is shorter than that below the line. So the counter-clockwise path from S2 to T2 is shorter than the clockwise path. In this case, S to T's shortest

Chapter 4 Shortest Path Finding Algorithm

path should be counter-clockwise; and therefore, S1 and T1 are OGNs for S and T respectively.

If none of the intervals of arc S1-S2 or S2-S1 includes all of S1, T1 and T2, the Euclidean length of the path from S to T in counter-clockwise direction must be compared with that in clockwise direction. The smaller one is the direction of the shortest path from S to T. The dynamic program can be described with the pseudo codes in Appendix:

4.7 Computation Complexity of 1-IRS Dynamic Shortest Path Algorithm

4.7.1 Preliminaries on Computation Complexity

The performance of an algorithm and the program coded according to it is usually evaluated with two methods. One method is **performance analysis**. The other one is **performance measurement**. Performance analysis involves **time complexity** analysis and **space complexity** analysis.

4.7.1.1 Space Complexity

Space complexity analysis is to evaluate the memory space a program needs to execute normally. It's very useful when the program requires large size data structure and code body. When the memory space the program needs exceeds the memory limit of the computer, the space complexity should be analyzed. The analysis should be able to find a solution to reduce the memory demand of the program to execute. With the computer technology advancing so fast, it's very common that a personal computer could be equipped with more than 1G-byte physical memory, which can hold most of large size program. Some commercial operation system like Microsoft Windows 2000 uses the paging technique to make it possible that the operation system can logically handle unlimited memory space which may be much larger than

Chapter 4 Shortest Path Finding Algorithm

the physical memory a computer is really equipped. Currently Windows 2000 and Windows XP can handle 4G-byte virtual memory. In fact, it's hardly to find a program which exceeds this memory requirement. Therefore space complexity analysis is rarely needed when an algorithm is implemented into a program. The necessary analysis is on time complexity, which can give a theoretical view of the program performance in terms of computation time.

4.7.1.2 Time Complexity

One way to estimate the time complexity of a program or function is to select one or more operations, such as add, multiply, and comparison and to determine the total number of the selected operations conducted in this program or function. The success of this method depends on our ability to identify the operations that contribute most to the time complexity. For a graph computing program like IRS or Dijkstra's shortest path algorithm, a node visit, an arc visit, or a necessary comparison operation can be counted reasonably as an important operation which we should base our time complexity analysis on.

Operation counts can be used to compare the time complexities of two programs that perform the same function, like the IRS program and the Dijkstra's program which compute a shortest path between two specified nodes in the same network. But operation counts can't yield a very accurate measure of time complexity. When using operation counts, we focus on certain "key" operations and ignore all others. Actually it is not necessary to know the exact count. If we have two programs with a complexity of $c_1n^2 + c_2n$ and c_3n , respectively, then we know that the program with complexity c_3n will be faster than the one with complexity $c_1n^2 + c_2n$ for sufficiently large values of n . For small values of n , either program could be faster. No matter what the values of c_1 , c_2 , c_3 , there will be an n beyond which the program

Chapter 4 Shortest Path Finding Algorithm

with complexity c_3n will be faster than the program with complexity $c_1n^2 + c_2n$. This value of n is breakeven point.

n here used is an example of instance characteristics. Instance characteristics typically include factors that determine the size of the problem instance. Take a grid for example, the row number r and column number c together determine the size of the grid in terms of the node number and the arc number. As an instance of the grid will be passed to the shortest path program, and the size of this grid instance directly determines the operation counts we defined, r and c can be treated as two key instance characteristics for our problem.

4.7.1.3 Asymptotic Notation (O)

The theoretical complexity of an algorithm is usually expressed with the big oh notation. The big oh notation provides an upper bound for the function f . The concept of the big oh is as follows.

Definition [Big oh]: $f(n)=O(g(n))$ if positive constants c and k exist such that $f(n) \leq cg(n)$ for all $n, n \geq k$.

$f(n)$ denotes the time complexity of a program measured as a function of the instance characteristic n . If there are more than one instance characteristics, like r and c in our case, the complexity can be denoted as $f(r,c)=O(g(r,c))$.

4.7.2 Time Complexity of 1-IRS Dynamic Shortest Path Algorithms

For 2D grids, the worst case is the case when the source and the target are respectively very close to the two different corners on a diagonal of the grid. In this case, the computation step number is maximal. With the IRS information stored for

Chapter 4 Shortest Path Finding Algorithm

each static node, the shortest path searching process will visit useful nodes only once without any visit on useless nodes. At each useful node, constant numbers of comparison operations are conducted. Therefore, for 2D grids, the worst case time complexity of the 1-IRS static shortest path algorithm is $O(r+c)$, where r is the number of rows in the grid and c is the number of columns. If the source and/or the target are dynamic, according to the 1-IRS dynamic shortest path algorithm, only several steps of computing on OGN selection are conducted. So the worst case time complexity is also $O(r+c)$.

For trees, with similar considerations, the worst case time complexity of the 1-IRS static shortest path algorithm is $O(D(T))$, where $D(T)$ is the diameter of the tree. The diameter of the tree is the longest possible path in the tree.

For rings, the worst case time complexity of the 1-IRS static shortest path algorithm is $O(n)$, where n is the number of static nodes in the ring.

If Dijkstra's shortest path algorithm is implemented with a standard heap, its computation complexity is $O(m+n\log n)$ (n is the number of static nodes in the network, and m is the number of edges). It's obvious that, compared to Dijkstra's shortest path algorithm, the routing strategy can significantly lower the computation complexity of shortest path finding especially inside a highly regular topology like an EC (Element Class).

4.8 Performance Comparison of 1-IRS Dynamic Algorithms and Dijkstra's Algorithm

4.8.1 Introduction

In section 4.7, the theoretical up-bound computation complexities (denoted by O) of 1-IRS dynamic algorithms to grids, trees and rings are given. This analysis implies

Chapter 4 Shortest Path Finding Algorithm

that 1-IRS dynamic algorithms may have faster execution speed than Dijkstra's algorithm when they are implemented for the same network. With the node and edge numbers increase, the speed difference may also increase. In order to prove this, several experiments are conducted.

All programs for this performance comparison are coded in standard C++ and run on an Acer TravelMate290 laptop with 1.4GHz Centrino CPU and 512 RAM. All the codes are included in the CD accompanied with this thesis.

4.8.2 Performance Comparison of 1-IRS and Dijkstra's Algorithms on 2D Grids

Two programs are coded for the performance comparison of 1-IRS dynamic algorithm and Dijkstra's algorithm on 2D grids. When the Dijkstra's algorithm program for 2D grids is run, the user should input an integer n . n is treated as the row and column number. And the network data structure of an $n \times n$ grid for Dijkstra's algorithm will be generated by the program automatically. Each node is given an integer label from 1 to $n \times n$. The sequence of the labels is from top to bottom and from left to right. In Figure 4.18, a 4×4 grid is labeled in this way. Two random locations, corresponding to dynamic source and target locations are also generated. In order to find the worst case performance, the distance between this two locations are made as long as possible. So the locations of these two nodes will be made respectively near two diagonal corners of the generated grid. Figure 4.18 also gives an example for this source and target generation. In this program, the length of all edges is set to 1.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.18 4 x 4 Grid and Source-Target Pair Generated by the Program

When the nodes are labeled and source and target locations are generated, this data structure is input into a Dijkstra’s algorithm. The shortest path will be given in the form of a sequence of node numbers, and the time for the shortest path computing is also given in the number of computer clock ticks. 1 tick is approximately 0.001 second. The output path for the network in Figure 4.18 may be like S-2-6-7-11-15-T. As the grid is too small in terms of node and edge numbers, the tick number for Dijkstra’s shortest path computation is 0.

Another program is implemented for 1-IRS dynamic algorithm. When an integer n is input, the same n x n grid and source-target pair are generated. The program will do 1-IRS node and arc labeling automatically to this grid. When this is done, the program will calculate the shortest path with 1-IRS dynamic algorithm. The output path is given in the similar way to the above Dijkstra’s algorithm program, and a tick number is given as well.

Table 4.1 shows the experiment data on Dijkstra’s algorithm and Table 4.2 shows that on 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

Table 4.1 Experiment Data of Dijkstra’s Algorithm on $n \times n$ 2D Grids

Dijkstra on 2D Grids ($n \times n$)	
n	Computer clock ticks
30	0
50	5
70	10
100	25
150	55
200	100
250	155
300	225

The curve in Figure 4.19 shows the relationship between computing time (in computer clock ticks) and grid size ($n \times n$) of Dijkstra’s algorithm on $n \times n$ 2D Grids. The curve in Figure 4.20 shows this relationship of 1-IRS dynamic algorithm.

Figure 4.19 Relationship between Computing Time and Grid Size of Dijkstra’s Algorithm on $n \times n$ 2D Grids

Chapter 4 Shortest Path Finding Algorithm

Table 4.2 Experiment Data of 1-IRS Dynamic Algorithm on $n \times n$ 2D Grids

1-IRS on 2D Grids ($n \times n$)	
n	Computer clock ticks
600	10
900	10
1100	10
1200	15
1300	15
1400	20
1500	20
1600	100
1700	105
1750	105
1800	430

Figure 4.20 Relationship between Computing Time and Grid Size of 1-IRS
Dynamic Algorithm on $n \times n$ 2D Grids

Chapter 4 Shortest Path Finding Algorithm

From the experimental result, it can be found that, the computing time of 1-IRS dynamic algorithm is much less than that of Dijkstra's algorithm. When grid size is under 50×50 , the speed difference is not significant. But when the grid size is larger, the difference is significant.

In Table 4.2, there is a sudden jump of clock ticks from 20 to 100 when n is between 1500 and 1600. This jump is due to the memory paging process of the Windows XP operation system taking place when the memory usage of the program is larger than the size of the physical RAM of the computer. When n is bigger than 1500, on the experiment computer, paging process is activated in the background. And this also explains the much larger slope of the curve when n is bigger 1600. This memory paging process is much related to the memory capacity of individual computers and the operation system, and not directly related to the algorithm.

In our testbed emergency evacuation system for N1-B3 floor of Nanyang Technological University, the system network is basically a 2×3 grid. Although 1-IRS dynamic algorithm can be used as primary path finding algorithm in our system, we use Dijkstra's algorithm, because there is no significant difference when the grid size is so small, which is shown in our experiment.

Additionally, the topology of N1-B3 (Figure 4.3(b)) floor of Nanyang Technological University is not a pure 2×3 grid. Actually there are some branch nodes and edges, and the overall network can be treated as a complex structure. To this structure, 2-layer 1-IRS dynamic algorithm, which is described in Section 4.10, can be applied. But in future research, we may need to extend our system to perform other functions, like real-time networks changing, which may make IRS information invalid in real time. So considering above facts, we use Dijkstra's algorithm as the primary path finding algorithm for our testbed emergency evacuation system in stead of 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

4.8.3 Performance Comparison of 1-IRS and Dijkstra's Algorithms on Trees

Two programs are coded for the performance comparison of 1-IRS dynamic algorithm and Dijkstra's algorithm on trees. When the Dijkstra's algorithm program for trees is run, the user should input an integer n . Then a tree data structure for Dijkstra's algorithm with $n \times n$ nodes is generated. Take $n=4$ for example, generated tree topology and node labeling are shown in Figure 4.21. This depth first labeling of the tree can be used directly for 1-IRS algorithm. The length of all edges is set to 1. S and T are set as the two ends of the tree to get the worst case computing time. For the tree in Figure 4.21. S and T are set to node 16 and node 7 respectively to get the worst case data. The result path is 7-6-5-4-3-2-1-14-15-16.

Figure 4.21 A Tree Generated by the Program

For the IRS program for trees, we just test the static 1-IRS algorithm, because except a couple of comparison operations, there is no significant performance difference between static and dynamic 1-IRS algorithms which is implied by the 1-IRS dynamic algorithm. Table 4.3 shows the experiment data on Dijkstra's algorithm and Table 4.4 shows that on 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

Table 4.3 Experiment Data of Dijkstra’s Algorithm on n x n trees

Dijkstra on trees (n x n)	
n	Computer clock ticks
30	0
50	5
70	10
100	25
150	55
200	100
250	155
300	225

Table 4.4 Experiment Data of 1-IRS Static Algorithm on n x n trees

1-IRS on Trees (n x n)	
n	Computer clock ticks
600	10
700	10
900	10
1000	20
1050	20
1100	105
1150	105
1200	105
1300	110

The curve in Figure 4.22 shows the relationship between computing time (in computer clock ticks) and grid size (n x n) of Dijkstra’s algorithm on n x n 2D trees. The curve in Figure 4.23 shows this relationship of 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.22 Relationship between Computing Time and Tree Size of Dijkstra’s Algorithm on n x n Trees

Figure 4.23 Relationship between Computing Time and Tree Size of 1-IRS Algorithm on n x n Trees

From the experimental result, it can be found that, the computing time of 1-IRS dynamic algorithm is much less than that of Dijkstra’s algorithm. This performance difference is significant.

