Simulation and Optimized Scheduling of Pedestrian Traffic

From geometric modeling to pedestrian navigation

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat) genehmigte Abhandlung

Vorgelegt von

Srihari Narasimhan

aus Chennai, Indien

Hauptberichter:	Prof. Dr. rer. nat. Hans-Joachim Bungartz
Mitberichter:	Prof. Dr.-Ing Dr. h. c mult. Paul J. Kühn
Tag der mündlichen Prüfung:	20.03.2007

Institut für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart

2007

Acknowledgments

An English novelist named G. B. Stern once quoted that "*silent gratitude isn't much use to anyone*". Taking this piece of advice personally, I decided to start my thesis by expressing my heartfelt thanks to all those who have directly or indirectly helped me with the completion of this thesis.

It is said that "a Ph.D. is an academic marriage between the candidate and the adviser, meaning equal amount of effort should be put in from both sides". I am lucky to say that my supervisor Prof. Dr. Hans-Joachim Bungartz lived up to the statement by continuously supporting me not just for every concept built in this thesis, but also throughout my stay at the department. I would therefore like to thank him for every idea, suggestion, help, encouragement, countless amount of thesis reading and feedback, and the occasional soft-kicking from the back (absolutely necessary for someone like me) which made me bring this thesis to what it is today. Finally, I would also like to thank him for being brutal with me when it came to learning the German language and thereby helping me indirectly to enjoy my stay in Germany all these years.

I would also like to thank Prof. Dr. Paul J. Kühn for agreeing to be my second examiner. Ever since the day I first arrived in Germany, Prof. Kühn has continuously extended his support to me during my studies at the INFOTECH program. He has just demonstrated his friendliness once again by taking up the role of the second examiner.

I am indebted to thank my colleagues for their continuous help and co-operation in completion of this thesis. First, I owe my thanks to Dr. rer. nat. Ralf-Peter Mundani for his suggestions to geometric and octree modeling, which truly is a base to this work. Next, I would also like to thank Dipl.-Inf. Michael Moltenbrey for the discussions regarding the relationship and the possible interface between pedestrian simulation and traffic simulation. Finally, I would like to thank Dipl.-Inf. Frank Dürr for clarifying my questions regarding the Nexus project.

Numerous feed back and suggestions have been offered by my colleagues, which contributed to a significant improvement of this thesis. I would like to begin with Dr.-Ing. Jürgen Barthlott for his kind help and corrections to the German abstract. I would like to thank my colleagues Dipl.-Inf. Michael Moltenbrey, Dr. rer. nat. Ralf-Peter Mundani, Dipl.-Inf. Ioan Lucian Muntean and Dr. rer. nat. Stefan Zimmer for their valuable suggestions, feedback and corrections to various parts of this thesis. To my old, as well as new colleagues both in Stuttgart, and München, I wish to extend my sincere thanks for making my stay during the last 5 years a peaceful and a memorable one.

I would also like to thank the Universität Stuttgart for providing me with a comfortable work place at the university and also supporting me financially during my stay at the institute.

A sincere thanks to my parents for their continuous support and encouragement all through my academic career. A special thanks to my engineer father whose motivation right from my childhood made me become what I am today.

Despite of the tremendous amount of time and effort needed to complete this thesis, it also needs a great deal of support from the outside world to afford such time and effort. My wife, Sujatha Mukunthan, has extended her love and support throughout this thesis, without which this thesis simply would not have existed today. The final stages of my dissertation coincided with her pregnancy and her tight work schedule before beginning her maternity period. Even in such a difficult situation, she took over all our private and household activities, for which, I owe my sincere thanks to her.

Srihari Narasimhan, December, 2006.
snhari@yahoo.com

Abstract

Today, more and more simulation tasks with a traditionally non-geometric background need to be embedded into some geometric context, in order to provide spatial context to non-spatial data. This holds especially true for graph-based applications in some location-aware context. As an example, one might think of a theme park or a large commercial center, where the customers shall be provided with some navigation and scheduling information such as where to go and when – either a priori or even in real time via some mobile device. This can be done by analyzing the pedestrian traffic and waiting time situation by simulating the pedestrian movement and using the simulation data to optimally navigate and schedule the tasks that are to be executed by the customer. The main issues addressed in this thesis are as follows.

Initially, a flexible simulation framework is built to simulate the pedestrian movement in a 3D scenario, for example, a commercial building. Since the pedestrians strongly interact with the environment surrounding them, the geometry is taken into account. Architectural data such as paths, type and capacity of the paths, destinations and its properties, etc., is extracted from the CAD-model and are organized in a graph structure. The movement of the pedestrians and the waiting queues at the destinations are modeled as queuing systems using the discrete event simulation technique. These queuing systems are then embedded into the geometry model. The necessary input modeling parameters are also defined. The resulting scenario, when simulated, gives an overview of congestions and waiting times across the scenario for different time stages.

Apart from the simulation, the geometry data – or here the graph – is hierarchically organized in an octree structure. An octree-based model is chosen since octrees have the natural property of hierarchically storing 3D data. The octree data is used to identify the position of the pedestrian within the scenario. The potential destinations in the neighborhood that can be visited by the customer are also identified using neighbor search algorithms. Combining the simulation data with the octree modeling, the customer is navigated to the optimal destination.

Furthermore, when visiting several destinations, combinatorial optimization methods are used to optimally schedule the set of tasks to be executed by the customer. The optimization methods take into account the congestion information obtained from the simulation data, and the octree structure for navigation. This approach results in an effective pedestrian navigation system.

Zusammenfassung

Es besteht heute oftmals die Notwendigkeit, Simulationen ohne geometrischen Hintergrund in einen geometrischen Kontext zu integrieren, um einen Bezug zwischen räumlichen und nicht-räumlichen Daten herzustellen. Dies gilt insbesondere für graphenbasierte Anwendungen mit räumlichem Bezug. Wir betrachten hier ein System, das in einem großen Bürogebäude Kunden Informationen zur Navigation und Zeitplanung zur Verfügung stellt, d.h. wohin sie zu welchem Zeitpunkt gehen müssen. Diese Informationen können im Voraus oder in Echtzeit über ein mobiles Gerät zur Verfügung gestellt werden. Dazu wird zunächst das Fußgängerverhalten und die Wartezeit an Bedienstationen in einem dreidimensionalen Modell (Bürogebäude) analysiert, indem das beschriebene Szenario simuliert wird. Danach werden die verschiedenen möglichen Anwendungen unter Verwendung der Simulationsdaten untersucht. Ziel einer solchen Anwendung ist die Entwicklung eines intelligenten Fußgänger-Navigationssystems – betrachtet man z.B. einen Kunden, der das Bürogebäude besucht, um bestimmte Aufgaben durchzuführen, besteht das Problem aus der optimalen Planung, die sowohl Informationen über den Weg als auch die Bearbeitungsreihenfolge der Aufgaben enthält. Dabei werde das aktuelle (Fußgänger-)Verkehrsaufkommen sowie die Warteschlangensituation im Bürogebäude berücksichtigt. Die notwendigen Daten erhält man als Ergebnis der Fußgängersimulation.

Problembeschreibung

Modellierung und Simulation des Fußgängerverhaltens – mit typischen Anwendungen z.B. Evakuierungsszenarien, Wartesystem-Modelle und die Abschätzung und Verfolgung des Besucheraufkommens – ist abhängig von geometrischen Parametern, die das Fußgängerverhalten beeinflussen. Die meisten Modelle zielen auf ein sehr spezifisches Szenario wie z.B. die Schätzung der durchschnittlichen Wartezeit an der Kasse eine Supermarktes – *Ist es sinnvoll, zusätzliche Kassen zu planen? Wenn ja, wie viele?* In solchen Anwendungen werden nur grundlegende geometrische Parameter wie die Länge der Warteschlange und die Kundenkapazität betrachtet. Architekturdetails wie z.B. der Abstand zur Wand (Flurkapazität), genaue Maße des Raumes (Fläche), Art der Wege (Treppe, Rampen usw.) und Türen werden in solchen Fällen hingegen ignoriert. In manchen Fällen, wie bei der Simulation eines Evakuierungsszenarios oder der “Location-Aware” Fußgängersimulation, ist es jedoch wichtig, nicht nur die Architekturdetails zu betrachten, sondern auch das Simulationsmodell in den geometrischen Kontext zu integrieren. Bei der Modellierung eines Evakuierungsszenarios in einem Bürogebäude, in dem ein Notfall eintritt, ist hier das Ziel, zu überprüfen, ob die Zeit, die für die Räumung benötigt wird, innerhalb der erlaubten Grenze liegt. Falls die Simulation keine starke Kopplung zwischen den Einheiten und dem Geometriemodell vorsieht, kann das System weder eine unerwartete Blockierung (Säulen, Möbel, usw.) neben dem Ausgang erkennen noch besteht eine Möglichkeit, das Geometriemodell zu optimieren (Änderung des Architekturdesigns).

Aufgabenstellung

Zielsetzung dieser Arbeit ist die Entwicklung eines Softwarekonzepts für eine Arbeitsumgebung zur Integration und Einbettung von Fußgängersimulationen in eine Geometrieumgebung. Die Aufgabenstellung gliedert sich dabei wie folgt:

- Entwicklung eines effizienten Verfahrens zur Extraktion und Strukturierung von Wegstrecken, Zielknoten und Architekturdetails aus dem Geometriemodell
- Hierarchische Strukturierung der extrahierten Dateien und Entwicklung eines Verfahrens für Positionserkennung, Nachbarschaftssuche und Fußgängernavigation
- Entwicklung einer graphen- und wartesystembasierten Fußgängersimulation; Integration und Einbettung des Simulationsmodells in das Geometriemodell.
- Organisation und Planung für einen Besucher des Gebäudes mit der Analyse des Verkehrsaufkommens unter Verwendung der Daten der Simulation.

Geometrische Modellierung

Fußgänger bewegen sich in einem Szenario normalerweise mit einer spezifischen Absicht. Beim Modellieren des Fußgängerverhaltens wurden zwei verschiedene Aktivitäten berücksichtigt, nämlich die Bewegung entlang einer Route und das Verhalten am Ziel. Um diese beiden Aktivitäten zu modellieren, identifizieren wir zunächst die Wege, auf denen sich die Fußgänger bewegen und die Zielknoten, an denen sie ihre Aufgaben erledigen. Die Liste der Wege und die Liste der Zielknoten werden durch Analyse des gegebenen CAD-Modells automatisch identifiziert. Diese Daten werden danach miteinander vernetzt, um eine Graphenstruktur zu erzeugen. Dabei stellen die Kanten des Graphen die Wege dar und bestimmte Knoten des Graphen beziehen sich auf die Zielknoten im Szenario. Einige andere Eigenschaften wie z.B. die Kapazität des Weges, der Abstand zwischen den Wänden und die Länge der Wegstrecke werden ebenfalls identifiziert.

Der resultierende Graph zusammen mit allen weiteren abgeleiteten Parametern wird dann benutzt, um den Fußgängerverkehr zu modellieren und zu simulieren.

Hierarchische Strukturierung des Modells

Es besteht die Möglichkeit für weitere Anwendungen des erzeugten Graphen, wie z.B. Pfadsuche-Algorithmen oder Navigationssysteme. Dafür wird das Geometriemodell – oder hier der Graph – in einer hierarchischen Struktur (Oktalbäume) gespeichert. Die Knoten des Graphen werden mit Hilfe eines Oktalbaumes partitioniert, bis jedes Blatt des Oktalbaums maximal einen Knoten enthält. Der Vorteil bei der Verwendung eines Oktalbaums ist, dass er eine natürliche Eigenschaft besitzt, dreidimensionale Daten hierarchisch zu speichern. Angenommen, ein Kunde, ausgerüstet mit einem Navigationsgerät, kommt in ein Einkaufszentrum und will eine Apotheke finden. Weiterhin nehmen wir an, dass das Navigationsgerät die aktuellen Positionskoordinaten des Kunden ermitteln kann. Dann können die Koordinaten in der Oktalbaumstruktur durchsucht werden, um die genaue Position im Oktalbaum und damit den nächsten Punkt auf dem Graphen zu identifizieren. Der Vorteil bei der Verwendung einer solchen Methode ist die Steigerung der Effizienz bei der Positionssuche. In Vergleich zur Linearsuche (Abstand zwischen aktueller Position und allen Knoten berechnen und dadurch die Nachbarknoten erkennen) mit einer Komplexität von $\mathcal{O}(n)$ ist die hierarchische Suche mit der Komplexität $\mathcal{O}(\log n)$ wesentlich günstiger, insbesondere für große Graphen (z.B. die Darstellung einer Stadt). Aufgrund der hierarchischen Struktur, wird die enorme Größe des CAD-Modells auf einen einfachen Graphen und eine Oktalbaumstruktur reduziert. Auf die gleiche Weise wurden Nachbarschaftssuche-Algorithmen auf Oktalbäume angewendet, um eine Liste der Zielknoten in der unmittelbaren Nähe des Fußgängers zu identifizieren, die Wege auf dem Graphen zu kennzeichnen und den Fußgänger zum gewünschten Zielknoten zu führen.

Fußgängerverkehrssimulation

Um Fußgängerbewegungen und -verhalten zu modellieren und zu simulieren, konzentrieren wir uns auf die Methode der ereignisdiskreten Simulation. Die Modellierung des Fußgängerverhaltens ist aufwändig: Typische Aktionen, wie das plötzliche Stoppen, das Überholen anderer Fußgänger, die Änderung des Kurses (Richtung), die schnelle Änderung der Geschwindigkeit und Beschleunigung, das Treffen spontaner Entscheidungen, die Interaktion mit anderen Fußgängern und das Vermeiden von Zusammenstößen, sind - verglichen etwa mit der Simulation von Fahrzeugen - häufig schwierig zu modellieren. Diese Parameter beeinflussen auch andere Fußgänger in der Umgebung. In den meisten Fällen jedoch basiert das Modell auf den Parametern Geschwindigkeit, Dichte und Fluss. In dem verwendeten Ansatz wird beabsichtigt, ein mikroskopisches Fußgängermodell entlang des Weges und den Zielknoten zu simulieren. D.h, jeder Weg wurde so extrahiert, dass man sich in beide Richtungen geradlinig bewegen kann und die o.g. Parameter, wie die Geschweindigkeitsänderung und das plötzlich Stoppen, bleiben entlang des Wegs konzistent. Deshalb können diese Parameter ignoriert werden. Sowohl die Bewegung entlang der Route als auch das Verhalten am Bestimmungsort wurden als Wartesystem modelliert. Im Falle einer Bewegung entlang der Route kommen die Fußgänger an einem Wegstück an, warten, wenn der Weg blockiert ist, und beginnen den Weg entlang zu gehen, wenn sich die Menge bewegt. Die Laufgeschwindigkeit wird durch den Status des Weges (z.B. Dichte, Bewegungsrichtung, Kapazität) berechnet. Wenn die Fußgänger am Ziel angekommen sind, erreichen sie die Warteschlange, warten so lange, bis eine Bedieneinheit frei geworden ist, und erledigen dann ihre Aufgabe. Das Wartesystem wird in jeden Weg (Kante) und jeden Zielknoten (Knoten) im Szenario eingebettet, um ein Wartenetz zu bilden.

Die Modellierung der Eingangsdaten ist ein wichtiges Konzept bei der Ereignisdiskreten Simulation. Da jeder Fußgänger unterschiedliche Eigenschaften und Zielsetzungen hat, wurden einige Profile definiert und in einer Datenbank verwaltet. Diese Profile werden verwendet, um die Eingangsdaten für die Fußgängersimulation zu erzeugen, die mittels stochastischer Funktionen als Eingangsgrößen des Wartesystems ausgewählt werden. Die Simulation des resultierenden Modells gibt einen Überblick über das Verkehrsaufkommen und den Status der Warteschlangen, die im Szenario während der unterschiedlichen Zeitpunkte auftreten.

Das Simulationsframework wurde so flexibel entworfen, dass es generische CAD-Modelle importieren und die nötigen Parameter extrahieren kann. Außerdem ist die Modellierung der Eingangsdaten so entworfen, dass beliebig viele Kundenprofile definieren werden können. Dadurch ist es möglich, eine Vielzahl verschiedener Szenarien zu simulieren.

Routenplannung

Auf Basis dieser Simulation wird untersucht, welche Möglichkeiten bestehen, die statistischen Daten aus dem Fußgängersimulationsmodell zu verwenden. Betrachten wir wieder das Szenario, bei dem ein Kunde in ein Kaufhaus kommt und nach einer Apotheke sucht. Seine Position im Gebäude und eine Liste der Apotheken in der Umgebung seien bereits identifiziert. Nutzt man die Simulationsdaten, kann man die Apotheke mit der geringsten Wartezeit und den Weg dorthin mit dem geringsten Verkehrsaufkommen ermitteln. Falls der Kunde mehrere Aufgaben erledigen möchte, kann man die Simulationsdaten dazu verwenden, um den Status aller Zielknoten zu identifizieren. Dadurch kann eine optimale Routenplanung zur Durchführung der Aufgaben und zur Navigation zwischen jeder Aufgabe vorbereitet werden. Das Problem enthält das kombinierte Finden des optimalen Weges und der optimalen Reihenfolge. Die kontinuierliche Änderung der Simulationsdaten über die Zeit muss für die Routenplanung und Navigation ebenfalls betrachtet werden. Einige kombinatorische Methoden und Heuristiken wie "Brute-Force search", stochastische Optimierung, Greedy-Heuristiken, Nachbarschaftssuche (nearest-neighbor) und die Nutzung einiger Vorbedingungen und Einschränkungen wurden im Rahmen dieser Arbeit verwendet, um einen optimalen Routenplan für den Kunden zu erzeugen. Die im Rahmen dieses Auskunftssystems gewonnenen Informationen über die Einheiten, die das System nutzen, werden in die Simulation einbezogen, um die Simulationsdaten ständig aktuell zu halten.

Aussicht

Durch das Zusammenführen der oben beschriebenen Methoden ist es möglich, das ganze Szenario in ein intelligentes Fußgängernavigationssystem zu konvertieren. Der Schlüssel zu einem solchen System ist die Koppelung des Simulationsmodells in der Geometrie des Szenarios. Dieser Ansatz eröffnet auch einige weiterführende Fragestellungen, wie z.B. Gebäudeplanoptimierung (*Passen die täglichen Szenarien zur Architektur? Sollte der Hauptweg breiter sein?*), Evakuierungsszenarien (*Blockiert die Säule die Evakuierung? Können die Fußgänger schneller evakuieren werden, falls die Säule entfernt wird?*), Entscheidung über die Örtlichkeiten innerhalb der Gebäude (*Wo werden mehr Besucher erwartet? Ist es besser, die Gaststätte an einen anderen Ort zu verschieben?*). Ein anderes Thema, das in dieser Arbeit diskutiert wird, ist das Skalierbarkeit des gesamten Systems. Wenn das Fußgängerverhalten z.B. in einem großen Vergnügungspark simuliert wird, könnte der Graph aus sehr vielen Knoten bestehen und die Simulation könnte sehr viele Kunden enthalten. Obwohl die Simulation im Framework skalierbar ist, benötigt jeder Knoten und jedes Kundenbedürfnisse eine gewisse Rechnerzeit und ein normaler PC wäre für so ein großes Modell nicht leistungsfähig genug. Daher müssen in solchen Fällen parallele und verteilte Methoden verwendet werden. Mit diesem Ansatz wird nicht nur Rechnerkapazität gespart, sondern auch Rechenzeit, falls mehrere Iterationen simulieren werden. Wenn man z.B. bei einer Gebäudeplanoptimierung unterschiedliche voneinander unabhängige Szenarien simulieren will, kann man diese parallel simulieren und die Enddaten später gemeinsam analysieren.