Chapter 4 Shortest Path Finding Algorithm

4.8.4 Performance Comparison of 1-IRS and Dijkstra's Algorithms on Rings

Two programs are coded for the performance comparison of 1-IRS dynamic algorithm and Dijkstra's algorithm on rings. When the Dijkstra's algorithm program for rings is run, the user should input an integer n . n is treated as number of all nodes. And the network data structure of an n -node ring for Dijkstra's algorithm will be generated by the program automatically. Each node is given an integer label from 1 to n . Two random locations, corresponding to dynamic source and target locations are also generated. In order to observe the worst case performance, the distance between this two locations are made as long as possible. In this program, the length of all edges is set to 1.

Another program is implemented for 1-IRS dynamic algorithm. When an integer n is input, the same n -node ring and source-target pair are generated. The program will do 1-IRS node and arc labeling automatically to this ring. When this is done, the program will calculate the shortest path with 1-IRS dynamic algorithm. The output path is given in the similar way to the above Dijkstra's algorithm program, and a tick number is given as well.

Table 4.5 shows the experiment data on Dijkstra's algorithm and Table 4.6 shows that on 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

Table 4.5 Experiment Data of Dijkstra’s Algorithm on n-node Rings

Dijkstra on Rings(n)	
n	Computer clock ticks
751	4
1501	7
3001	10
4001	15
5001	20
6001	27
7001	30
8001	120
9001	120
10001	130

Table 4.6 Experiment Data of 1-IRS Algorithm on n-node Rings

1-IRS on Rings (n)	
n	Computer clock ticks
3001	5
4001	7
5001	10
6001	12
7001	12
8001	100
9001	102
10001	110

The curve in Figure 4.24 shows the relationship between computing time (in computer clock ticks) and ring size (n) of Dijkstra’s algorithm on n-node rings. The curve in Figure 4.25 shows this relationship of 1-IRS dynamic algorithm.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.24 Relationship between Computing Time and Ring Size of Dijkstra’s Algorithm on n-node Rings

Figure 4.25 Relationship between Computing Time and Ring Size of 1-IRS Algorithm on n-node Rings

The curves in Figure 4.24 and Figure 4.25 show that in worst case, 1-IRS and Dijkstra’s algorithms has no significant performance difference. This has been implied by the 1-IRS dynamic algorithm for trees. In worst case, 1-IRS may need to visit all

Chapter 4 Shortest Path Finding Algorithm

nodes and arcs in the ring, similar to Dijkstra's algorithm. But usually, the shortest path problem is not in the worst case condition, and therefore, 1-IRS algorithm can still save more node and arc visits than Dijkstra's algorithm.

4.9 Dynamic Shortest Path Algorithm for Complex Networks

4.9.1 Element Classes and Non-Classified Segments

As mentioned in section 4.2.4, a lot of complicated indoor structures can be treated as a collection of connected 1-IRS structures; with this view, some divide-and-conquer idea like the one presented in CIG (Complete Intersection Graph) [59], can be applied with the help of the IRS.

With the algorithms developed for the 1-IRS structures for grids, trees and rings, we can apply the IRS techniques in static computer networks to dynamic Euclidean networks. We have analyzed common inside-building structures and find that most of structures of single buildings can be modeled into the basic structures like grids, trees and rings. We call them Element Classes (EC). And one single node can be treated as a trivial EC here.

Sometimes the path finding source S and T may exists in different buildings, and the shortest path may pass through arcs and nodes belonging to multiple ECs. In emergency evacuation, this does not happen very often. Usually the users of our system are supposed to be rescue team members. They often have been in the emergency building before they are equipped and issue shortest path finding command to the system. But we still need to consider more complicated structures than ECs.

Some buildings have more complicated topologies than any 1-IRC EC. In this case we can treat this building as a collection of ECs connected by some special segments.

Chapter 4 Shortest Path Finding Algorithm

These special segments don't belong to any EC; therefore they are called Non-Classified Segments (NCS). All segments included in each EC are called Classified Segments (CS).

A more complicated network structure is shown in Figure 4.26. EC0 is a trivial EC. Five ECs are connected by a collection of NCS as shown. CSs are not shown because we do not need them for the overall Dijkstra's algorithm. We call this graph Classified Segment Hidden Graph (CSHG) of a complex network. Note that two ECs may be connected by a single intersection node, not a NCS. In this case, we can add a virtual edge with zero weight to build the CSHG of the network.

Figure 4.26 CSHG of a Complicated Structure

Any node of an EC, which is one end of any NCS, is called Boundary Node (BN) of this EC or the boundary node to the EC at the other end of the NCS. For Example, A is a BN of EC1, and a BN to EC2. B is a BN of EC2, and a BN to EC1.

4.9.2 General Procedure Description

When an Element Class EC_a is input to the overall Dijkstra's algorithm, it may take

Chapter 4 Shortest Path Finding Algorithm

two forms. Because the source location S and target location T are not in the same location, when S is in ECa , the overall Dijkstra's algorithm only cares about the length of the path from S to each BN of ECa . So we can draw ECa in a star shape as shown in Figure 4.27.

Figure 4.27 Star Form of an EC

All spokes of the star, SA , SB , SC , and SD are virtual segments. They are constructed with the fast dynamic IRS algorithms illustrated before. And the length of each virtual segment is the length of the real shortest path between the two end nodes of the virtual segment. Similarly, if T but not S is in ECa , the overall Dijkstra's algorithm only cares about the length of the path from each BN (Boundary Node) of ECa to T . So in this case, ECa still takes the star form to join the overall Dijkstra's algorithm.

Another form ECa may take to join the overall Dijkstra's algorithm is an n -partite. n is the number of EC s directly connected to ECa by some NCS . This happens when neither S nor T is located in EC . In Figure 4.28, ECa is in a bi-partite form, when A and D are BNs to the same EC , and B and C are BNs to another EC . AB , AC , DB and DC are all virtual segments; and the length of each virtual segment is the length of the real shortest path between the two end nodes of the virtual segment.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.28 2-Partite Form of an EC

As S and T are assigned in real time, the star form of an EC must be constructed in real time. The n-partite form of an EC can be constructed in advance in order to save run-time computing resources. Before S and T are assigned, EC of the network constructed in advance all take the form of n-partite. In this stage, the overall network consists of n-partite form ECs connected with NCSs. We call the graph of the overall network in this stage Primary Complete Connectivity Graph (PCCG).

When S and T are assigned two different ECs, EC1 and EC2, these two ECs are transferred into star form through real time path calculations. We call the graph of the overall network in this stage Complete Connectivity Graph (CCG). With CCG, Dijkstra's algorithm can be applied and a path p from S to T can be found. p is a sequence of arcs. Some arcs represent virtual segments, so each of these virtual arcs should be converted to real sub-path. Then the virtual path p is transferred to be the real path P.

The general procedure of this method is described in the flow chart in Figure 4.29. Following the procedure, let's look at how it works for the network in Figure 4.26. The PCCG of this network is shown in Figure 4.30. If S is assigned in EC1 and T is assigned in EC4, then the CCG is as in Figure 4.31. In CCG, EC1 and EC4 are both in star form.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.29 The Flowchart for Multi-EC Algorithm

Figure 4.30 The PCCG for a Network

Chapter 4 Shortest Path Finding Algorithm

Figure 4.31 The CCG for a Network

Figure 4.32 Virtual Path for a Network

After the Dijkstra's algorithm is applied to CCG. The virtual shortest path, which is drawn with bold segments in Figure 4.32, consists of S-D, D-C, C-B, B-A and A-T. S-D, C-B and A-T are virtual arcs, which can be converted to be real arc sequences with the IRS dynamic algorithms in EC4, EC3 and EC1 respectively.

Chapter 4 Shortest Path Finding Algorithm

4.9.3 Performance Considerations

To evaluate the performance of the algorithm, we compare its computing process with that of pure Dijkstra's algorithm applied to the whole real network.

In our algorithm, IRS dynamic algorithm is very fast. So Dijkstra's algorithm part is supposed to be the most time consuming part. So comparing the performance of this part to the pure Dijkstra's algorithm applied to the whole real network is meaningful for us. An algorithm's computing time is roughly proportional to the input size, which for a network algorithm is segment and node numbers.

Suppose we have an network with x ECs; they are EC_i ($i=1,2,\dots,x$). In EC_i , the node number is N_i and the segment number is A_i . The number of NCSs is a constant K . If we input the whole real network to pure Dijkstra's algorithm, the total node number N and total segment number is A ; they are:

$$N = \sum_{i=1}^x N_i$$

$$A = K + \sum_{i=1}^x A_i$$

In CCG of a network, most of ECs take n -partite form except the ECs where S or T stays. So node and segment numbers of PCCG can represent those of CCG's. For EC_i , CN_i is the node number of its n -partite form and CA_i is the segment number of its n -partite form. When we input the CCG to Dijkstra's algorithm, the total node number is CN and total segment number is CA ; they are:

$$CN = \sum_{i=1}^x CN_i$$

Chapter 4 Shortest Path Finding Algorithm

$$CA = K + \sum_{i=1}^x CA_i$$

So to allow our algorithm to have better performance, at least it is required that $CN \leq N$ and/or $CA \leq A$. If it is forced that $CN_i \leq N_i$ and $CA_i \leq A_i$ (it is called boundary requirements) for each i , the requirements is met. If an EC's CCG does not meet the boundary requirements, all real nodes and segments in the EC should join the Dijkstra's algorithm directly without any virtual form like star or n-partite. That means, in this case, this EC is ungrouped and is not treated as an EC in the following Dijkstra's algorithm.

Sometimes, an EC's CCG does not meet the boundary. One example is shown in Figure 4.33.

Figure 4.33 Real vs. Virtual Forms of an EC

In Figure 4.33, the EC is a ring with 6 nodes and 6 segments as in (a). 3 dashed segments connect the EC to another EC, called EC1. 3 dot-and-dash segments connect

Chapter 4 Shortest Path Finding Algorithm

the EC to another EC, called EC2. EC1 is different from EC2. So the n-partite form of the EC is a 3x3 bipartite, which has 6 nodes and $3 \times 3 = 9$ segments. So the n-partite form of the EC is not used. The EC is removed, and all its really nodes and segments should join the Dijkstra's algorithm directly.

When all ECs' CCG meet the boundary requirements, it can not be sure that the algorithm with ECs involved performed better than pure Dijkstra's algorithm. Because in order to implement EC involved algorithm, the data structure needed is unavoidably bigger than that of pure Dijkstra's algorithm. The space complexity of the EC involved algorithm is higher than that of pure Dijkstra's algorithm. Data structure size affects the speed of data transiting between computing resources on the computer. When the data structure is too big, it will significantly affect a program's running speed.

The performance of the algorithm with EC involved will be much better than pure Dijkstra's algorithm when the nodes and/or segments in the network can be significantly reduced by constructing CCG with virtual forms of the ECs. If it is not the case, its performance should be tested for arbitrary network.

4.10 2-Layer 1-IRS Dynamic Shortest Path Algorithm

4.10.1 Introduction

When constructing the CSHG (Classified Segment Hidden Graph) of a complex network as the one in Figure 4.26, we find that if the CSHG is a tree with ECs as its static nodes and Non-Classified Segments (NCSs) as its edges, the Dijkstra's algorithm is not necessary to be involved in the shortest path building process. This is due to the property of the tree topology. If the source and the target are located within the same EC, then the shortest path between them can be built with 1-IRS dynamic algorithms. If the source and the target are located in two different ECs, the NCS and

Chapter 4 Shortest Path Finding Algorithm

EC node series in the shortest path will be unique, which is one of trees' properties. If the NCS and EC node series can be determined by IRS information, the shortest path can be built without Dijkstra's algorithm involved at all.

The topologies whose CSHG are trees appear frequently in large modern buildings. Before a real example is given, let's look at the algorithm first.

4.10.2 2-Layer 1-IRS Dynamic Shortest Path Algorithm

The 2-Layer 1-IRS labeling is the same as in 1-Layer 1-IRS described before, except that the node label should add another integer number, the ECID. The node in CSHG (Classified Segment Hidden Graph) should be labeled with depth first method and the number given to each EC node is the ECID of that EC. This ECID should be added to the node label of each node in the correspondent EC. Then each node in the network will have a label of two layers, one is its ID in a single EC, and the other is the ID of this EC in CSHG of the overall network. The labeling of arcs inside each EC, which are called CSs before, is the same as in 1-Layer 1-IRS. The labeling of arcs outside ECs, which are called NCS before, is the same as in 1-Layer 1-IRS for tree; and the interval integers indicates the ECIDs. With this labeling, a systematic algorithm can be given to calculate the shortest path between any two dynamic nodes in the network.

This systematic algorithm, which is based on 1-Layer 1-IRS dynamic algorithm can be described in the flow chart in Figure 4.34.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.34 2-layer 1-IRS Procedure

In next section, a real example is given to demonstrate this algorithm.

4.10.3 An Example of 2-Layer 1-IRS Dynamic Shortest Path Algorithm

To illustrate the procedure in the previous section, we take the networks of N1-B3, N2-B3 and N1.1-B3 of Nanyang Technological University as an example.