Obwohl die Simulation einen guten Überblick über das Verkehrsaufkommen und den Status der Warteschlangen gibt, können bestimmte spontane Änderungen (wie Unfälle, der Ausfall einiger Bedieneinheiten oder Situationen wie eine unerwartete Blockierung, die durch eine Gruppe von sich langsam bewegende Menschen verursacht wird), nicht exakt simuliert werden. Abhilfe können hier Sensoren schaffen, die mit mobilen Geräten und Kommunikationsservern kommunizieren. Solche Sensoren übertragen den aktuellen Status, in der sie sich befinden; durch das Verwenden von Echtzeitdaten solcher Sensoren, können die Simulationsparameter ständig aktualisiert und die Genauigkeit der Simulation verbessert werden. Diese Schnittstelle ist in weiten Teilen noch offen.

Contents

List of Figures

Chapter 1

Introduction

I think that in the discussion
of natural problems we ought
to begin not with the Scriptures,
but with experiments, and demonstrations.
– **Galileo Galilei**

Due to the rapid growth in technology and lifestyle, transportation has become increasingly important. Commuters using the public transportation to get to the work place, traffic on the highway due to cars and other vehicles, transportation of packets from one place to another, tourists traveling to different cities: all these concepts show that mobility is one of the most common activities seen in day to day life. As the number of entities participating in the transportation world increases, a bottleneck in the flow of the entities often occurs. Also, the infrastructure of the environment plays a significant role in keeping the traffic flow in motion. All these gives rise to the concept of mobility modeling. Modeling of mobility is the process of analysis of the behavior of the different participating entities in the mobile world. Mobility modeling is performed with an aim to improve the existing infrastructure as well as provide additional services such that mobility is made easier.

Mobility modeling of people can be broadly classified into vehicle traffic and pedestrian traffic. Modeling of both vehicle as well as pedestrian traffic is performed at various instances. The typical aim of modeling vehicle traffic is to analyze the congestion on roads, study of microscopic behavior of individual vehicles participating in the traffic, urban and town planning. On the other hand, pedestrian traffic modeling is seen in evacuation scenarios, improvement of pedestrian infrastructures in towns and cities, study of pedestrian dynamics, etc.

In case of a pedestrian modeling, the typical methods used are agent-based pedestrian simulation, cellular automata model, flow-based numerical simulation, and also queue models in simple cases, for the purpose of evacuation planning, estimation or tracking the visitors, or analyzing the flow density during peak hours. In all the situations, a strong integration with the geometry of the environment is seldom considered. However, in many cases, it is important to embed the simulations with traditionally non-geometric background into a geometry context. This is especially true for applications that require spatial data to model the scenario. The embedding into the geometry environment provides spatial context to non-spatial data.

1.1 Topic Description

In the different pedestrian simulation models analyzed so far, the concept of embedding the simulation into a geometry context was missing and it was also found that such an integration is important.

Consider for example a scenario where a pedestrian arrives in a commercial building or a similar environment to execute a certain list of tasks. The objective of the pedestrian is to determine an optimal sequence of the tasks to execute depending on the waiting time to execute each task, and an optimal path through the building considering the possible traffic congestions in the building. The customer therefore requires an effective navigation system that would prepare such a schedule and also suggest an optimal route through the building. Such a route planning is either made in advance considering the possible traffic congestions during the given time of the day or in real time once the customer is inside the building with some kind of mobile navigation device.

To schedule the tasks optimally, exact information about the waiting times at each service center is required. These service centers are modeled as queuing systems. These queuing systems, when simulated, give an overview of the waiting times at each service center. The results are used to schedule the tasks. Similarly, to prepare an optimal route, information about the traffic congestions in the building is as well required for which, the paths along the building are also modeled also as queuing systems. The results provide information on possible congestions inside the building at any given point of time.

Apart from this, the geometrical model of the building is taken into account which provides spatial information such as the capacity of waiting rooms, capacity of paths in the building and so on. The geometry embedding gives a spatial context to non-spatial scenarios. Location awareness is another key area, which provides the co-ordinates of the entity and also identify the happenings in the neighborhood. These informations are necessary for dynamic scheduling and routing of the tasks.

In this thesis, a generic framework is built, where the pedestrian behavior in a building is modeled and simulated, and the simulation is in turn integrated strongly within the geometry of the building. Possibilities of providing an effective navigation system to the customers are then analyzed by considering the simulation data.

1.2 Current Research and New Challenges

Pedestrian navigation services are often seen in the field of ubiquitous and pervasive computing systems that apply to location- and context-aware applications. Several researches has been done in providing improved context-aware services based on the location [BCMS06, BBR01, SAW94, ST93]. Due to the availability of affordable mobile hardware devices, location- and context-aware computing gained increasing popularity. Some of the typical uses include, forwarding the telephone call to the nearest telephone booth, guiding and navigating a visitor in a new environment, identifying the location of the nearby public transport service and determining the timetable to a destination (the destination may be derived from the diary planner in a PDA – Personal Digital Assistant), etc. These applications are not just restricted to navigation services (path identification in a new environment) but are made to exploit locally available information such that improved services can be provided to the user. An overview of such applications and the challenges in realizing them can be seen in [Sat01].

The Nexus[1] project (Spatial World Models for Mobile Context-Aware Applications), funded by the German Research Foundation under the title "SFB 627", currently an active project at the Universität Stuttgart, enhances mobility by inducing spatial-aware applications [HKL$^+$99]. The Nexus project targets technologies in the definition and realization of spatial world models, which include communication and information management, model representation, use of sensor data, etc. Applications in relation to the pedestrian navigation system mentioned in this thesis are used to validate the spatial world models built within the framework of the Nexus project.

Even though the pedestrian simulation and navigation framework presented in this thesis has some similarities with the Nexus project, both these approaches run on a different track and the ends do not currently meet. The pedestrian simulation

[1] `http://www.nexus.uni-stuttgart.de/`

framework relies on statistical data to model and simulate the pedestrian behavior using stochastic functions. The usage of sensor information as in the Nexus project to obtain current status of the scenario can enhance the performance of the pedestrian simulation. This interface is however still an open avenue of research.

1.3 Goals and Contributions

The objective of this thesis is to build a flexible framework for embedding the pedestrian simulation in a geometry context and thereby investigate the possibility of building an intelligent pedestrian navigation system using the data from the pedestrian simulation. The goals of the thesis are listed as follows.

- Development of an efficient method to extract and structure the list of paths, list of destinations and the various architectural parameters from a given geometric model.
- Structuring of the extracted data hierarchically and development of a position identification system by using the hierarchical data. Also the implementation of efficient neighbor search methods and a method for a pedestrian navigation system.
- Development of a graph- and a queuing system-based pedestrian simulation framework and embedding the simulation model into the geometry model.
- Usage of a framework to organize and plan a visit by the pedestrian in the given scenario. Efficient generation of an itinerary or a visit schedule by considering the pedestrian congestion and waiting queues obtained from the simulation data.

1.4 Thesis Outline

Reference Scenario and Model
Throughout this thesis, we imagine a generalized hypothetical scenario and develop each component needed to realize this scenario. We assume a generic scenario, where a pedestrian arrives at a building with an intention to execute a set of tasks in the building. The pedestrian wishes to complete these tasks within the allotted time limit. This scenario will henceforth be termed as *reference scenario* and the building to be visited by the pedestrian will henceforth be termed as *reference model*. In order to realize this scenario, the first step is to model

and simulate the pedestrian behavior in the given geometry environment. This means the geometry of the reference model must be taken into account and the simulation must be embedded within the geometry. The next step is to use the simulation data from the simulation to plan an optimal schedule and routing information such that the pedestrian executes the tasks in the given time. Several components and background information are needed to build such a system. They will be explained step by step throughout the thesis.

Thesis Structure
Figure 1.1 shows the structure of the thesis and the chapters, where the various components are realized. The chapters of this thesis are structured as follows.

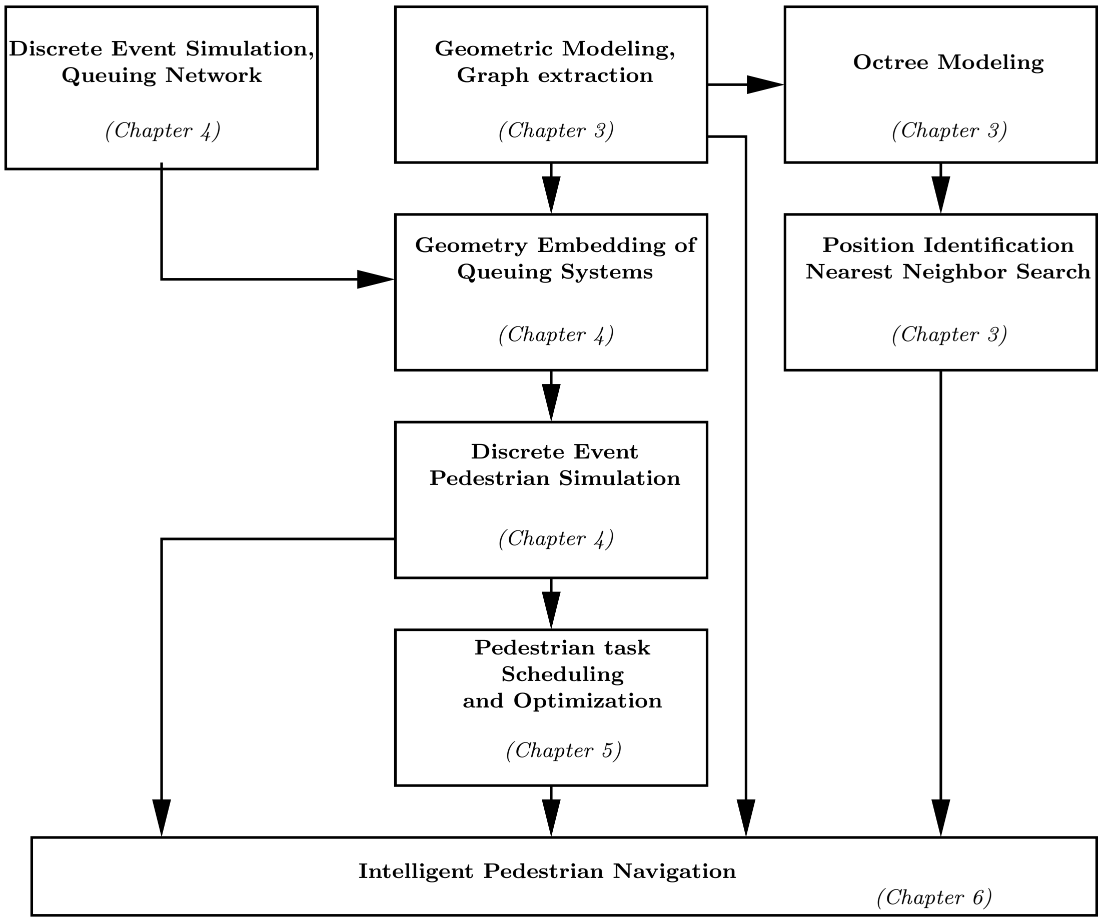

Discrete Event Simulation, Queuing Network
(Chapter 4)
Geometric Modeling, Graph extraction
(Chapter 3)
Octree Modeling
(Chapter 3)
Geometry Embedding of Queuing Systems
(Chapter 4)
Position Identification Nearest Neighbor Search
(Chapter 3)
Discrete Event Pedestrian Simulation
(Chapter 4)
Pedestrian task Scheduling and Optimization
(Chapter 5)
Intelligent Pedestrian Navigation
(Chapter 6)

Figure 1.1: Structure of the Dissertation.