The passageway topology of N1-B3 and N2-B3 are 2x3 2D grids; the N1.1-B3 a 2x2 grid. N1-B3 and N1.1-B3 are connected by a straight passageway; and N2-B3 and N1.1-B3 are connected by a straight passageway as well. The CSHG of the network is shown in Figure 4.35. AB and CD are two NCSs (Non-Classified Segements).

Chapter 4 Shortest Path Finding Algorithm

Figure 4.35 CSHG of the Example Network

We can see that the CSHG of the example network is a simple path, which is tree. So we can use 2-Layer 1-IRS dynamic algorithm to calculate the shortest path between any two dynamic nodes in the network. The network should be labeled first. The labeled network is shown in Figure 4.36.

Chapter 4 Shortest Path Finding Algorithm

Figure 4.36 2-Layer 1-IRS Labeling

The interval labels for arc AB, BA, CD and DC are for ECIDs. If the source node s and target node t exist in the same EC, which means their OGNs' label in ECID layer are all the same, 1-IRS dynamic algorithm can be applied to get the shortest path. This is the simplest case.

If *s* and *t* exist in different ECs as shown in Figure 4.28, we have to calculate the path from EC1 to EC3 in CSHG first with the IRS information stored for relevant BNs like A, B, C and D. For this example, the path in CSHG should be EC1-AB-EC2-CD-EC3. Then the path in each EC can be calculated respectively. In EC1, the path is from *S* to BN A. In EC2, the path is from BN B to BN C. In EC3, the path is from BN D to target *t*. The overall path can be built by connecting all the sub-paths found previously.

Chapter 4 Shortest Path Finding Algorithm

If s and/or t are on some NCS, the same procedure can be applied. The only difference is the path in CSHG may start and/or end by one or two dynamic nodes instead of one or two ECs.

4.11 Summary

In this thesis, common inside-building topologies are discussed first with their simplified accessibility graphs. According to their characteristics, IRS techniques in static computer networks are extended to the dynamic Euclidean networks. The dynamic algorithms for some frequent appearing structures, such as 2D grids, 3D grids, trees and rings are discussed and described by pseudo codes in Appendix. These codes can be converted to other object-oriented language codes. The theoretical computation complexity of the dynamic 1-IRS algorithms is analyzed for each kind of high regular topologies. The comparison between the performance of IRS algorithms and Dijkstra's shortest algorithm is also given through several experiments.

Complicated structures can be treated as a collection of connected ECs. Each EC can be a grid, a tree or a ring. In this view, a Dijkstra's involved algorithm is discussed. Its performance is compared with pure Dijkstra's algorithm, which is the classic algorithm in calculation Euclidean shortest paths.

For certain complex topologies, a pure IRS algorithm can be applied. 2-Layer 1-IRS algorithm is introduced as an extension of 1-Layer 1-IRS dynamic algorithm. In a previous research done by our project team [88], a special case of 2-Layer 1-IRS topology and its shortest path algorithm was given. The CSHG of the topologies in that research is in star shape, which is a special tree. The algorithm in that research can be deemed as a variation of the 2-Layer 1-IRS dynamic algorithm developed in this thesis.

Chapter 4 Shortest Path Finding Algorithm

For some topologies appearing frequently inside large buildings, the routing strategy like IRS can be applied. As its computation complexity is much lower than Dijkstra's algorithm, for large networks, it can be a better alternative method in shortest path finding. Additionally, as the IRS information is stored for each static node, some fast tools like real time road guide can be developed on the basis of IRS concepts. This can be done in future research.

Chapter 5 Flow Control Algorithm

5.1 Introduction

In a rescue evacuation system, a flow control algorithm may play a very important role, especially when the number of evacuees is relatively large. The flow control problem in this chapter is supposed to be independent of the shortest path research in chapter 4, although its solution can be integrated with the algorithms developed in chapter 4.

Before the flow assignment computation, the path finding computations should be done first to find the shortest paths from the source to all the exits on the same floor. After the shortest paths have been found and the length of each shortest path has been found in terms of traversal time, they will be treated as the input for the flow assignment computation.

Because the real evacuation processes may have different contexts, all existing evacuation algorithms for flow control have different limitations and assumptions. In our research, some assumptions are made according to the characteristics of the inside-building structures and real time evacuation tasks. Based on these assumptions, a flow assignment algorithm has been developed.

5.2 Problem Description

5.2.1 Introduction

Although not mentioned very obviously in their literature, there is a very important assumption when the capacitated network flow transshipment algorithm [82] is used, which has very high time and computation complexity. We can see from the

Chapter 5 Flow Control Algorithm

inside-building figure used in EVACNET+ and SAFE-R, almost all doors between passageways and stairs are omitted [83] [84]. These two figures are shown in Figure 5.1. The dynamic capacities of all the staircases are directly used as the input to the algorithm. This suggests that EVACNET+ and SAFE-R assume that the dynamic capacity of a door between a passageway and a staircase is much bigger than those of the passageway and the staircase, i.e. exit doors are not the bottlenecks for any evacuation flow. When a flow goes through this door, the bottleneck would be the passageway or the staircase, but not the door.

Figure 5.1 Building Structure in a Previous Research
Source: [84]

Chapter 5 Flow Control Algorithm

But the buildings subjected to our project are not in this shape. Take Block N1 in the campus of Nanyang Technological University and most office buildings for example, most exit doors are narrower than 1.20m, but the width of the narrowest passageway is wider than 2m. The widths of all passageways and staircases are all significantly wider than exit doors. This implies that in most of the cases these doors are the bottlenecks. A flow contributed by one evacuation source would fully occupy it and therefore, later flows on the same floor will have to wait until this flow is completely evacuated. With this observation, we can make an assumption for our research problem, which significantly reduces the computation complexity of the flow control algorithm. This assumption enables our flow control computation to give real time result with a linear method instead of going over a multi-stage dynamic program [82]. With this assumption, our flow control part of the system can be a powerful flow optimization tool for real time tasks, compared to existing evacuation software, which can only be used for process simulation [89-92] or slow flow planning [83,84].

5.2.2 Preliminary Definitions

Before we start our discussion about the flow control problem, some useful definitions are introduced. They appear frequently in emergency evacuation literature and can facilitate our discussion.

5.2.2.1 Nodes and Arcs

The building is described with the help of a network. The network consists of a set of points (nodes) connected by line segments (arcs). Nodes are used to describe the building spaces, such as rooms, corridors, stairs, halls, etc. The arcs represent the travel paths between two nodes. An arc is the passageway through which occupants traverse.

The definition of the nodes is different from those in chapter 4, but they can be treated

Chapter 5 Flow Control Algorithm

as dynamic locations in chapter 4.

5.2.2.2 Traversal Time (TT)

Traversal Time (TT) is the amount of time which is required to pass through the length of an arc. In our algorithm, traversal time is obtained by dividing arc length with the average walking speed of the occupants. The extent of traversal time depends upon the age, sex and the number of people in the streams as well as on the environmental conditions. Some human behavior research has analyzed experimental data of pedestrian movement in linear channels and suggested some formulas to calculate TT in certain situations [85]. But this would be too complicated for our real time system. Some simplifications must be made.

TT represents the length of an arc in terms of the time needed to pass through it. We can just use the average walking speed of common people, and calculate TT for each arc. People in emergency may have faster walking speed; this can be analyzed later by some human behavior experiments in order to make the flow control method workable for real evacuation process.

The TT for a complete evacuation path, which consists of a series of connected arcs, can be called the Total Traversal Time (TTT) of that path.

5.2.2.3 Dynamic Capacity (DC)

Dynamic Capacity (DC) is the number of people that can flow through an arc in a given time. DC is represented in the form of number of people moving per unit time period. In a sense, an arc's DC can represent its evacuation capacity. If the length of the time period is given, for example 1 second, the DC is in people/sec.

Chapter 5 Flow Control Algorithm

5.2.2.4 Wide Doors and Narrow Doors (WD and ND)

In most office buildings, shopping centers and campus buildings which are the major applications of our evacuation system, there are exit doors of a variety of sizes. Through our observation, most exit doors can be divided into two categories in terms of its relative width to that of the passageways. The width of an exit door is denoted by W_d in the following discussion. In a building, different segments of passageways may have different widths. The maximum passageway width of the building is denoted by W_{max} ; the minimum passageway width of the building is denoted by W_{min} . The widths of one category of the exit doors in a building are much less than W_{min} . Take Block N1 of Nanyang Technological University for example, W_{min} is wider than 1.8 meters. But a standard exit door is only 0.8 meters width. We call these types of doors, whose width is equal to or less than W_{min} , Narrow Doors (ND). The width of another category of exit doors is bigger than the W_{max} of the building. In most cases they are the main gates of the building. On one floor, there are only one or two exit doors of this type. We call them Wide Doors (WD).

A single WD can usually be converted to several NDs, whose locations are very close to the WD. It is observed that the WD usually appears as the main gate of the building and people usually reach a hall after entering the main gate and can proceed to different parts of the building through different passageways, which branch out from the hall. We can logically install NDs at the entering locations of these passageways. In this way, if a WD exists, it can be converted to multiple NDs.

Chapter 5 Flow Control Algorithm

5.2.2.5 Single Narrow Door (ND) Problem

If there is only one Narrow Door (ND) on the floor, all evacuees should evacuate from this door. The first question is whether the evacuees should be divided into more than one group, with each of them taking a different path to the same exit door.

Let N denote the total number of all evacuees, P denote the path a group of evacuees take to the ND, T_p denote the time needed to cover P , (i.e. the Total Traversal Time of P), and T_d denote the total time needed for all the members of an evacuation group to pass through the exit door. Then the total evacuation time T_e for this group is $T_p + T_d$.

As $T_p = L_p/V$, (L_p is the length of P , V is the average walking speed of evacuees), and $T_d = N_p/W$, (N_p is the number of evacuees in the group, W is the DC of door), so $T_e = L_p/V + N_p/W$.

When this group takes the shortest path from the source to the door, T_p is minimized. Suppose the evacuees are divided into n groups, with n_i evacuees in group i ($i=1,2,3,\dots,n$). Groups take different paths (if two groups take the same path, they are treated as one group). A group of evacuees are assumed to occupy the full width of the bottleneck of the path in the evacuation process. As the door is ND, it is assumed to be the bottleneck of the evacuation route of any group. Only one group can pass through the door during its evacuation period of time and no other groups can pass the door simultaneously. Thus, no matter how many groups are assigned, the total time spent in passing the exit door for all evacuation groups is N/W .

If all evacuation groups take their respective shortest paths (note that the shortest paths may not be the same), they will reach the door at the same time and start to pass the door one by one. The time required by the whole evacuation process will be $T_e = L_p/V + N/W$. L_p is the length of the shortest path.

Chapter 5 Flow Control Algorithm

We can see that, if the evacuees are treated as only one group and follow one of the shortest paths, T_e will not increase and this solution saves the work on group division. When we select an ND as the evacuation door, we can find a shortest path from the source to it and no further group division is needed. All the evacuees can be assigned to this path.

5.2.2.6 Single Wide Door (WD) Problem

When the target door is a Wide Door (WD), the situation is much complicated. Remember a WD's DC (Dynamic Capacity) is larger than W_{max} . Because the width of a WD is never the bottleneck of any evacuation group, it allows more than one group to pass through the door simultaneously. It implies that if we want to find an optimal solution to this one source and one WD target problem, we will have to find all the simple paths (a simple path is a path that repeats no vertex) from the source to the target. Different combinations of the simple paths should be checked to find the optimal solution. To find all the potential simple paths is a highly complex computation process. Most paths in the optimal combination may overlap each other frequently. This makes evacuation groups, frequently intersect with one another, which is not desirable in our situation. To avoid frequent interceptions between evacuation groups, we can make some simplifications and convert this complicated WD problem into an ND problem. This conversion comes from our observation on common buildings as mentioned in section 5.2.2.4.

Take Takashimaya (a shopping mall in Singapore) for example, the main gate of the first floor is a WD. We can simply treat it as a series of juxtaposing NDs. But as these NDs are very close to each other, if the final computation result includes more than one group, the groups assigned to these NDs may intersect with each other frequently. There is another way to convert WD problems to ND problems that can significantly alleviate this intersection.

Chapter 5 Flow Control Algorithm

Let us see the floor plan first. When a person goes through the main gate, he will enter a large open space. Around this open space, there are passageways which can take the person to other parts of the building. If a person not in the main gate area wants to get out of the building through the main gate, his path to the main gate must pass at least one of these passageways. The DC of these passageways are less than or equal to W_{max} . We can add dummy doors at the main gate area end of these passageways. The DCs of these dummy doors are less than or equal to W_{max} . If we make a further simplification and treat these doors as NDs, this single WD problem becomes a multiple-ND problem.

Notice that most buildings only have 1 or 2 WDs and this simplification is reasonable and straightforward.

5.2.2.7 Multiple-ND Problem

Our algorithm provides a fast solution for the multiple-ND problem. If there are multiple NDs on a floor of a building, the rescue team should decide how to divide the people stuck in a dangerous area on this floor into several groups and dispatch them to the exit doors within the shortest possible evacuation time. This evacuation time includes the time for covering the distance for all paths and the time getting through the exit doors.

If all doors in the system are NDs and a subset of these doors must be assigned some Evacuation Groups (EG) to get the shortest possible evacuation time, the optimal evacuation time for this evacuation process can be obtained from the choice of the door subset. That means each ND holds one and only one EG and the choice of doors decides the overall evacuation time of the evacuation process.

The mathematical derivation is as follows:

Chapter 5 Flow Control Algorithm

There are totally N evacuees at the source waiting to be evacuated. They are going to be divided into k EGs. EG_i has N_i evacuees ($i=1,2,3,\dots,k$). The path EG_i takes to an ND is P_i , and the length of P_i is L_i . V is the average walking speed of evacuees. W_i is the dynamic capacity of $Door_i$ (NDs are bottlenecks for all EGs), which P_i goes to. T_i is the total evacuation time of EG_i . Then:

$$N = \sum_{i=1}^k N_i \quad (5.1)$$

$$T_i = L_i/V + N_i/W_i \quad (5.2)$$

In an optimal solution, all EGs should finish evacuation at the same time, so $T_1 = T_2 = \dots = T_k = X$.

Thus,

$$X = L_i/V + N_i/W_i \quad (5.3)$$

$$N_i = (X - L_i/V) * W_i \quad (5.4)$$

Substitute N_i in (1) with N_i in (4), we get

$$N = \sum_{i=1}^k (X - L_i/V) W_i = X \sum_{i=1}^k W_i - \sum_{i=1}^k L_i W_i / V \quad (5.5)$$

$$X = (N + \sum_{i=1}^k L_i W_i / V) / \sum_{i=1}^k W_i \quad (5.6)$$

From above mathematic derivation, we can see if the destination ND subsets are given, the evacuation time X can be calculated. This evacuation time X may or may not be optimal. If the evacuation time is the shortest, the destination ND subset is called the *optimal ND subset*.

Chapter 5 Flow Control Algorithm

The subset of exit doors is not selected without any limitation. A reasonable exit door subset should make $X-L_i/V \geq 0$ in equation (5.5) for all doors in the subset. This makes sure that in the subset there is not a door such that the length of the shortest path to this door is too long so that even it is assigned only 1 person, it still will need longer time for the person to reach this door than the total evacuation time of all other EGs. If we allow any N_i to be negative, the resulting door set is clearly not optimal.

To get the optimal door subset, a straightforward method is by simple enumeration of all subsets of the door set. By enumerating all possible subsets of the door set and comparing their evacuation time, the optimal flow solution can be got. With a further understanding of the problem, the solution can be obtained with a much simpler method instead of the time consuming enumeration and comparison process. This method is given in section 5.3.

5.3 The Flow Control Algorithm for the Multiple-ND Problem

5.3.1 A Generalization for Multiple-ND Problems

When a source location is given, the shortest paths from this location to all NDs and their lengths can be calculated with some shortest path algorithm. If the average walking speed is given as well, the time needed to cover each shortest path can be calculated. So after the source location of the evacuees is given, we can construct an ordered time set $Tp = \{Tp_i, i=1,2,\dots,m\}$. Tp_i is the time needed to cover P_i , a shortest path from the source to $Door_i$. And the order of the Tp set is from the shortest to the longest, i.e. $a < b$ if $Tp_a < Tp_b$. If there are any equal values in the Tp set, the order between the equal values can be given arbitrarily. Each door is given a door label according to its Tp value, i.e. if the Tp to the door is Tp_c , then the label of the corresponding door is c . m is the number of NDs available for the evacuation process.

Chapter 5 Flow Control Algorithm

An ordered door can be built with each door corresponding to its Tp in the Tp set. This door set is $Door=\{Door_i, i=1,2,\dots,m\}$. $Door_i$ corresponds to Tp_i . So $Door_1$ is the ND nearest to the source location, and $Door_m$ is the farthest. The corresponding DC of the doors can be used to construct an ordered DC set $DC=\{DC_i, i=1,2,\dots,m\}$. The solution to a multiple-ND problem will include a subset of the Door set. And a path set can be built as $Path=\{Path_i, i=1,2,\dots,m\}$. $Path_i$ corresponds to Tp_i .

In section 5.2.2.7, it is illustrated that when a subset of the $Door$ set is given as a solution to a multiple-ND problem. The evacuation time can be calculated with equation (6). This evacuation time may be the shortest or not. If it is the shortest, we call this subset of the $Door$ set Optimal $Door$ Subset (ODS).

If the door subset, a solution to a multiple-ND problem, does not include $Door_1$, its evacuation time can be enhanced by assigning some number of evacuees to $Path_1$, because the EG to $Door_1$ (denoted by EG_1) can get to $Door_1$ earlier than other EGs getting to their respective doors (the order of elements in the $Door$ set corresponds to the order of their Tp in the Tp set). During this time difference, some people can be evacuated through $Door_1$. So this addition of $Door_1$ to the solution door subset can shorten the evacuation time. If a door subset includes more than one door, $Door_2$ must be included with the similar reason.

So we generate this conclusion as follows:

If the size of the ODS of the optimal solution to a multiple-ND problem is n , the ODS is $\{Door_i, i=1,2,\dots,n\}$, where $n \leq m$ and m is the number of NDs available for the evacuation process.

With this generalization, we can give an algorithm as described in section 5.3.2.

Chapter 5 Flow Control Algorithm

5.3.2 The Multiple-ND Flow Control Algorithm

It is assumed that there is at least one door in the final result door set. So $Door_1$ must be included first, because it is the ND nearest to the source location. Starting from $Door_1$, we decide if $Door_2$ should be added to the result door set, and $Door_3$, $Door_4$, and so on. Remember elements in $Door$ are in the same order as their corresponding Tp in the Tp set.

Let the number of evacuees be N . Assume the evacuation rate at each door to be the DC of that door considering the exit door to be the bottleneck. Let T denote the overall evacuation time.

When all evacuees are assigned to $Door_1$ (i.e. the door subset selected is $\{Door_1\}$), the total evacuation time is $T = Tp_1 + N/DC_1$. If $N \leq (Tp_2 - Tp_1) * DC_1$, then $T \leq Tp_1 + (Tp_2 - Tp_1) * DC_1 / DC_1$, thus $T \leq Tp_2$. So there is no need to assign any evacuees to $Door_2$. So the ODS of the solution to this multi-ND problem is $\{Door_1\}$.

If $N > (Tp_2 - Tp_1) * DC_1$, $T > Tp_2$, then add $Door_2$ to the result optimal door set can enhance this solution. So we add $Door_2$ to result set and we get the new evacuation

$$\text{time: } T = (N + \sum_{i=1}^2 Tp_i * DC_i) / \sum_{i=1}^2 DC_i$$

This T function is implied by equation (5.6) in section 5.2.2.7 and is rewritten below as equation (5.7). If $N \leq (Tp_3 - Tp_1) * DC_1 + (Tp_3 - Tp_2) * DC_2$, then $T \leq Tp_3$. So there is no need to assign any evacuees to $Door_2$. So the ODS of the solution to this multi-ND problem is $\{Door_1, Door_2\}$.

$$T = (N + \sum_{i=1}^k Tp_i * DC_i) / \sum_{i=1}^k DC_i \quad (k \text{ is the size of the door subset}) \quad (5.7)$$

Chapter 5 Flow Control Algorithm

If $N > \sum_{i=1}^k (Tp_k - Tp_i) * DC_i$ and $N \leq \sum_{i=1}^{k+1} (Tp_{k+1} - Tp_i) * DC_i$; then, according to equation (7), $T > Tp_k$ and $T \leq Tp_{k+1}$.

This means that, adding $Door_{k+1}$ into the solution door subset does not enhance the evacuation time. So the ODS is the first k elements in the $Door$ set.

So whole process of the algorithm can be described as follows:

Step1: construct the ordered Tp , ordered DC set, and ordered $Door$ set with the source location and walking speed information provided.

Step2: find the k value, where

$$N > \sum_{i=1}^k (Tp_k - Tp_i) * DC_i \text{ and } N \leq \sum_{i=1}^{k+1} (Tp_{k+1} - Tp_i) * DC_i$$

Step3: the optimal door set is the first k doors in $Door$ set, so the total evacuation time

is $T = (N + \sum_{i=1}^k Tp_i * DC_i) / \sum_{i=1}^k DC_i$ and the evacuee number assigned to each door is

$N_i = (T - Tp_i) * DC_i (i=1,2,\dots,k, \text{ where } k \leq m \text{ and } m \text{ is the number of available evacuation doors}), \text{ which is implied by equation (4) in section 5.2.2.7}$

5.4 K-Limited Flow Control Algorithm

If there is a limitation on the number of EGs, the algorithm in section 5.3 may not give a feasible solution. The algorithm in section 5.3 can give an optimal door subset, which can have as many as m EGs, where m is the number of element in the Tp set. The size of $Door$ set is also m .

Chapter 5 Flow Control Algorithm

In a real evacuation task, there is always a limitation on the number of systems available for rescue team members, and this number k is usually less than m . When this k limitation exists, an enumeration method has to be used. This method is called k -limited flow control algorithm.

This algorithm is described as follows:

Step 1:

Find a shortest path from the source to each ND. Construct the ordered T_p and Door sets with this shortest path information, walking speed and the source location. Let m be the size of T_p set. In this step, all 1-element result door subset is tested and compared. For each door, there is a specific evacuation time T if all evacuees are assigned to that door. Then we can find the minimum T for a one door solution, we call it T_{min1} . Dismiss all the doors whose T_p is equal or larger than T_{min1} in the following steps, because these doors are too far from the source and it will never appear in the ODS. This dismissing will modify T_p and Door sets. After this dismissing, the total ND number we will consider in the next step is decreased to N_1 . If N_1 is one, the algorithm can stop and the ODS is the set with only one ND.

Step 2:

Get all combinations of two different NDs from the modified Door set in step 1. And we can get all 2-ND solutions. Denote the minimum T of all 2-ND solutions as T_{min2} . If T_{min2} is equal or bigger than T_{min1} , the algorithm can stop and the optimal result door subset is the set with only one ND. If T_{min2} is smaller than T_{min1} , dismiss all the doors whose T_p is equal or larger than T_{min2} in the following steps, because these doors are too far from the source and it will never appear in the ODS. After this dismissing, the total ND number we will consider in the next step is decreased from N_1 to N_2 . If $N_2=2$, the algorithm can stop and the optimal result door subset is the set with the two left

Chapter 5 Flow Control Algorithm

NDs.

.....

Step i ,

In i th step, get all combinations of i different NDs from the modified Door set in step $i-1$, and calculate T_{min} of i -ND solutions. If $T_{mini} \geq T_{min(i-1)}$, the algorithm can stop and the optimal result door subset is the $(i-1)$ -door solution with $T_{min(i-1)}$ in step $i-1$. If T_{mini} is smaller than $T_{min(i-1)}$, dismiss all the doors whose T_p is equal or larger than T_{mini} in the following steps. After this dismissing, the total ND number we will consider in the next step is decreased from N_{i-1} to N_i . If $N_i = i$, the algorithm can stop and the optimal result door subset is the set with the i left NDs

...

Step k , (remember k is the maximal number of EGs allowed)

In k th step, get all combinations of k different NDs from the modified Door set in step $k-1$, and calculate T_{min} of k -ND solutions. If $T_{mink} \geq T_{min(k-1)}$, the algorithm can stop and the ODS is the $(k-1)$ -door solution with $T_{min(k-1)}$ in step $k-1$. If T_{mink} is smaller than $T_{min(k-1)}$, the algorithm can stop here and the optimal result door subset is the set with the k left NDs

In real situations, the algorithm may get the optimal result door subset much early than the k th step. In this enumerating and comparing way, the k -limited optimal solution will be found. This algorithm may be frequently used in real evacuation tasks, because there is usually limited number of our systems, which is likely to be less than the number of doors on the floor.

Chapter 5 Flow Control Algorithm

5.5 Summary

In a real-time system, especially for emergency evacuation, computation tasks or functions should be able to be finished in a reasonably short time. Users of our system are supposed to be rescue team members. When they are evacuating a large group of people, how to assign evacuees to different exits to shorten the evacuation time to a largest degree is a big challenge. With some flow control functions provided, the system should be able to provide real-time evacuee division and dispatching suggestions within a short time of computation. Existing evacuation software's flow control algorithms are not effective enough for our system in terms of their theoretical computation complexity and actual computing speed, especially when the network is large in terms of arc and node numbers.

The flow computation methods of some existing evacuation software have been reviewed first. As the tasks of our system are unique compared to existing evacuation systems, the assumptions of our flow control computation are evaluated. The characteristics of the target buildings of our system, such as shopping malls, campus buildings, and large office buildings are analyzed. On the basis of all these observations, our problem is modeled as a multiple-ND problem and two fast flow control algorithms, slimmed for our VR augmented emergency evacuation system, have been developed. The first algorithm introduced is used when there is no limitation on the number of systems. And the second is used when there is a k limitation. These two algorithms have been implemented in our evacuation system.