Chapter 2 provides an overview of some fundamental concepts that are needed to understand this thesis. The concepts of algorithms and data structures necessary for the implementation of the reference scenario are discussed here. Also the frequently used hierarchical data structures – octrees – in structuring the reference model are also discussed here. Since the pedestrian behavior is modeled and simulated using the discrete event simulation methodology, a short introduction to the components in a discrete event simulation is given. The concepts of location and geometry awareness, scalability issues, etc., are also briefly discussed.

Chapter 3 introduces the concept of geometric modeling and describes a method to provide pedestrian navigation. In the reference scenario, the pedestrian arrives at the reference model to execute a certain list of tasks. Due the strong interaction of the pedestrian with the geometry of the building the CAD model of the reference model is parsed in chapter 3 to extract the geometric and architectural properties of the building. The method to extract the geometry parameters and the structuring of the geometry data in form of a graph is presented. Then, using the graph, an environment to support pedestrian navigation through position identification, neighbor search methods and path-search algorithms is developed. For test cases, the model of the computer science building at the Universität Stuttgart will be used.

Chapter 4 models and simulates the pedestrian behavior within the given reference model. In the reference scenario, the schedule for the pedestrian visit is planned based on the actual congestion and waiting time situations that exist in the reference model. In order to identify such congestions and queue sizes, the discrete event simulation methodology is used to model and simulate the pedestrian behavior. The movement of the pedestrians, as well as the behavior in a queue is modeled using a queuing system. Chapter 4 presents a flexible framework, where a specific pedestrian scenario can be built (number of pedestrians, type of pedestrians, the activities executed by a pedestrian, etc.) and the resulting scenario is used as input parameters to model the pedestrian behavior. The simulation is then embedded into the geometry of the reference model and the entire system, when simulated, gives us an overview of possible congestions and waiting times that occur at different points in the reference model during the simulation time period. The status of each path and destination is periodically collected during the simulation and the simulation data is used to schedule the set of tasks for a new visitor.

Chapter 5 analyzes different optimization methods to plan and schedule the tasks to be executed by the pedestrian. In the reference scenario, the pedestrian arrives with an intention to execute a number of tasks in the shortest possible time. Chapter 5 uses different combinatorial optimization methods and heuristics such as brute-force search, greedy approaches, simulated annealing methods, etc., to optimally sequence the list of tasks to execute as well as find an optimal path between two tasks located in different points of the building. An analysis of the different methods is made by mentioning the worst, best and average case performances of the optimization methods for different types of pedestrian scenarios.

Chapter 6 integrates the components developed in chapters 3,4 and 5 within a single pedestrian simulation framework and presents some numerical examples for each phase of the simulation by defining a hypothetical scenario. Once the scenario is modeled and simulated, the simulation data regarding the congestions and waiting times are collected. The tasks of the pedestrian in the reference scenario are optimally scheduled by using the optimization methods presented in chapter 5.

Chapter 7 summarizes the concepts developed in this thesis and lists out the targets reached. An outlook into the usage of the pedestrian simulation framework for other possible applications such as evacuation simulation, mobile computing, building plan optimization, etc., as well as extensions to the components developed throughout the thesis is made.

Chapter 2

Underlying Concepts and Nomenclature

Education is what remains after one has forgotten everything he learned in school.
– **Albert Einstein**

This chapter introduces certain fundamental background concepts that are related to this thesis work. Initially, an overview of the concepts of algorithms and data structures is made. In this part, the hierarchical data structures and graph theory that are often used in this thesis are also discussed. Certain other geometry concepts such as location awareness, navigation systems, etc., and also the issues on scalability and parallelization are discussed next. Finally, an overview of the various components of a discrete event simulation and its internal working is then presented.

2.1 Algorithms and Data Structures

When solving a problem or writing a program, the foremost study is actually the method that is used to solve the problem. In computer science, the term *algorithm* describes the methods to solve a problem, which can be translated to a computer program. There may exist more than one algorithm to solve a problem. Algorithms mostly involve methods to organize and structure the data that is used for computation. Such objects created to store, handle and manipulate such data are the *data structures*. Algorithms and data structures are well explained in [AHU87, Sed98].

Algorithm and Abstraction

Formally, an algorithm is a finite sequence of well defined instructions to solve a specific task, where

- each instruction is clearly defined,
- the instructions can be executed with a finite amount of effort,
- the instructions may or may not require input data,
- each instruction exits after a finite amount of time (not entering an endless loop), and
- the algorithm produces a finite (correct) output.

Algorithms are generally not defined for a very specific purpose, but are rather general. When solving a problem, generally existing methods are studied and a solution is then built upon by using the existing methods. Therefore, the method of *abstraction*, i.e. reduction and factoring of the concepts such that the concepts can be focused on separately, is used. An object-oriented programming language such as C++ [Str01] (also the programming language used in this thesis) uses objects that consist of data and operations to be performed on this data. Therefore, the use of abstract data types (ADT) is seen here. A *class* describes an object such that many numbers of same objects can be created and different data can be processed in each object. The classes can be represented graphically with the use of UML (Unified Modeling Languages).

The concept of an algorithm was formalized through the TURING machine [Tur36]. The CHURCH-TURING thesis stresses that TURING machine indeed demonstrates the method in logic and mathematics, and states that any computable problem can be performed an algorithm by running on the computer.

Complexity Analysis

For practical applications, efficiency of the algorithm plays an important role. Two parameters analyzed are:

- **Space Complexity**: The amount of memory needed for computation
- **Time Complexity**: Depending on the problem size, the amount of time or the number of steps needed for computation

To analyze the performance of the algorithm, we study the computational complexity of the algorithm. The algorithm is analyzed by calculating the cost with the increasing problem size of n. The *Big-$\mathcal{O}$* notation is used to represent the computational complexity. To differentiate between the behavior of the algorithm for different data sets, situations such as

- best case that requires the least effort (e.g., searching n numbers, where the required result is located first in the list),
- average case that requires the normal effort (e.g., searching n numbers, where the required result is located somewhere in the middle or the list),
- worst case that requires the most effort (e.g., searching n numbers, where the required result is located at the end of the list),

is used.

Design of the algorithm is often a trade-off between the cost and effort. Therefore, algorithms that are used often and require a lot of memory and computing capacity are optimized to the maximum extent possible, whereas negligible algorithms are implemented without much optimization.

Elementary and Hierarchical Data Structures

Apart from the data types used to represent data (`integer` for representing integer number or `float` for representing real numbers), handling of a large amount of data is done with the use of data structures. One of the most elementary methods to represent a set of data is a *linked list* (where individual variables are connected through pointers and pointers denote the sequence of the data) or an *array* (where the list of data are accessed through an index). In a linked list, the

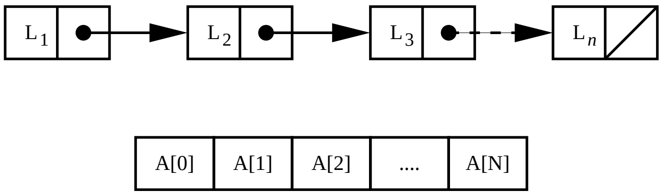


Figure 2.1: Representation of a linked-list (top) and an array (bottom).

memory is allocated dynamically and each variable points to the next variable, whereas in an array, the required amount of memory must be reserved in advance as a block (see Figure 2.1). Apart from memory issues, the efficiency of the operations that are performed on the data structures also counts. For example, when searching for a specific data, the entire list (or array) must be searched, which results in a computational complexity of $\mathcal{O}(n)$. If the list is already sorted, a *binary search* can be performed on an array. In a binary search, the middle element in an array is searched and if there is no match, the array is partitioned and either the lower or the higher block is searched. Due to the partitioning, the search is logarithmic. Therefore, the computational complexity in case of using an array is $\mathcal{O}(\log n)$. Other data structures such as stacks, queues, etc., can be built upon by using a linked list or an array. A decision on choosing the data structure is made by considering the overall cost of the algorithm (time and space complexity). The objective is to minimize the overall cost.

The concept of hierarchy is often seen in computing, where an algorithm or a program is split up into several modules and each module are in turn nested with more modules. In a computer, the components are composed of modules and are organized hierarchically (board, microprocessor, chip, transistor). In data organization (file system or a database), the data (or files and directories) are structured hierarchically. In an object-oriented programming, classes are organized hierarchically and the classes are related through inheritance.

The typical data structure used to organize the data hierarchically is the tree structure. A tree – originally said to have originated from graph theory – is a collection of elements called *nodes*, and one of the nodes is defined as a *root* of the tree from which the remaining nodes are structured hierarchically. The nodes share a "parent-child" relationship. The bottom most node is called the *leaf* node of the tree. Each node, except the leaf nodes, shall have one or more child nodes and each node, except the root node, shall have one parent. If the tree contains only one element, the root node is same as the leaf node. The (maximum) number of children N for each parent node denotes the type of the tree – N-ary Tree (for $N = 2$, the tree is called a binary tree).

The concept of recursion is strongly coupled with the term hierarchy. A recursion is a function that calls itself. Once the steps are complete and the result is achieved, the recursion terminates. Computation of a factorial ($n! = n \times (n-1) \times \cdots \times 1$) can be done efficiently using a recursion function. The factorial function $F(n!) = n \times F((n-1)!)$ is therefore performed by recursively calling the factorial function as follows

```
int factorial(int N)
  {
    if (N==0) return 1;
    return N*factorial(N-1);
  }
```

Many algorithms work on the basis of the "*divide-and-conquer*" rule, i.e. in each recursion, the problem is partitioned and the sub-problems are then solved. The individual solutions are then combined to determine the final solution of the problem. The binary search technique is an example of divide-and-conquer recursion. Both the concepts of hierarchy and recursion are used frequently in this thesis.

Octrees

Octrees, like any other tree structure, are hierarchical data structures that are used to store data hierarchically. As the name implies, each parent in an octree can have a maximum of upto 8 children. Octrees are often used to represent 3-dimensional geometry data and its analog – Quadtree – is used to represent 2-dimensional geometry. For the given geometry, an octree is built as follows. First, the geometry object is bounded by a cube that just covers the entire geometry. The cube is then partitioned in all 3 dimensions along the x, y and z axis to form 8 cubes or octants. The octree is partitioned recursively until the object lies completely inside or outside an octant. This results in 3 cell types, namely geometry *in* the cell, geometry *out* of the cell and geometry *on* the cell. The geometry that lie *on* the cell needs recursive partitioning as long as they are either *in* or *out*. Certain curves sometimes may not lie completely inside or outside an octant even after several partitions. Therefore, the maximum depth or level of partition is generally defined to avoid endless partition. The octree is then stored as a tree structure and the leaves of the octree are used to identify the geometry structure. The inner nodes can be omitted. If the geometry is partitioned using equidistant cells with n cells in each direction, the computational complexity of the cell structure is $\mathcal{O}(n^3)$. Octrees on the other hand has a complexity of $\mathcal{O}(n^2)$ for the same geometry and the same amount of detail. Figure 2.2 illustrates the partitioning of a 2-dimensional geometry using a quadtree structure.

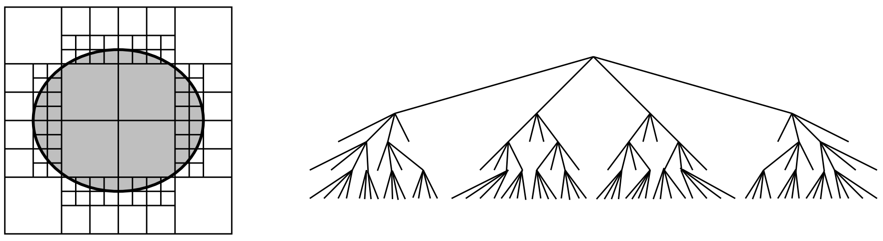


Figure 2.2: Quadtree partition of an ellipse (left) and the tree structure representation (right).

The hierarchical data structure quadtree was first named by R. Finkel and J.L. Bentley [FB74] in 1974. Ever since the introduction of such hierarchical structures to structure spatial data, quadtrees and octrees have been widely used for various applications. One of the foremost applications was in the area of computer graphics: image representation, view frustum culling of terrain data, etc. In geo-spatial applications, octrees are used for spatial operations due to its efficient boolean operation. Octrees are also commonly used as interfaces to applications in the field of simulation and visualization.

Even though the complexity of generating an octree is lower, the computing time for such recursive algorithms increases exponentially when generating octrees for geometries of higher resolution. This is critical especially if octrees need to be generated frequently. This is because, a floating-point decision on whether or not to refine the structure has to be taken for each cell and each recursion. [MBR$^+$03] proposes an efficient method to generate octrees in real time and also on-the-fly large recursion levels (even > 12). In [MBR$^+$03], face of the surface-oriented model is treated as a plane that divides the space in two half spaces (*in* and *out*). A volume-oriented model is built from intersecting all *in* attributes of the half spaces. The octree is then generated by comparing the corresponding plane that intersects the octree and encoding the result as a binary sequence. The octree generation is therefore free of redundant calculations and the overall memory requirements are reduced due to the use of stacks.

Octrees with a large depth will increase the number of voxels exponentially. The data resulting such octrees will be enormous. Therefore, efficient traversing and coding structures are needed to organize the octree data. Apart from organizing the octree in a classical tree structure, each node of the tree is identified by an unique identifier. The Morton index is used to identify the nodes. By naming each of the 8 nodes (of the root node), from '0' to '7' in a specific and consistent sequence, the Morton index of a cell can be calculated by accumulating all the numbers of the nodes by traversing from the root node till the desired node. This index-based method is termed as *position coding.* The pre-order tree traversing is used to parse the tree and process the individual nodes. Figure 2.3 shows a quadtree with position codes and the order of tree traversing.

Octrees are typically sequenced in the Lebesgue curve fashion.[1] Figure 2.4 shows an octree with Morton index in the sequence of a Lebesgue curve.

The algorithm 1 shows the method to generate an octree. The algorithm must then take care that the cells resulting from the octree are inserted into the tree

[1]Space filling curves (SFC) are a continuous function whose ranges contain the entire 2-dimensional unit square (or the 3-dimensional unit cube). In a discretized region, the curve passes through all the regions and each node of the curve occupy the entire cell. Several schemes to design such curves exist and the notable ones include Hilbert curve, Peano curve, Serpiński curve and Lebesgue curve. Additional information on space filling curves can be found in [Sag94].

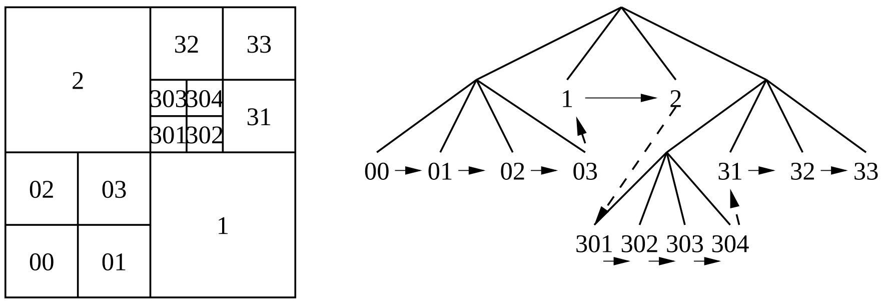


Figure 2.3: Position Codes in a quadtree (left) and the order of parsing in a tree structure (right).

in the order of Morton index. The binary sequence encoding is parsed to identify the position from the bit and thereby insert it into the corresponding child node of the octree.

Algorithm 1 Method to generate octrees.

```
Require: set model to half-space
 1: Func: refine()
 2: get cell-type from model
 3: if (¬ max-depth) then
 4:    if cell-type = on then
 5:       mark on
 6:       call refine()
 7:    else
 8:       if cell-type = in then
 9:          mark in
10:       else
11:          mark out
12:       end if
13:    end if
14: end if
```

Graph Theory

Many combinatorial optimization problems can be formulated as problems in a graph[2]. A graph $G = (V, E)$ consists of a finite set of nodes V and a finite set of

[2]Graph theory is considered to have first originated when a Swiss mathematician Leonhard Euler proved that there exists no route such that all the 7 bridges in Königsberg can be crossed exactly once – the "Königsberg Bridge Problem"

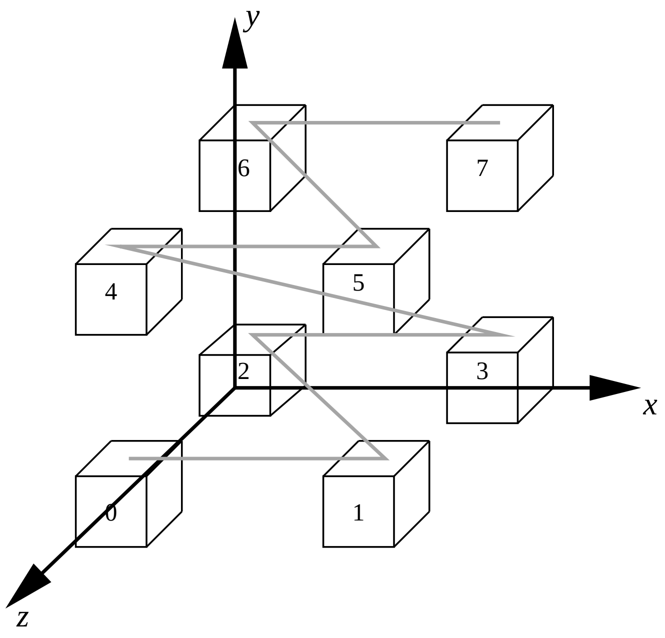


Figure 2.4: Illustration of the Morton index sequencing in an octree.

edges E. Each edge e of the graph consists of two end nodes u and v and is denoted as uv (or $e = \{u, v\}$). The edges of the graph may or may not denote a direction. For example, a graph of a street network using directed graph to indicate One-Way streets. In an undirected graph, there exists no difference between uv and vu, whereas in a directed graph both uv and vu are distinguished separately based on their existence. If the vertices $uv \in e$, then u and v are said to be adjacent vertices. The neighborhood of a vertex v, denoted by $N(v)$, is the set of vertices adjacent to v: $N(v) = \{x \in V | vx \in E\}$. The degree of v, denoted by deg(v), is the number of edges incident with v. A graph $G' = (V', E')$ is called a subgraph of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. When performing some search operation in a (large) graph and the region of search is known, a subgraph of the region is extracted before the search is performed. In that way, the computational effort decreases. An edge set $P = v_1v_2, v_2v_3, \ldots, v_{k-1}v_k$ is called a walk (or a $[v_1, v_k]$ walk). If $v_i \neq v_j \; \forall \; i \neq j$ then P is called a path or $[v_i, v_k]$ path. The length of a walk or path is the number of its edges and is denoted by $|P|$. If in a walk $v_1 = v_k$, then the path is a closed path as in a traveling salesman problem, where the salesman returns to the starting point after visiting all the cities.