In Chapter 6, our flow control algorithms are implemented for the first floor of Takashimaya, a shopping center in Singapore, in order to demonstrate their performance in real-time conditions. A complete testbed emergency evacuation system with location trackers, orientation trackers and HMD integrated is also implemented for the floor of N1-B3 of Nanyang Technological University, which is described in Chapter 6 as well.

Chapter 6 Virtual Reality Augmented Evacuation System for Large Indoor Areas

6.1 Flow Control Application to Takashimaya Shopping Center

To demonstrate the flow control algorithms, a program is coded in VC++ 6.0 for Takashimaya Shopping Centre on Orchard Road, Singapore. The primary path finding method is Dijkstra's shortest path algorithm. As the network is not highly regular, IRS algorithm is not suitable. The floor plan of Takashimaya is shown in Figure 6.1.

Figure 6.1 Takashimaya Floor Plan

When the program is initially run, the network of Takashimaya converted from the floor plan is shown in the main window of the program as in Figure 6.2. Node 1, 8, 14, 27, 30 are Narrow Doors (ND), whose DCs are recorded in a file named “doors.txt”. The whole network data structure is saved in a file named “post.txt”.

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

Figure 6.2 Takashimaya Network

A user can initiate a path finding command with the menu item, “Dijkstra (DC not considered)”, under the view menu. After this item is clicked, a dialog box will appear to allow the user to input the locations of the source and the target. This dialog box and some input information are shown in Figure 6.3.

The dialog box is titled "Dijkstra (no DC considered)". It contains two main sections: "Information of Source:" and "Information of Target:". Each section has three input fields: "Gating Node 1:", "Gating Node 2:", and "Distance to GN1:". The "OK" and "Cancel" buttons are at the bottom.

Information of Source:		
Gating Node 1:	2	Distance to GN1: 2
Gating Node 2:	4	

Information of Target:		
Gating Node 1:	18	Distance to GN1: 4
Gating Node 2:	22	

Figure 6.3 Input Dialog Box for Path Finding

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

When the OK button is clicked, the shortest path from the source to the target is shown in the main window of the program, which is shown in Figure 6.4.

Figure 6.4 Output Window for Path Finding

A user can initiate the non-k limited flow control algorithm by clicking the menu item, “No-k-optimal flow assignment”, under the view menu. The input dialog for this command is shown in Figure 6.5

Figure 6.5 Input Dialog Box for Non-k-limited Flow Control

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

The grouping profile will be given on a message box, which is shown in Figure 6.6, and be saved automatically in a file named output.txt. The routes to exit doors will be displayed in different colors. The output in program's main window is shown in Figure 6.7

Figure 6.6 Output Message Box for Non-k-limited Flow Control

Figure 6.7 Output Window for Non-k-limited Flow Control

Chapter 6 Virtual Reality Augmented Evacuation System for Large Indoor Areas

A user can also initiate a k-limited flow control computation by clicking the menu item, “k-optimal flow assignment”, under the view menu. The input dialog box is similar to the one shown in Figure 6.5, except that the limit on the number of systems should be provided. And the output message box and the route display in main program’s main window are similar to non-k-limited computation. The grouping profile is also saved automatically in a file named output.txt.

As our flow control algorithms can give the computation result very fast (less than 1 second), they are suitable for a real-time emergency evacuation. The Dijkstra’s shortest path algorithm is also good enough as primary path finding method.

6.2 Virtual Reality Augmented Evacuation System Application

6.2.1 Introduction

A testbed Emergency Evacuation system has been completed with all necessary hardware and software integrated. The system is developed for N1-B3 of Nanyang Technological University.

6.2.2 System Components

As 3D view is necessary for real evacuation tasks, some Augmented Reality (AR) hardware have been integrated into our system. The AR hardware we use in our current research is shown in Figure 6.8. The 3D scene on the head mounted display (HMD) changes according to the mobile user’s location and orientation, the information of which is collected by trackers. This is done by the AR engine implemented. The mobile user can issue path finding and/or evacuation profile commands to the system by mouse or keyboard. The AR engine, which is installed on the mobile computer or PDA taken by the user, will do corresponding computation and give the result on the HMD.

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

Figure 6.8 System Equipment

The system components' integration is illustrated in Figure 6.9.

Figure 6.9 AR System

With the system equipped, users will have no worry about getting lost or wasting time on finding a good-enough path to an exit door inside an unfamiliar building even the disaster has destroyed all the lighting or the real environment is invisible due to smoke and/or fire. The 3D view on the display device will give enough augmenting

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

information for them.

Current hardwares of our test-bed 3D system are listed in Table 6.1.

Table 6.1 Hardware List

Location Tracker	Ekahau Wi-Fi location tracker
Orientation Tracker	Inertia Cube2
Mobile Computer	Acer TravelMate
HMD	EyetoP LCD

When location trackers' accuracy can be as high as 3-5 cm, the system is expected to work perfectly for mobile users. But because the wireless network in our campus is not installed as expected by Ekahau's specification, our system accuracy level is around 10-15 meters. For the sake of budget limitation, our testbed system can only equipped with the location tracker with this accuracy level.

Inertia cube2 orientation tracker can give degrees of angles about three axes. Our system makes use of the angle about the z axis, whose direction is vertically upwards from the ground. This angle can define the view orientation of the mobile user in x-y plane, and it's good enough for evacuation tasks.

The potential users of the system and the embedded algorithms are emergency rescue staff, like fire-fighters or soldiers. But it can also be built for other tasks, especially when the tasks are location sensitive.

Before the flow assignment computation, the path finding computations should be

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

done first to find the shortest paths from the source to all the exits on the same floor. After the shortest paths have been found and the length of each shortest path has been found in terms of traversal time, they will be treated as the input for the flow assignment computation.

Before our system can be run for a large indoor area, this area must be converted to a network data structure, and all the locations in the concerned area can be mapped to some dynamic or dummy nodes on the network arcs. The next section will describe the detailed method in this aspect.

6.2.3 N1-B3 Floor Plan and Its Network Conversion

Our testbed system has been tried out using the B3 floor of Block N1 (N1-B3) on the Nanyang Technological University (NTU) campus, which is a typical large indoor area of a campus building and all the exit doors are NDs. A simplified floor layout graph is shown in Figure 6.10.

Figure 6.10 Floor Layout of N1-B3

There are four exit doors, whose maximum width is 1.2m. The minimum width of passageway area is 2.0m. This implies that the flow assignment problem can be treated as ND problem. Another assumption of our system is that the source location

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

of evacuees should be indicated as in passageway area for the convenience of the location tracker.

According to the topology of the floor layout, this large indoor area can be contracted to be a network shown in Figure 6.11.

Figure 6.11 Network of N1-B3

The network in Figure 6.11 is a *spatial map diagram* (introduced in Section 2.3.3.2) of the area and the arcs are the middle lines of the corresponding passageways. We call these arcs *rails* later on. As the evacuees may be at any location in the passageway area, we have to use some method to map the real passageway location to a close location on the rails. The method we use in our system is to divide the whole passageway area into a bunch of regions. Then the location in each region will be treated according to the specific feature of the region. The region division is shown in Figure 6.12.

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

Figure 6.12 Region Division of N1-B3

The method to map the location in each region to a dynamic or dummy node on the rails of the network in Figure 6.11 is shown in Table 6.2.

Table 6.2 Regions Mapping of N1-B3

Location in region	is mapped to
1	dummy node 1
2	dummy node 2
3	dummy node 3
4	dummy node 4
5	dummy node 5
6	dummy node 6
7	dummy node 7
8	dummy node 8
9	the closest dynamic node on rail 1-5
10	dummy node 9
11	dummy node 10
12	the closest dynamic node on rail 4-8
13	the closest dynamic node on rail 1-2
14	the closest dynamic node on rail 2-3
15	the closest dynamic node on rail 3-4
16	the closest dynamic node on rail 5-6
17	the closest dynamic node on rail 6-7
18	the closest dynamic node on rail 7-14
19	the closest dynamic node on rail 14-8

With this mapping method, when the x-y location coordinates are collected from the location tracker in real time, the system can determine in which region the location is. After that, that location can be converted to a dynamic node on the rails of the network. Therefore, the shortest path algorithm and the flow division and dispatching

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

algorithm can be implemented.

The 3D VR environment is drawn with the OpenGL library in Visual C++ 6.0. The floor is drawn in a 3D space, whose Cartesian system is shown in Figure 6.13.

Figure 6.13 the Cartesian System in OpenGL

In this Cartesian system, the lower left corner of the floor is (1, 1) and the upper right corner is (182.4, 21). The unit of the Cartesian system is meters. The coordinates of all relevant objects on the floor are stored in an AutoCAD file, whose softcopy is appended to this thesis.

In next section, a simulation case demonstrating how the system works is shown.

6.2.4 System Demonstration

Let us demonstrate the system with a real evacuation process. If 300 people are waiting to be evacuated and three firefighters equipped with our system have found them, one of the firefighters can issue a flow assignment request to the system. Suppose the location of the firefighters is (30, 1.5). For convenience of demonstration, we manually input this data to the program to execute a k-limited flow control computation. Click the menu item, “No-k-Limited Flow Control (XY Mode)”, under

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

the view menu. The input dialog box may appear as in Figure 6.14.

The location will be mapped to the closest dynamic node on rail 5-6. Then further calculation will be based on this dynamic node.

If the walking speed of the evacuees are given as some behavior research suggests or just given according to some experience data, the system will give the flow division profile.

Figure 6.14 Input Dialog for Testbed System

In the dialog box, the location is input manually by the firefighter. There is another input mode, in which the location information is read from the location tracker and is not input manually (tracker mode). In this demo case, as there are only 3 firefighters, the flow control problem is 3-limited. So the number three is input manually to the system. The walking speed is arbitrarily given in the current system.

After the firefighter press the “OK” button, the viewport of the window will snap to where the firefighter stand and will head to the direction indicated by the real time data from the orientation tracker. The calculation result is given after this snapping and the result display interface will display the optimal evacuation profile textually

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

and graphically. The evacuation routes will be given in different colors. The estimated evacuation time based on the given walking speed will also be given .They are shown in Figure 6.15.

Figure 6.15 System Output Window

The textual instructions on groupings for firefighters are given in the upper left corner of the window. The two evacuation routes are shown in red and blue colors respectively. The white sphere indicates the dynamic node corresponding to the firefighter's real location. The whole calculation can be done in the time less than one second, which is good enough for a real time system.

Users can also change the background color by click the menu item, "White or Black Background" under the view menu to make the black-and-white printout of the screen more clear. The output window with white background is shown in Figure 6.16.

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

Figure 6.16 System Output Window with White Background

The user can also change to a bird's eye view by click the menu item, "Bird's Eye View" under the view menu. The bird's eye view for Figure 6.16 is shown in Figure 6.17.

Figure 6.17 Bird's Eye View

Chapter 6 Virtual Reality Aumented Evacuation System for Large Indoor Areas

You may also notice that the DCs of the doors are arbitrarily given before the calculation starts. Actually the DCs of the doors of different widths can be given by doing some behavior experiments. These experiments can be done in future research.

The location input can be done in three modes. The first is to manually give GN information. The second is to manually give XY information. And the third is to let the system automatically take the XY information from the location tracker. In this demo case, we use the second mode. For each of these three modes, k and non-k limited flow control algorithm can be executed by click respective menu items under the view menu.

When giving input, the user can also decide whether to use k-limited flow assignment algorithm or non-k-limited flow assignment algorithm. The later algorithm can be used when there are enough numbers of firefighters equipped with our system.

Currently the shortest path algorithm used in our system is Dijkstra's due to the reasons described in section 4.8.2. The IRS algorithms can be a good alternative when there is no network changing.

The whole system is coded in Visual C++ 6.0. A soft copy of all the source code is appended to this thesis.

6.3 Summary

The Takashimaya network is coded to test the suitability of the flow control algorithms. A complete testbed system is described as well. The method of underlying network conversion is introduced. The flow control algorithms described in Chapter 5 have been embedded into this system. And a simulated evacuation scene is given to demonstrate how the system works.

Chapter 7 Conclusions and Future Work

7.1 Conclusions

When the Virtual Reality (VR) technology is applied to a real time system, some challenges may come from the characteristics of the system and the VR technology itself. This research is to develop shortest path finding and flow control algorithms for indoor emergency evacuation.

When path finding and flow control functions are only implemented for simulation, computing efficiency is not the main concern, because there is no computing speed requirement for it. But if these functions are to be provided for a real time evacuation system, they are expected to be able to give computation results in a very short time. In a system, which requires intensive computation, like a VR emergency system, additional computation tasks have to be implemented in a very efficient way. The word, *efficient*, means the system can complete the real time computing and present the result to the user in a short time in order not to compromise the evacuation tasks in terms of time.

In order to find fast path finding algorithms, this thesis has analyzed common indoor geometric topologies first, and then found that most of them are highly regular structures or the combinations of highly regular structures. Therefore, an efficient routing technique named Interval Routing Scheme, which is usually used in highly regular computer networks, was introduced. Some algorithms were developed to make sure the IRS techniques could work for dynamic source and target locations. A series of experiments have been conducted to compare the performance of I-IRS and Dijkstra's algorithms.

Chapter 7 Conclusions and Future Work

Sometimes, the path finding source and target may be located in different buildings or in a building with more complicated structures. An algorithm combining IRS and the classic Dijkstra's algorithm was developed to deal with this situation. The working procedure of this algorithm has been provided in section 4.9. Another series of frequently appearing topologies, 2-layer 1-IRS topologies are also analyzed. A non-Dijkstra involved IRS algorithm is provided for this series of topologies, which can calculate the shortest path between any two locations on the network arcs in a much lower computation complexity than Dijkstra's algorithm.

With the fast IRS path finding algorithms developed, other premium functions, such as real time road guide systems can also take advantage of the interval routing information stored for each arc.

In Chapter 5, we present our observations on inside structures of modern buildings. On these observations, we may convert a more complex flow control problem into a multiple-ND problem, which can be solved with a fast real time method. A variation of this multiple-ND problem, the k-limited flow control problem, is also discussed and an enumerating and comparing algorithm is provided. In Chapter 6, a demonstration program, which applying Dijkstra's shortest path algorithm and flow control methods to Takashimaya Shopping Center of Singapore, is described first. Then, with all VR hardware and software integrated, a testbed evacuation system based on the B3 floor of Block N1 on the Nanyang Technological University campus is described. The flow control algorithms are embedded as real time tools for rescue team members to divide and dispatch large group of evacuees to different exit doors to achieve a shortest evacuation time. Although the system is still in its testing and refining stage, it has been able to work for some simulated evacuation tasks.

Chapter 7 Conclusions and Future Work

7.2 Academic Contributions of this Research

There are five main contributions from this research till now. They are:

(1) A fast routing technique, named Interval Routing Scheme (IRS), is introduced from abstract static computer networks into the field of Euclidean dynamic networks. 1-IRS topologies have been analyzed systematically. New algorithms for 2D grids, 3D grids, trees and rings were developed and extended the static IRS applications to the dynamic networks. It is the first attempt to apply routing techniques in computer networks to physical Euclidean networks.

(2) An IRS and Dijkstra combined algorithm was developed to incorporate complex intra- or inter-building topologies. It can become an efficient substitute for traditional pure Dijkstra's algorithm for our evacuation system. And in some situations, as analyzed in section 4.9.3, it can perform much better than pure Dijkstra's algorithm.

(3) 2-layer 1-IRS algorithm is given in the form of a systematic procedure on the basis of 1-IRS dynamic algorithms. For 2-layer 1-IRS topologies, this algorithm can completely replace Dijkstra's shortest path algorithm. With the presentation of the IRS information for each arc, some other useful tools, like real time road guide, can be developed.

(4) To meet the requirement on calculation speed, we have simplified the complex flow control problem to a multi-ND problem, to which a fast flow assignment algorithm can be applied, provided some reasonable assumptions. This fast flow algorithm and one of its variations can provide very useful flow control suggestions for rescue team members. And the testbed system has demonstrated its efficiency in real world evacuation tasks.

Chapter 7 Conclusions and Future Work

(5) The testbed system is the first system taking advantage of wireless location tracking technology to conduct indoor real time emergency evacuation tasks. Our work may encourage more academic effort in this area.

When applying the IRS algorithms, we have to remember that only the topologies with certain features can be the subject. Before applying IRS algorithms, the nodes and arcs of the network must be labeled with the certain methods, which raise more work than implementing Dijkstra's shortest path algorithm. When the network is changed in emergency condition, part of the IRS information may be useless. This is another point which should be remembered.

Computer networks and buildings internal structures often take similar highly regular structures. The reason may be that they are both artificial products of human beings, who prefer orderliness, standardization and easiness of replication especially for engineers. Frequent appearances of highly regular topologies in network designs of a variety of engineering fields may reflect this preference broadly existing in engineering area. That may be the original momentum driving our research to introduce the routing strategy in computer networks for the use in the Euclidean shortest path finding area.

As our evacuation system is on a VR platform, which has integrated state-of-the-art technologies, like wireless location trackers and HMDs, the flow control algorithm can be implemented as a practical tool for real tasks. This is an important trend in modern intelligent transportation systems. Our micro indoor evacuation system is one of these systems.

Chapter 7 Conclusions and Future Work

7.3 Future Research

It's very likely that k-IRS can also be extended into Euclidean networks to deal with shortest path finding problem for more complex networks. Some other path finding tools based on routing strategy can also be developed.

As for flow control, in this thesis, we are dealing with the buildings with the ND feature. With this feature, the flow control algorithm can make significant simplification on DCs of all passageways; therefore, the computation complexity is largely lowered. When the subject building is not ND featured, new fast algorithm must be developed for real time systems.

References

References

- [1] A. Rekimoto and M. Saitoh. Metronaut: A wearable computer with sensing and global communication capabilities. In *Proce.ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 116-122, Cambridge, MA, October 13-14 1997.
- [2] S. Long, D. Aust, G. Abowd, and C. Atkenson. Cyberguide: Prototyping context aware mobile applications. In *CHI '96 Conference Companion*, pages 293-294, April 1996.
- [3] J. Loomis, R. Golledge, and R. Klatzky. Personal guidance system for the visually impaired using GPS, GIS and VR technologies. In *Proc. Conf. on Virtual Reality and Persons with Disabilities*, Millbrae, CA, June 17-18 1993.
- [4] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Proc. ISWC '97 (First Int. Symp. On Wearable Computers)*, pages 74-81, Cambridge, MA, October 13-14 1997.
- [5] B. Thomas, V. Demczuk, W. Piekarski, D. Hepworth, and B. Gunther. A wearable computer system with augmented reality to support terrestrial navigation. In *Proc. ISWC '98 (Second Int. Symp. on Wearable Compters)*, pages 168-171, Pittsburgh, PA, October 19-20 1998.
- [6] B. Jang, J. Kim, H. Kim, and D. Kim. And outdoor augmented reality system for GIS applications. In Y. Ohta and H. Tamura, editors, *Mixed Reality, Merging Real and Virtual Worlds*, pages 391-399. Ohmsha/springer, Tokyo/New York, 1999.
- [7] T. Hollerer, S. Feiner, and J. Pavlik. Situated documentaries: Embedding

References

multimedia presentations in the real world. In *Proc. ISWC '99 (Third Int. Symp. on Wearable Computers)*, San Francisco, CA, October 18-19 1999.

[8] A. Harter, A. Hopper, P. Steggles, . Ward, and P. Webster. The anatomy of a context-aware application. In *Proc. of the Fifth ACM/IEEE Int. Conf. on Mobile computing and Networking (MobiCom)*, pages 59-68, Seattle, WA, Aug. 1999.

[9] I. Getting. The global positioning system. *IEEE Spectrum*, 30(12):36-47, Dec. 1993.

[10] T. Starner, D. Kirsch, and S. Assefa. The locust swarm: An environmentally-powered, networkless location and messaging system. In *Proc. ISWC '97 (First Int. Symp. On Wearable Computers)*, pages 169-170, Cambridge, MA, October 13-14 1997.

[11] A. Butz, J. Baus, and A. Kruger. Augmenting buildings with infrared information. In *Proceedings of the International Symposium on Augmented Reality ISAR 2000*, pages 93-96. IEEE Computer Society Press, Oct 5-6 2000.

[12] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, and K. Tachibana. Virtual object manipulation on a table-top ar environment. In *Proceedings of the International Symposium on Augmented Reality ISAR 2000*, pages 111-119. IEEE Computer Society Press, Oct 5-6 2000.

[13] F. Raab, E. Blood, T. Steiner, and R. Jones. Magnetic position and orientation tracking system. *IEEE Trans. On Aerospace and Electronic Systems*, AES-15(5): 709-718, September 1979.

[14] 3rdTech. <http://www.3rdtech.com/HiBall.htm>, 2001

References

- [15] T. Hollerer, D. Hallaway, N. Tinna and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *AIMS'01: Second Int. Workshop on Artificial Intelligence in Mobile Systems*, Seattle, WA, Aug. 4, 2001, pp.31-37.
- [16] <http://www.ekahau.com/>, 2004
- [17] <http://www.isense.com/>, 2004
- [18] <http://www.sense8.com/>, 2004
- [20] G.C. Burdea and P. Coiffet. *Virtua Reality Technology, second edition*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2003.
- [21] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.* 7: 175-179, 1978.
- [22] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6: 594-606, 1977.
- [23] D. Avis and G. T. Toussaint. An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recogn.*, 13:395-398. 1981.
- [24] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 339-349, 1982.
- [25] B. Chazelle. Approximation and Decomposition of Shapes. In [26], page 145-185, 1987.
- [26] J. T. Schwartz, and C. K. Yap. *Algorithmic and Geometric Aspects of Robotics*,

References

Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[27] J. C. Lamtombe. *Robot Motion Planning*, Luwer Academic Publishers, 1991.

[28] N. J. Nilsson. A mobile automation: an application of artificial intelligence techniques. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington D. C., 509-520, 1969.

[29] D. T. Lee. Proximity and reachability in the plane. Report R831, Dept. Elect. Engrg., University of Illinois, Urbana, IL, 1978.

[30] T. Asano, T. Sano, L. J. Guibas, J. Hersberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49-63, 1986.

[31] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, 20:167-171, 1985.

[32] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20:888-910, 1991.

[33] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Inform. Process. Lett.*, 23:71-76, 1986.

[34] G. L. Dirchlet. Uber die reduction der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *J. Reine Angew. Math.*, 40:209-227, 1850.

[35] G. M. Voronoi. Nouvelles applications des parameters continues a la theorie des formes quadratiques. premier Memoire: Sur quelques proprietes des formes quadratiques positives parfaits. *J. Reine Angew. Math.*, 133:97-178, 1907.

References

- [36] G. M. Voronoi. Nouvelles applications des parametres continues a la theorie des formes quadratiques. deuxieme Memoire: Recherches sur les paralleloedres primitives. *J. Reine Angew. Math.*, 134:198-287, 1908.
- [37] M. I. Shamos and D. Hoey. Closest-point problems. In *Proc. 16th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 151-162, 1975.
- [38] S. J. Fortune. A sweeping algorithm for Voronoi diagrams. *Algorithmica*, 2: 153-174, 1987.
- [39] L. J. Guibas and J. Stolfi. Ruler, compass and computer: the design and anlysis of geometric algorithms. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD. NATO ASI Series F*, vol. 40, pages 111-165. Springer-Verlag, 1988.
- [40] F. Aurenhammer. Voronoi diagrams: properties, algorithms and applications. *SIAM J. Comput.*, 16:78-96, 1987.
- [41] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, U.K., 1992.
- [42] V. Klee. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34:75-80, 1980.
- [43] J. Van Leeuwen, R.B. Tan, Interval routing, *The Computer Journal* , 30 (1987) 298-307
- [44] Dijkstra EW. A note on two problems in connection with graphs, *Numerische Mathematic* 1959;1:269-71.
- [45] C. Gavoille, A survey on interval routing, *Theoret. Comput. Sci.* 245 (2000)

References

217-253

- [46] N. Santoro, R. Khatib, Labeling and implicit routing in networks, *The Computer Journal*, 28 (1985) 5-8
- [47] J. S. B. Mitchell, C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the Association for Computing Machinery*, Vol 38. No 1 January 1991. pp 18-73
- [48] J. Hershberger, S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.* Vol 28, No. 6, pp. 2215-2256, 1999
- [49] S. Rajasekaran, S. Ramaswami. Optimal mesh algorithms for the Voronoi diagram of line segments and motion planning in the plane. *Journal of parallel and distributed computing*, 26, 99-115 (1995).
- [50] R. Khosravi, M. Ghodsi. Shortest paths in simple polygons with polygon-meet constraints. *Information Processing Letters* 91(2004) 171-176.
- [51] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.* Vol. 28, No. 2, pp 652-673, 1998.
- [52] J. A. Azevedo, M. E. Santos Costa, J. J. Silvestre Madeira, E. Vieira Martins. An algorithm for the ranking of shortest paths. *European Journal of Operation Research* 69 (1993) 97-106.
- [53] E. Vieira Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operation Research* 18 (1984) 123-130.
- [54] C. Bajaj, T. T. Moh. Generalized unfoldings for shortest paths in Euclidean 3-space. *Int. J. Robot. Res.*1987.