Shortest Path Algorithm

The shortest path problem involves the search of a path between two vertices of the graph such that the sum of the weights of its constituent edges is minimized. With the use of a single-source shortest path (SSSP) problem, it is possible to define a start node and identify the shortest path to all other nodes in the graph. The commonly used algorithm to determine the shortest path between two points was proposed by E.W.Dijkstra in 1959 [Dij59]. Given a weighted graph $G = (V, E)$, real-valued weight function $f : E \to \mathbb{R}$ and one vertex $s \in V$, the single source shortest path problem identifies the shortest path from source s to all other vertices in V. The algorithm 2 shows the implementation of the Dijkstra's shortest path algorithm. The Dijkstra's algorithm determines the shortest path for positive weighted edges. The algorithm assume 2 sets of vertices namely S and Q, where S is the set of vertices whose distance is already calculated (begins as an empty set) and Q is the set of vertices whose distance is not yet calculated (begins with all vertices).

Algorithm 2 Dijkstra's algorithm to determine the shortest path from source s to all other vertices in V.

```
1: set $source = s$
2: $S = \phi$
3: $Q = V$
4: while $Q \neg empty$ do
5:    $u = \min(Q)$
6:    $S = S \cup \{u\}$
7:    for each edge $(u, v)$ from $u$ do
8:       if distance($u$)+weight($u, v$)<distance($v$) then
9:          distance($v$)= distance($u$)+weight($u, v$)
10:         $value[v] = u$
11:      end if
12:   end for
13: end while
    {To identify the shortest path between source $s$ and a vertex $t$}
14: $S = \phi$
15: $u = t$
16: while $value[v]$ do
17:    Insert $u$ to $S$
18:    $u = value[u]$
19: end while
```

The storage of the shortest path using simple arrays or linked list will cause the Dijkstra's algorithm to have a computational complexity of $\mathcal{O}(n^2)$. For sparse graphs, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a binary heap or Fibonacci heap as a priority queue. The best results are achieved with a Fibonacci heap and has a complexity of $\mathcal{O}(m + n \log n)$

Geometry Awareness

The integration of simulation into the geometry context requires a better understanding of the geometry scenario. Issues such as location awareness and geometry parameters must be incorporated into the simulation system. During the simulation, a constant necessity for geometry data arises. Therefore, the geometry parameters must be efficiently structured. The graph and the data structures explained above are used to efficiently store the necessary architectural data. The octree-based modeling explained above uses the graph data to build a location-aware system where neighbor search methods are employed to identify the happenings in the neighborhood. The concept of geometry awareness is explained in detail in chapter 3.

2.2 Scalability Issues

Scalability is a common issue when modeling systems that has the capability to grow larger. Scalability indicates the ability of the system to handle growing amounts of data with an insignificant loss in performance. That is, the system must be able to enlarge readily. In the pedestrian simulation framework described here, scalability issues come in the form of larger regions and larger number of pedestrians. Handling of such growth is possible by splitting the problems into smaller tasks and solving them separately. Finally the results must be integrated together without loss of any information.

These sub-tasks are generally performed in parallel on different computing machines. Often, there is a necessity to communicate between the sub-tasks during its execution. Parallel architectures and the use of parallel and distributed programming can support such operations. The major difference between a normal serial execution and a parallel execution is that the sub-tasks are executed in parallel with a hope of improving the speed and performance of computing. Unfortunately, it is not always possible to implement the tasks to function independently and efficiently in parallel. This is because certain tasks require information from other sub-tasks, which make the tasks to occur in a serial fashion. Amdahl's law suggests that there exists a certain amount of serial code in most tasks and the speed up of the system depends on the amount of serial and parallel code available in the task. Amdahl's law is given as

$$S_p \ := \ \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

where α is the fraction of serial code, which cannot be parallelized and p represents the number of processes. As the number of processors increase, the speed up does not necessarily increase proportionally. However, the Gustafson's law identified that the size of the problem is important when considering the amount of parallelization.

In a very large simulation scenario containing many buildings and pedestrians, the computing and performance issues can be improved by decomposing the simulation problem into smaller regions (e.g., simulating each building separately) and simulating them separately. Care must be taken that the partition is performed efficiently such that the communication between the tasks, thereby the amount of serial process is minimized. Efficient graph partitioning techniques may be used to decompose the simulation problem [KL70, SKK00, BGOM03].

2.3 Discrete Event Simulation

Even though the technology advances at an enormous rate, modeling complex systems remains to be a major challenge. For example, systems such as manufacturing systems or business process modeling require efficient organization of the structure to optimize the throughput or production, traffic control systems needs to handle the flow of a large number of vehicles and a cargo department in a large airport must make sure that the system carries the right baggage to the right aircraft. Modeling of such systems can be achieved by using a number of methods and tools, one among them is simulation. A simulation is a technique, where models that resemble the original or the proposed system is built and the behavior or the performance of the model is studied.

A discrete event simulation is one such method to model a system to study a time-based or an event-based behavior of a system. Like in any other simulation method, a discrete event simulation involves collecting of data, modeling the system to the desired level of detail, validate the model, make initial test runs, run experiments for the required input parameters and analyze the results generated. A discrete event simulation typically progresses as and when the events occur. For example, in a post office, some of the events include arrival at a specific queue, departing the queue and entering the service counter, activity at the service counter, joining another queue, etc. A typical queuing system consists of a queue of certain or infinite length and one or more service counters. Customers or entities arrive in the queuing system, wait in the queue until the previous entities are served and a service counter is available, execute the task at the service counter and depart from the queuing system. Such a system can be modeled using a discrete event simulation method.

Entity Management

The term entity is used to designate a unit of traffic (a transaction). Entities change their state by responding to events. An event is a happening that changes the state of a model (or system). There are two types of entities, namely external entities (which are created explicitly by the modeler) and internal entities (which are created and manipulated implicitly by the simulation software itself). A resource denotes a system element that provides a service to the entities. Resources often have a limitation with their capacity and the entities must therefore wait if the resource is occupied by other entities. The movement of the entity through the system is carried out by an operation. A process-oriented or a transaction-oriented approach simulates the system by focusing on the operations performed on each entity. During a simulation run, the entity takes different states. Some of the states are active state (the state of the currently moving entity), ready state (entities waiting to enter the active state), time delayed state (if it is not possible to perform an operation on the entity and the operation is postponed to a later time) and the conditional delayed state (if there are certain conditions bound to occur before the operation can be performed).

Clock

The simulation clock is an important component to control the simulation run. The clock (is not a wall-clock) is directly associated with the simulation run. The clock keeps track of the passage of the simulated time. The clock advances in discrete steps (fixed time slices or always the next event). Once all activities are executed at a given simulation time, the clock is advanced to the time of the next earliest event to execute all the activities at that time.

Input Modeling

The input modeling function provides the input data for the simulation. In a simulation, the behavior of an existing model is imitated and the resulting system is simulated. Therefore, input data, as good as the realistic behavior is needed to simulate the system. Statistics from an existing (real) system are collected and these statistics are translated to mathematical functions that produce random numbers, whose pattern resembles that of the real system. Different distribution functions are used to model the input data and these distributions transform a uniformly distributed random number to a specific distribution.

Execution

Once the parameters are initiated within the model, the system is simulated or executed. The execution function is responsible for the advance of the time, generate the events, schedule the events and execute the events as the simulation clock advances. Execution is probably the most complex part in a discrete event simulation. The key of the execution lies in the efficient structuring of the list of entities and events, as well as the scheduling of the events. Different event lists are created and all new events generated are inserted into the event list in the chronological order of their occurrence. As the simulation clock advances, the events from the event list, which are bound to occur at the time of the clock, are executed. The execution terminates once all the events are complete or once the termination condition has reached.

Analysis

The purpose of simulation is to analyze the behavior of the system for a given set of parameters. Therefore, during the simulation run, the states of the system are constantly saved as output data. The behavior of the system can be determined by analyzing the output data. For example, in a queuing system, properties such as the average waiting time, average throughput of the system, average service time, etc., are analyzed by performing fitness tests, acceptance-rejection tests, etc., on the data obtained from the output of the simulation.

In this thesis, the pedestrian behavior is simulated using the `SIM` simulation software library. The appendix A presents an implementation of an elementary $M|M|1$ queuing system using the `SIM` simulation library.

Chapter 3

Geometric Modeling: A Foundation for Pedestrian Simulation

Where there is matter, there is geometry
– **Johannes Kepler**
(Ubi materia, ibi geometria.)

Right from stunning special effects in movies or computer games in the field of entertainment to realistic scientific simulations such as performance of an aircraft structure or interaction of molecules, the study of objects in terms of its geometry structure is inevitable. Geometric modeling is often linked with graphical representation of 3D objects. Geometric modeling is also the study of computational structures that capture the spatial aspects of the objects, which are of interest to an application. Computational geometry or in other words CAGD (Computer-Aided Geometric Design) deals with the study of classical geometric algorithms and uses computational and mathematical methods to solve them. CAGD primarily deals with the construction and representation of free-form curves, surfaces, or volumes. Beginning from the renaissance artists, geometric modeling has come a long way.

Huge difficulties existed in efficiently plotting and drawing curves. Several mechanical methods and even French curves[1] were commonly used to draw curves. One of the foremost applications was the design and construction of automobiles. The development of Bézier curves and surfaces and the de Casteljau algorithm independently by P. Bézier at Rénault [Béz74, Béz76] and P. de Casteljau at Citroën [dC63] respectively, were a major breakthrough in the representation

[1]A French curve is a template made out of plastic, metal or wood and composed of many different curves. It is used in manual drafting to get smooth curve of varying radii.

of polynomial curves and surfaces [Far02b, Far02a]. The significance of geometric modeling can today be attributed to the growth in the field of computer graphics.

Tremendous improvements have been made in the field of graphic hardware and the complexity of modern 3D graphics hardware has today far exceeded the capabilities of a general purpose processor. 3D models are today generated using computer graphic tools, mathematical equations or even with the use of a 3D scanner to reproduce realistic objects. The so-called 3D-engines translate the model into an image that can be displayed on a monitor. Realistic images are also generated by using textures and lighting effects (ray-tracing method). In spite of such an advance, graphical modeling is still time consuming when it comes to producing high quality images. For example, the animated feature film "Cars" produced by Pixar Animation studios required 17 hours to compute a single frame of the movie. With a sophisticated network of 3000 computers and with a $4\times$ computing power as used for its predecessor movie "Incredibles", it took many days to render a single second of finished film. Continuous advancement in computer graphics are regularly presented at the ACM's SIGGRAPH annual conference.

Geometric modeling is virtually seen in every scientific and engineering application such as Geographic Information Systems (GIS), medical imaging, scientific simulation and visualization, architectural and structural design.

In the reference model used in this thesis, the geometry parameters that are necessary for the pedestrian simulation must first be formulated and structured. The geometric modeling plays a vital role in supporting the pedestrian simulation model. Traditional simulation models with a non-geometric background now-a-days sometimes need to be integrated into a geometry context, which in turn provides spatial context for non-spatial data. For applications that require spatial context, it becomes more and more important to extensively model the geometry data as well as integrate the simulation model into the geometry context. The geometric modeling is used to derive the necessary architectural details such as location of the rooms in the building, staircases, distance between two walls, etc., from the building used in our reference scenario. The architectural parameters are also necessary to establish a communication between the pedestrian model and the surrounding environment, where the simulation takes place.

This chapter deals with the geometric models, which support the pedestrian simulation, as well as the associated algorithms and methods that are used for constructing and querying the models. This chapter begins with an overview of geometry representation and the technical concepts that lie underneath. Then, the chapter focuses on various methods and data structures needed to extract the geometry data for the pedestrian simulation. This includes the concepts of representation of the geometry model in terms of a graph structure, path-search algorithm performed on the graph, partitioning the geometry using octrees, hierarchical storage methods and finally the communication with the pedestrian simulation. The octree model is used as a base for a pedestrian navigation system. Therefore, the concepts of position identification, neighbor search, etc., are also presented.

3.1 Geometry Representation

In order to visualize a realistic scene, the scene must be efficiently represented in terms of geometry model. One direct approach is to represent the scene is a 3D raster format, where the entire scene is partitioned into small cells. Each cell contains the information if the cell actually belongs to the body of the scene or not. The quality of the output model depends on the level of discretization and with the increase of the number of cells, the memory requirements increase exponentially. Another method to represent solid bodies is the use of octrees, where each cell is partitioned further into 8 smaller cubes only (as shown earlier in 2.1) if the approximation level has to be reduced. Use of octrees reduces the memory requirements.

It is also possible to combine elementary or existing volume models to build new models. Boolean set operations such as union, difference, and intersection can be used to combine solid bodies to produce new models. These operations are 3D equivalent of a 2D boolean operation. Applying an ordinary boolean set operation to two solid objects does not necessarily yield a solid object. Therefore, regularized boolean set operations can be used to yield new solid model by combining existing solids. Regularized boolean set operations are described in [Req77].

Since volume models capture the entire geometry of the solid, volume are often used in modeling physical processes such as collision detection of objects, finite element method, fluid dynamics, etc. However, due to the enormous amount of memory needed, volume models are seldom used for visualizing realistic scenes. Alternatively, a surface model is used. In a surface model, the visible parts of the scene are represented using free form surfaces such as Bézier, NURBS surfaces or polygon meshes such as triangulation. Curved surfaces can be discretized, represented in the form of triangles, and are rendered to produce curved surfaces. For a smooth curve, it is important to use as many triangles as possible. Detailed information on geometry representation can be found in [BGZ04].

The reference model, from which the geometry parameters are extracted, is represented as a surface model. Since we are interested in the dimensions of the paths and rooms (or in other words the dimensions of the space available) and not in the solid model such as thickness of the walls or roof, a volume-oriented model is not used here. The reference model is represented as a triangle mesh and each triangle is then rendered to produce the scene of the reference model. Figure 3.1 shows the reference model with smooth surfaces and the triangular mesh lying under it.

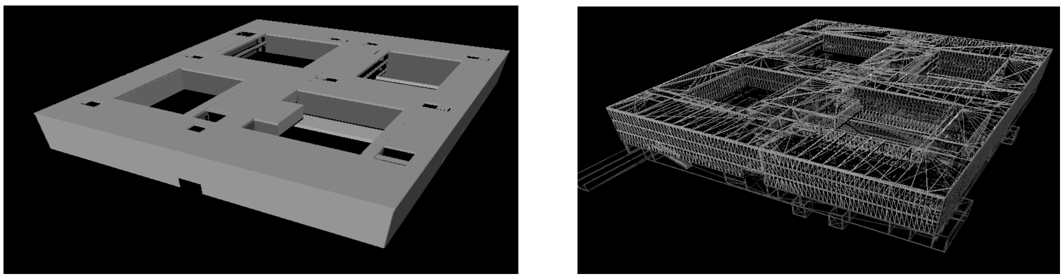


Figure 3.1: Surface model (left) and the triangulation representation (right) of the reference model.

3.2 Geometry Parameters for Pedestrian Simulation

In general, pedestrians do not move arbitrarily, but they typically have a motivation: to execute a task at the destination (working, shopping, relaxing, etc.) [Wer96]. This process involves movement within the building, execute the task and also wait at the destination when it is not possible to execute the task. A task here defined as any activity that does not involve a movement along the path. A task can virtually be any activity: relax on a bench or watch the notice board or wait in the queue to pay at the cash counter. Therefore, there are two pedestrian activities modeled in the reference scenario, namely movement along the paths and execution of a task at the destination. During these activities, the pedestrians interact not only with other pedestrians participating in the scenario but also with the geometry of the environment. The geometry parameters such as the paths, location of the destinations, type of the paths, etc., are therefore needed to model the pedestrian behavior. From the CAD data of the reference model, we extract the necessary geometric parameters and the architectural details, and use the data to model the pedestrian behavior in the given scenario. Architectural parameters such as the thickness of the walls, pipelines, position of the windows, etc., are however irrelevant for the pedestrian simulation here and are hence ignored.

The geometric parameters, once extracted need to be structurally well organized such that the pedestrian simulation is performed efficiently. The geometry data are interlinked together into a graph consisting of edges and nodes, where edges represent the paths and certain nodes represent the destinations. The graph data is itself a nested linked list structure, where each edge is linked to the nodes on both ends of the edge and each node is linked to the set of edges that connect to this node. The graph data is then relational. That is, if given a node, it is possible to identify all the edges connecting it and vice versa. The remaining geometric parameters extracted such as the capacities, types of paths or rooms, etc., are also embedded within this graph. All pedestrian algorithms implemented here refer to this single graph structure to access any parameter concerned with the geometry of the scenario. In this section, the algorithms and data structures that are used to extract, manipulate and access the graph are introduced and the various algorithms implemented are discussed.

3.2.1 Extraction of Paths and Destinations

A pedestrian visit planning involves finding a path between two points and executing the task at the end point. Therefore, the destinations and the paths that lead to the destination must first be identified from the scenario. Typical use of such data can be found in a building guidance system – for example, the FIFI[2] (Fakultätsinformationssystem für Infoterminals) at the Computer Science building of the Universität Stuttgart [GSG04] – where routing information to the destination is provided to visitors. Such systems are typically built for a specific scenario. In the case of FIFI, a graph connecting all rooms is built manually such that the path between the Infoterminal and the destination can be calculated using a path-search algorithm. The web-interface then displays the path graphically (see Figure 3.2).

The graph of such an Infoterminal is not sufficient since the graph is scenario specific and the architectural details (path width, stairs or ramp, room capacity, etc.) are missing. Therefore, the graph, along with the required architectural parameters, must be extracted automatically from any generic CAD model of a building.

The Pathscan [Dre03] tool, developed in our group, is one such tool that can be used to automatically extract a graph from the given CAD model. Pathscan tool uses the graphic hardware to identify the depth buffer from a surface-oriented

[2] `http://infoterminals.informatik.uni-stuttgart.de`

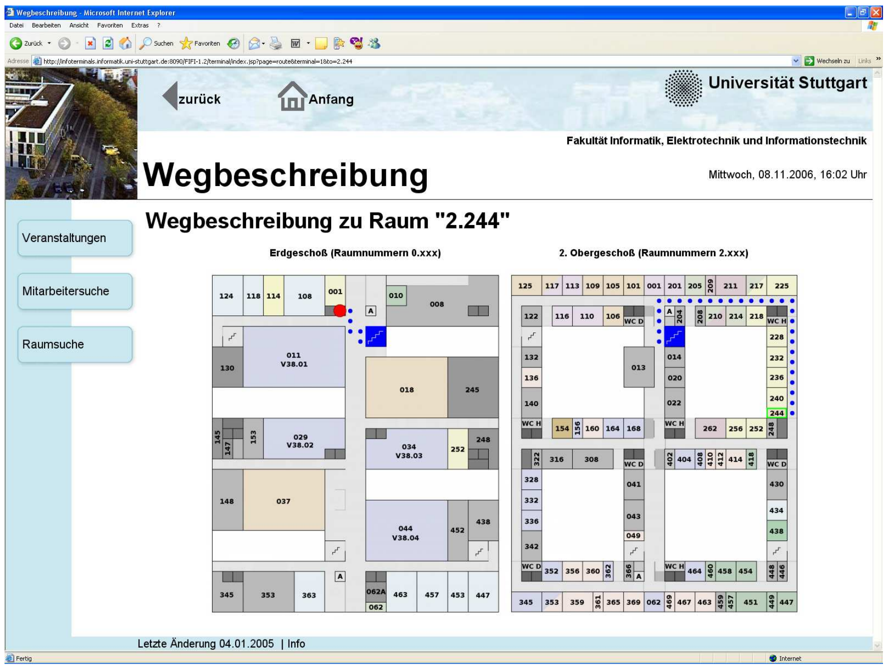

Wegbeschreibung - Microsoft Internet Explorer
zurück
Anfang
Universität Stuttgart
Fakultät Informatik, Elektrotechnik und Informationstechnik
Wegbeschreibung
Mittwoch, 08.11.2006, 16:02 Uhr
Wegbeschreibung zu Raum "2.244"
Veranstaltungen
Mitarbeitersuche
Raumsuche
Erdgeschoß (Raumnummern 0.xxx)
2. Obergeschoß (Raumnummern 2.xxx)
Letzte Änderung 04.01.2005 | Info