References

- [55] S. T. Jones. Solving problems involving variable terrain Part I: A general algorithm. *Byte* 5, 2 (Feb. 1980), 58-68.
- [56] J. S. B. Mitchell, C. H. Papadimitriou. The weighted region problem. *Proceedings of the 3rd Annual ACM Conference on Computational Geometry*, 1987, pp 30-38.
- [57] D. Z. Chen, K. S. Klenk, H. T. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *Proceedings of the 11th Computational Geometry*, page 370-379, 1995.
- [58] F. S. Hillier, G. J. Lieberman. Introduction to operations research. McGraw-Hill, Inc. 1995
- [59] J. Wang, T. Kaempke. Shortest route information in distributed systems. *Computer & Operations Research* 31 (2004) 1621 – 1633.
- [60] F. J. Radermacher, M. Schaal, S. Schnittger. Durchgangige Elektronische Fahrplaninformation – DELFI, *Proceedings of 4th world Congress on Intelligent Transportation Systems ITS*, Berlin, FAW, Research Institute of Applied Artificial Intelligence, Helmholtzstr. 16, 89081 Ulm, Germany, 1997.
- [61] The EU-SPIRIT API approach, FAW, Research Institute of Applied Artificial Intelligence, Helmholtzstr. 16, 89081 ULm, Germany, 1999.
- [62] S. Schnittger. Fast path, FAW, Research Institute of Applied Artificial Intelligence, Helmholtzstr. 16, 89081 ULm, Germany.
- [63] T. Kaempke, M. Schaal. Distributed generation of fastest paths. *Proceedings of Conference on Parallel and Distributed Computing and Systems PDCS'98*, Las Vegas,

References

1998. p. 172-7.

[64] J. Van Leeuwen, R.B. Tan. Interval routing. *The Computer Journal* , 30 (1987) 298-307.

[65] C. Gavoille. A survey on interval routing. *Theoret. Comput. Sci.* 245 (2000) 217-253.

[66] N. Santoro, R. Khatib. Labeling and implicit routing in networks. *The Computer Journal*, 28 (1985) 5-8.

[67] N. Santoro, R. Khatib. Routing without routing tables. *Technical Report SCS-TR-6* School of Computer Science, Carleton University, Ottawa, 1982.

[68] D. May, P. Thompson. Transputers and Routers: Components for concurrent machines, INMOS Ltd., 1990.

[69] B. Zerrouk, V. Reibaldi, F. Potter, A. Greiner, D. Anne. RCube, a gigabit serial links low latency adaptive router, in the *Records of the IEEE Hot Interconnects IV*, Palo Alto CA, U.S.A, August 1996.

[70] J. van Leeuwen, R.B. Tan. Compact routing methods: a survey, in: P. Flocchini, B. Mans, N. Santoro(Eds.), *1st Internat. Coll. on Structural Information & Communication Complexity (SIROCCO)*, Carleton University Press, May 1994, pp. 99{110.

[71] H. Buhrman, J.-H. Hoepman, P. Vitanyi. Optimal routing tables, in: *15th Ann. ACM Symp. on Principles of Distributed Computing (PODC)*, ACM Press, New York, May, 1996, pp. 134-142.

References

- [72] P. Fraigniaud, C. Gavoille. Interval routing schemes. *Algorithmica* 21 (1988) 155-182.
- [73] T. Eilam, S. Moran, S. Zaks. A lower bound for linear interval routing. in: . O. Babao_glu, K. Marzullo(Eds.), *10th Internat. Workshop on Distributed Algorithms (WDAG), Lecture Notes in Computer Science*, vol. 1151, Springer, Berlin, October 1996, pp. 191-205.
- [74] S.S.H. Tse, F.C.M. Lau. An optimal lower bound for interval routing in general networks, in: D. Krizanc, P. Widmayer (Eds.), *4th Internat. Coll. on Structural Information & Communication Complexity (SIROCCO)*, Carleton Scienti_c, July 1997, pp. 112-124.
- [75] M. Flammini, E. Nardelli. On the path length in interval routing schemes, manuscript, 1996.
- [76] B. Zerrouk, S. Tricot, B. Rottembourg, L. Patnaik, Proper linear interval routing schemes. Technical Report N 94-29, IBP-MASI, October 1994.
- [77] I. Sakho, L. Mugwaneza, Y. Langue. Routing with compact tables: interval labelling scheme for generalized meshes. in: C. Girault (Ed.), *IFIP Transactions: Applications in Parallel and Distributed Computing (APDC)*, Elsevier Science B.V., North-Holland, Amsterdam, 1994, pp. 297-308.
- [78] F. S. Hillier, G. J. Lieberman. *Introduction to Operations Research*. McGraw-Hill, Inc. (1995)
- [79] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, Canmbridges (1985).

References

- [80] G. Chartrand. *Introductory Graph Theory*. Dover, New York (1985).
- [81] E. F. Moore. The shortest path through a maze. *Proc. International Symposium of Switching Theory*. Harvard University Press, Cambridge (1959), 285-292.
- [82] W. W. White. Dynamic Transshipment Networks: an algorithm and its application to the distribution of empty containers. *Networks*, 1972;2:211-36.
- [83] T. M. Kisko and R. L. Francis. EVACNET+: a computer program to determine optimal building evacuation plans. *Fire Safety Journal*, 9 (1985) 211-220.
- [84] A. K. Gupta, P. K. Yadav. SAFE-R: a new model to study the evacuation profile of a building. *Fire Safety Journal* 39 (2004) 539-556.
- [85] FI. Stahl. BFIREs-II: a behavior based computer simulation of emergency egress during fires. *Fire Technol.* 1982; 18:49 .65.
- [86] N. L. Biggs. *Discrete Mathematics*. Oxford University Press, 2002.
- [87] Chen P. H. & Feng F. Path finding for real-time evacuation systems for large indoor areas. *The Proceedings of 22nd International Symposium on Automation and Robotics in Construction*. ISARC, 2005.
- [88] Chen P. H. & Feng F. Path Finding in 2-layer 1-IRS Topologies for Real-Time Evacuation Systems. *The Proceedings of New Technologies for Urban Safety of Mega Cities in Asia*. USMCA, 2005.
- [89] Fahy, R.F. EXIT 89: an evacuation model for high-rise buildings. In *Fire Safety Science – Proceedings of the 3rd International Symposium*, ed. G. Cox & B. Langford. Elsevier Science, London, 1991.

References

- [90] Galea, E.R. & Perez Galparsoro, J.M., A computer based simulation model for the prediction of evacuation from mass transport vehicles, *Fire Safety J.* 22 (1994) 341-66.
- [91] Ketchell, N., Webber, D.M. etc. The EGRESS code for human movement and behavior in emergency evacuations. In *Engineering for Crowd Safety – Proceedings of the International Conference for Crowd Safety*, ed. R.A. Smith & J. F. Dickie. Elsevier Applied Science, Amsterdam, 1993.
- [92] Thompson P.A. & Marchant E.W., A computer model for evacuation of large building populations. *Fire Safety J.* 24 (1995) 131-148.

Publications

Publications

Conference Papers:

[1] Chen P. H. & Feng F. Path finding for real-time evacuation systems for large indoor areas. *The Proceedings of 22nd International Symposium on Automation and Robotics in Construction*. ISARC, 2005.

[2] Chen P. H. & Feng F. Path Finding in 2-layer 1-IRS Topologies for Real-Time Evacuation Systems. *The Proceedings of New Technologies for Urban Safety of Mega Cities in Asia*. USMCA, 2005.

Appendix: Algorithms in C++ Style Pseudo-Codes

Appendix: Algorithms in C++ Style Pseudo-Codes

Algorithm in section 4.3.2:

Class Gridnode {Arc [] arclist; int nodelb; int x; int y}

/ Arclist is an array of the arcs starting from the node. Nodelb is the 1-LIRS label for the node. x and y are the coordinates of the node in the grid*/*

Class Gridarc {Gridnode stnode, ennode; float arclength; int sint, eint;}

/ stnode and ennode are the start and end nodes of the arc. arclength is the Euclidean length of the arc; sint and eint are respectively the first and second number of the linear interval label. */*

Program pathstatic (node s, node t)

{ Gridarc arc; Arc [] result; Gridnode ndtemp; ndtemp=s;

While (ndtemp <> t) do

{for each arc in ndtemp.arclist

if (t.nodelb >= arc.sint and t.nodelb <= arc.eint) then

{result.add(arc); ndtemp=arc.ennode; break;

} endif;

endfor;}

return result [];

Algorithm in section 4.3.3:

Program Findpath2d (S, T, G)

// S, T are the source and G is the network consisting of nodes and arcs.

{ Arc [] result; Gridnode ndtemp;

if (S and T are both static) then result=pathstatic(S,T); endif; // static path finding

if (S is dynamic and T is static) then // case 1

Appendix: Algorithms in C++ Style Pseudo-Codes

```

{S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; // function returning an array of two GNs of S in G
if ((S1.x+S1.y) > (S2.x+S2.y)) Then {ndtemp=S1; S1=S2;, S2=ndtemp;} endif;

//Let S1 be the left or upper GN
if (S1.x = S2.x) then // S on vertical segment
    {if (T.y <= S1.y) then result=pathstatic(S1,T); else result=pathstatic(S2,T); endif} endif;
if (S1.y = S2.y) then // S on horizontal segment
    {if (T.x <= S1.x) then result=pathstatic(S1,T); else result=pathstatic(S2,T); endif } endif;
} endif;

if (S is static and T is dynamic) then // case 2
    {T1=findgn(T, G)[0]; T2=findgn(T,G)[1]; // function returning an array of two GNs of S in G
    if ((T1.x+T1.y) > (T2.x+T2.y)) Then {ndtemp=T1; T1=T2; T2=ndtemp;} endif;

    //Let T1 be the left or upper GN
    if (T1.x = T2.x) then // T on vertical segment
        {if (S.y <= T1.y) then result=pathstatic(S,T1); else result=pathstatic(S,T2); endif} endif;
    if (T1.y = T2.y) then // T on horizontal segment
        {if (S.x <= T1.x) then result=pathstatic(S,T1); else result=pathstatic(S,T2); endif } endif;
    } endif;

If (Both S and T are Dynamic) then // case 3: dynamic path finding
    {S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; T1=findgn(T, G)[0]; T2=findgn(T,G)[1];
    if ((S1.x+S1.y) > (S2.x+S2.y)) Then {ndtemp=S1; S1=S2;, S2=ndtemp;} endif;

    //Let S1 be the left or upper GN
    if ((T1.x+T1.y) > (T2.x+T2.y)) Then {ndtemp=T1; T1=T2; T2=ndtemp;} endif;

    //Let T1 be the left or upper GN

    if (S1.x=T1.x and S1.y=T1.y and S2.x=T2.x and S2.y=T2.y ) then result[]=0; endif;

    // S and T on same segment

```

Appendix: Algorithms in C++ Style Pseudo-Codes

// S and T are on different horizontal segments

if ($S1.y = S2.y$ and $T1.y = T2.y$ and $T2.x \leq S1.x$) then result=pathstatic($S1, T2$); endif;

if ($S1.y = S2.y$ and $T1.y = T2.y$ and $T1.x \geq S2.x$) then result=pathstatic($S2, T1$); endif;

if ($S1.y = S2.y$ and $T1.y = T2.y$ and $T1.x = S1.x$ and $T1.y < S1.y$) then

 {if ($(|SS1| + |TT1|) \leq (|SS2| + |TT2|)$) then result=pathstatic($S1, T1$);

 else result= pathstatic($S2, T2$);endif;} endif;

// S and T are on different vertical segments

if ($S1.x = S2.x$ and $T1.x = T2.x$ and $T2.y \leq S1.y$) then result=pathstatic($S1, T2$); endif;

if ($S1.x = S2.x$ and $T1.x = T2.x$ and $T1.y \geq S2.y$) then result=pathstatic($S2, T1$); endif;

if ($S1.x = S2.x$ and $T1.x = T2.x$ and $T1.y = S1.y$ and $T1.x < S1.x$) then

 {if ($(|SS1| + |TT1|) \leq (|SS2| + |TT2|)$) then result=pathstatic($S1, T1$);

 else result= pathstatic($S2, T2$);endif;} endif;

// S on vertical segment and T on horizontal segment

if ($S1.x = S2.x$ and $T1.y = T2.y$ and $T1.y \leq S1.y$ and $T2.x \leq S1.x$) then result=pathstatic($S1, T2$);
endif;

if ($S1.x = S2.x$ and $T1.y = T2.y$ and $T1.y \leq S1.y$ and $T1.x \geq S1.x$) then result=pathstatic($S1, T1$);
endif;

if ($S1.x = S2.x$ and $T1.y = T2.y$ and $T1.y \geq S1.y$ and $T2.x \leq S1.x$) then result=pathstatic($S2, T2$);
endif;

if ($S1.x = S2.x$ and $T1.y = T2.y$ and $T1.y \leq S1.y$ and $T1.x \geq S1.x$) then result=pathstatic($S1, T1$);
endif;

// S on horizontal segment and T on vertical segment

if ($S1.y = S2.y$ and $T1.x = T2.x$ and $T1.x \leq S1.x$ and $T2.y \leq S1.y$) then result=pathstatic($S1, T2$);
endif;

if ($S1.y = S2.y$ and $T1.x = T2.x$ and $T1.x \leq S1.x$ and $T1.y \geq S1.y$) then result=pathstatic($S1, T1$);
endif;

if ($S1.y = S2.y$ and $T1.x = T2.x$ and $T1.x \geq S1.x$ and $T2.y \leq S1.y$) then result=pathstatic($S2, T2$);

Appendix: Algorithms in C++ Style Pseudo-Codes

```

endif;

if (S1.y = S2.y and T1.x = T2.x and T1.x >= S1.x and T1.y >= S1.y) then result=pathstatic(S1,T1);

endif;

} endif;

return result[]; } // end of the program findpath2d()

```

Algorithm in section 4.4.3

```

Class Gridnode {Arc [] arclist; int nodelb; int x; int y; int z}

/* Arclist is an array of the arcs starting from the node. Nodelb is the 1-LIRS label for the node. x and y
are the coordinates of the node in the grid*/

Program Findpath3d (S, T, G)

// S, T are the source and G is the network consisting of nodes and arcs.

{Arc [] result; Gridnode ndtemp;

if (S or T is static) then result=pathstatic(S, T);endif; // static and semi-static cases

if (both of S and T are dynamic) then // dynamic cases

    {S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; T1=findgn(T, G)[0]; T2=findgn(T,G)[1];

    if ((S1.x+S1.y) > (S2.x+S2.y)) Then {ndtemp=S1; S1=S2; S2=ndtemp;} endif;

    if ((T1.x+T1.y) > (T2.x+T2.y)) Then {ndtemp=T1; T1=T2; T2=ndtemp;} endif;

    if (S1.x=T1.x and S1.y=T1.y and S2.x=T2.x and S2.y=T2.y) then

        {if (|SS1|+|TT1| <= |SS2|+|TT2|) then result=pathstatic(S1, T1);

        else result= pathstatic(S2, T2); endif;}

    else result=findpath2d(S, T, G); endif;} endif;

return result[];}

```

Algorithm in section 4.5.1

```

Program treestatic (node s, node t)

{ Gridarc arc; Arc [] result; Gridnode ndtemp; ndtemp=s;

```

Appendix: Algorithms in C++ Style Pseudo-Codes

```

While (ndtemp <> t) do
  {for each arc in ndtemp.arclist
    {if (arc.sint<=arc.eint and t.nodelb >= arc.sint and t.nodelb <= arc.eint) then
      {result.add(arc); ndtemp=arc.ennode; break;} endif;
    if (arc.sint>arc.eint and ( t.nodelb >= arc.sint or t.nodelb <= arc.eint)) then
      {result.add(arc); ndtemp=arc.ennode; break;} endif;} endfor;}
  return result [];}

```

Algorithm in section 4.5.2

Program treedynamic(S,T,G) // S, T are source and target nodes

```

{ Arc [] result;
if (S and T are both static) then result=treestatic(S,T); endif; // static path finding
if (S is dynamic and T is static) then // case 1
  {S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; // function returning an array of two GNs of S in G
  Gridarc arc;
  For each arc in S1.arclist
    {if (arc.sint<=arc.eint and (S2.nodelb>=arc.sint and S2.nodelb<=arc.eint)) and
      (T.nodelb>=arc.sint and T.nodelb<=arc.eint)) then result=treestatic(S2,T); break;} endif;
    if (arc.sint>=arc.eint and (S2.nodelb>=arc.sint or S2.nodelb<=arc.eint)) and
      (T.nodelb>=arc.sint or T.nodelb<=arc.eint)) then result=treestatic(S2,T); break; } endif; endfor;
    if result=null then result=treestatic(S1,T);endif;
  } endif;

if (S is static and T is dynamic) then // case 2
  {T1=findgn(T, G)[0]; T2=findgn(T,G)[1]; // function returning an array of two GNs of S in G
  Gridarc arc;
  For each arc in T1.arclist
    {if (arc.sint<=arc.eint and (T2.nodelb>=arc.sint and T2.nodelb<=arc.eint)) and

```


Appendix: Algorithms in C++ Style Pseudo-Codes

```

(S.nodelb>=arc.sint and S.nodelb<=arc.eint)) then result=treestatic(S,T2); break;} endif;

if (arc.sint>=arc.eint and (T2.nodelb>=arc.sint or T2.nodelb<=arc.eint)) and

(S.nodelb>=arc.sint or S.nodelb<=arc.eint)) then result=treestatic(S,T2); break; } endif; endfor;

if result=null then result=treestatic(S,T1);endif;

} endif;

```

If (Both S and T are Dynamic) then // case 3: dynamic path finding

```

{S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; T1=findgn(T, G)[0]; T2=findgn(T,G)[1];

Treenode tempS, tempT; Gridarc arc;

For each arc in S1.arclist

{if (arc.sint<=arc.eint and (S2.nodelb>=arc.sint and S2.nodelb<=arc.eint)) and

(T1.nodelb>=arc.sint and T1.nodelb<=arc.eint)) then tempS=S2; break;} endif;

if (arc.sint>=arc.eint and (S2.nodelb>=arc.sint or S2.nodelb<=arc.eint)) and

(T2.nodelb>=arc.sint or T.nodelb<=arc.eint)) then tempS=S2; break;} endif; endfor;

if tempS=null then tempS=S1;endif; //Find OGN of S

For each arc in T1.arclist

{if (arc.sint<=arc.eint and (T2.nodelb>=arc.sint and T2.nodelb<=arc.eint)) and

(S1.nodelb>=arc.sint and S1.nodelb<=arc.eint)) then tempT=T2; break;} endif;

if (arc.sint>=arc.eint and (T2.nodelb>=arc.sint or T2.nodelb<=arc.eint)) and

(S1.nodelb>=arc.sint or S1.nodelb<=arc.eint)) then tempT=T2; break;} endif; endfor;

if tempT=null then tempT=T1;endif

result=treestatic(tempS,tempT);

} endif;

return result[{}]; // end of the program findpath2d()

```

Appendix: Algorithms in C++ Style Pseudo-Codes

Algorithm in section 4.6.1

```

Class Ringnode {Arc [] arclist; int nodelb;} // node definition

Class Ringarc {Ringnode stnode, ennode; float arclength; Boolean ifclockwise; int sint, eint;}

/* stnode and ennode are the start and end nodes of the arc. arclength is the Euclidean length of the arc;
sint and eint are respectively the first and second number of the linear interval label. Ifclockwise is a
sign indicating the arc's direction.*/

Program ringstatic (node s, node t)
{ Ringarc arc; Arc [] result; Ringnode ndtemp; ndtemp=s;

  While (ndtemp <> t) do

    {for each arc in ndtemp.arclist

      {if (arc.sint<=arc.eint and t.nodelb >= arc.sint and t.nodelb <= arc.eint) then

        {result.add(arc); ndtemp=arc.ennode; break;} endif;

      if (arc.sint>arc.eint and ( t.nodelb >= arc.sint or t.nodelb <= arc.eint)) then

        {result.add(arc); ndtemp=arc.ennode; break;} endif;} endfor;}

  return result [];}

```

Algorithm in section 4.6.2

```

Class Ringnode {Arc [] arclist; int nodelb;} // node definition

Class Ringarc {Ringnode stnode, ennode; float arclength; Boolean ifclockwise; int sint, eint;}

/* stnode and ennode are the start and end nodes of the arc. arclength is the Euclidean length of the arc;
sint and eint are respectively the first and second number of the linear interval label. Ifclockwise is a
sign indicating the arc's direction.*/

Program ringdynamic (node s, node t)
{ Ringarc arc; Arc [] result, result1, result2; Ringnode ndtemp;

  if (S is dynamic and T is static) then // Case 1

    {S1=findgn(S, G)[0]; S2=findgn(S, G)[1];

    for each arc in S1.arclist

```

Appendix: Algorithms in C++ Style Pseudo-Codes

```

{if ((arc.sint<=arc.eint and arc.ennode=S2 and (S2.nodelb>=arc.sint and S2.nodelb<=arc.eint)
and (T.nodelb>=arc.sint and T.nodelb<=arc.eint)) or (arc.sint>=arc.eint and arc.ennode=S2 and
(S2.nodelb>=arc.sint or S2.nodelb<=arc.eint) and (T.nodelb>=arc.sint or T.nodelb<=arc.eint)))
then result=ringstatic(S2,T); break;} endif;} endfor;

```

```

for each arc in S2.arclist
{if ((arc.sint<=arc.eint and arc.ennode=S1 and (S1.nodelb>=arc.sint and S1.nodelb<=arc.eint)
and (T.nodelb>=arc.sint and T.nodelb<=arc.eint)) or (arc.sint>=arc.eint and arc.ennode=S1 and
(S1.nodelb>=arc.sint or S1.nodelb<=arc.eint) and (T.nodelb>=arc.sint or T.nodelb<=arc.eint)))
then {result=ringstatic(S1,T);break;} endif;} endfor;

```

```

if (result=null) then
{result1= ringstatic(S1,T); result2= ringstatic(S2,T);
if (|SS1|+result1.length<=|SS2|+result2.length) then result=result1; else result=result2; endif;}
endif;} endif;

```

if (S is static and T is dynamic) then // **Case 2**

```

{T1=findgn(T, G)[0]; T2=findgn(T,G)[1];

```

```

for each arc in T1.arclist
{if ((arc.sint<=arc.eint and arc.ennode=T2 and (T2.nodelb>=arc.sint and T2.nodelb<=arc.eint)
and (S.nodelb>=arc.sint and S.nodelb<=arc.eint)) or (arc.sint>=arc.eint and arc.ennode=T2
and (T2.nodelb>=arc.sint or T2.nodelb<=arc.eint) and (S.nodelb>=arc.sint or
S.nodelb<=arc.eint))) then result=ringstatic(S,T2); break;} endif;} endfor;

```

```

for each arc in T2.arclist
{if ((arc.sint<=arc.eint and arc.ennode=T1 and (T1.nodelb>=arc.sint and T1.nodelb<=arc.eint)
and (S.nodelb>=arc.sint and S.nodelb<=arc.eint)) or (arc.sint>=arc.eint and arc.ennode=T1
and (T1.nodelb>=arc.sint or T1.nodelb<=arc.eint) and (S.nodelb>=arc.sint or
S.nodelb<=arc.eint))) then result=ringstatic(S,T1); break;} endif;} endfor;

```

Appendix: Algorithms in C++ Style Pseudo-Codes

```

if (result=null) then
  {result1= ringstatic(S,T1); result2= ringstatic(S,T2);
  if (|TT1|+result1.length<=|TT2|+result2.length) then result=result1; else result=result2; endif;}
endif;} endif;

```

if (S and T are both dynamic) then // case 3

```

{S1=findgn(S, G)[0]; S2=findgn(S,G)[1]; T1=findgn(T, G)[0]; T2=findgn(T,G)[1];

```

```

if ((S1.arclist[0].Ifclockwise=true and S1.arclist[0].ennode=S2) or (S1.arclist[1].Ifclockwise=true and
S1.arclist[1].ennode=S2) ) then {ndtemp=S1; S1=S2; S2=ndtemp;} endif;

```

//arrange S1 and S2 and let S1 be the counter-clockwise GN for S and S2 be the clockwise GN.

```

if ((T1.arclist[0].Ifclockwise=true and T1.arclist[0].ennode=T2) or (T1.arclist[1].Ifclockwise=true and
T1.arclist[1].ennode=T2) ) then {ndtemp=T1; T1=T2; T2=ndtemp;} endif;

```

//arrange T1 and T2 and let T1 be the counter-clockwise GN for T and T2 be the clockwise GN.

for each arc in S1.arclist

```

{if ((arc.sint<=arc.eint and arc.ennode=S2 and (S2.nodelb>=arc.sint and S2.nodelb<=arc.eint)
and (T1.nodelb>=arc.sint and T1.nodelb<=arc.eint) and (T2.nodelb>=arc.sint and
T2.nodelb<=arc.eint) ) or (arc.sint>=arc.eint and arc.ennode=S2 and (S2.nodelb>=arc.sint or
S2.nodelb<=arc.eint) and (T1.nodelb>=arc.sint or T1.nodelb<=arc.eint) and
(T2.nodelb>=arc.sint or T2.nodelb<=arc.eint)))
then result=ringstatic(S2,T1); break;} endif;} endfor;

```

for each arc in S2.arclist

```

{if ((arc.sint<=arc.eint and arc.ennode=S1 and (S1.nodelb>=arc.sint and S1.nodelb<=arc.eint)
and (T1.nodelb>=arc.sint and T1.nodelb<=arc.eint) and (T2.nodelb>=arc.sint and
T2.nodelb<=arc.eint) ) or (arc.sint>=arc.eint and arc.ennode=S1 and (S1.nodelb>=arc.sint or
S1.nodelb<=arc.eint) and (T1.nodelb>=arc.sint or T1.nodelb<=arc.eint) and

```


Appendix: Algorithms in C++ Style Pseudo-Codes

```
(T2.nodelb>=arc.sint or T2.nodelb<=arc.eint)))  
    then result=ringstatic(S1,T2); break;} endif;} endfor;  
} endif;  
return result[];} //end of the program
```