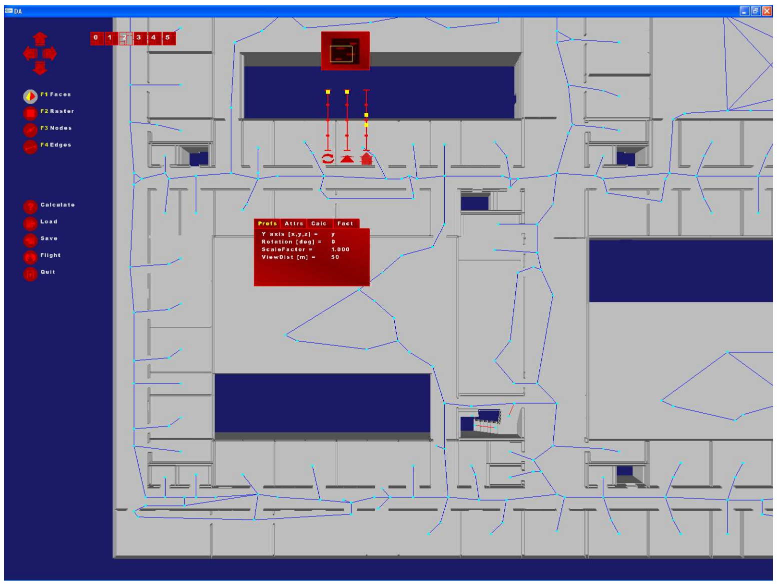
Figure 3.2: Room search using the FIFI Infoterminal.

model to build the graph (the vertical faces are not visible to the user). The model is projected along the x-z-plane and the model is parsed in the y (upright) direction from top to bottom and slices of the model are made. The complexity of the method depends on the number of raster points along the x-z-plane (not every point is considered but the size is decided based on practical use, e.g., Door width of 80 cm) and the number of slices s (a slice is made as soon as a surface is detected along the x-z-plane since a pedestrian movement is possible only on top of a surface, e.g., stairs, floors). The angle of inclination of the polygons along the y axis is to be considered when making slices (e.g., walls are placed at 90° and can be ignored, but a ramp with an inclination of 5° cannot be ignored). The complexity therefore is $\mathcal{O}(n_x\ n_z\ n_s)$.

By using the distance transform method to skeletonize[3] the slices, closed rooms

[3]Skeletonization is a process of reducing foreground regions in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels. Skeletonization can be done in two ways. First, the boundary pixels are eroded by preserving the end points of the line segment until no more eroding is possible. What is left over approximates to a skeleton. Another approach is the use of distance transformation method. In a distance transformation method, the binary image is transformed to a gray scale image except that the gray level intensities of points inside the foreground regions are changed to show the distance to the closest boundary from each point. The skeleton then lies on the singularities in the distance transform.

and thereby, the possible nodes are identified. The graph is then built using these nodes. It is important to check if the node can accommodate a pedestrian (height dimensions) and a transition to the adjacent node is possible (height difference, blockage between 2 nodes and the minimum distance to the wall). The Pathscan tool was originally developed as an intelligent path-search system for a given architectural model. It therefore identifies all walking paths from the given CAD model. The graph is then used to navigate a pedestrian (a virtual tour) to the desired destination.



```
Y axis [x,y,z] =    y
Rotation [deg] =    0
ScaleFactor =       1.000
ViewDist [m] =      50
```

Figure 3.3: Snapshot of the Pathscan tool with the graph extracted automatically.

As seen from Figure 3.3, the graph extracted from the Pathscan tool is very extensive and irregular. This is because the tool identifies all possible paths, where a pedestrian movement exists and interconnects every adjacent node. Using such a graph to model the pedestrian behavior is complex and computationally intense. Therefore, the graph is reduced by merging certain edges together such that the properties of the graph suitable for the pedestrian simulation are still preserved. In a reduced graph, all edges that occur within a confined region are merged into a single edge and this edge contains the properties of the other merged edges.[4]. An edge will have a closed boundary around itself such that the pedestrian movement, anywhere in the path surrounding the edge, can be

[4]Two main properties are the type of the path and capacity. The path type is unique for all edges since they lie within the same region. The path capacity is calculated from the available area by considering the distance to the wall

modeled using the parameters of the edge. Now, each edge will now support only uni-directional or bi-directional pedestrian movement. For example, consider a wide long corridor, which has doors on either end of the path (see Figure 3.4). The graph extracted from the Pathscan tool consists of many paths between the two doors such that the entire area of the path is included in the graph. The reduced graph contains a single edge that passes through the center of the corridor with edges to the doors along the paths. The properties (path dimensions and path types) of all edges along the path are merged into this single edge.

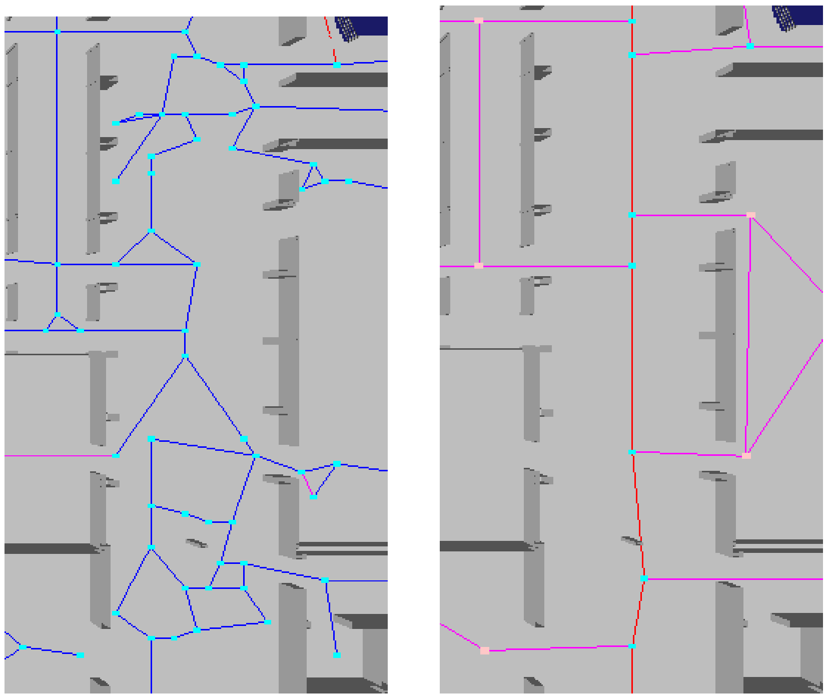
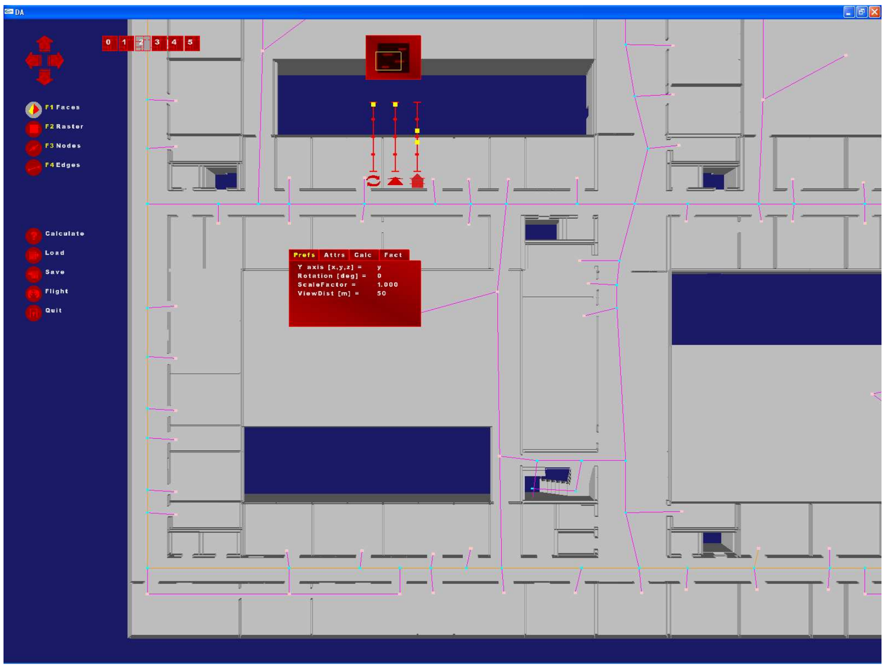


Figure 3.4: Extensive graph from the Pathscan tool (left) and the reduced graph for the pedestrian simulation (right).

From the geometric specifications of the path, the tool can identify the type of the path ramp, stairs, etc., (is the path suitable for a wheel chair?). However, difficulties exist in identifying escalators or elevators. The CAD model generally does not differentiate the path types (private path, emergency exits) or provide room details. The tool is therefore built to be flexible such that it is possible to

- modify the graph (adding or removing of new nodes and edges),
- define room numbers,
- define edge properties (path type), and

- export the graph and model in other formats (XML, VRML, etc.).



```
Y axis [x,y,z] =    y
Rotation [deg] =    0
ScaleFactor =       1.000
ViewDist [m] =      50
```

Figure 3.5: Snapshot of the Pathscan tool with the reduced graph.

The final graph, as shown in Figure 3.5, contains a set of edges and nodes spread across a 3-dimensional space and each node therefore contains the co-ordinates of its location in the CAD model. Since the Pathscan tool also exports the extracted graph into an universal XML (Extensible Markup Language) format, the XML version of the graph can easily be read and manipulated[5] to generate the reduced graph. For the pedestrian simulation, it is important to include several properties to the graph such as type of the path (stairs, ramp, private path, etc.), capacities of each path, the distance between the walls, the exact description of the destinations, capacities of the destinations, etc. The properties of paths and destinations used for pedestrian simulation are described in the next part.

3.2.1.1 Path Properties

The path properties needed for modeling pedestrian behavior are listed as follows

1. **Length**: The length of the path is necessary to calculate the time it takes for a pedestrian to walk along the path. The length of the path is automatically

[5]The Libxml2 library – the XML parser and toolkit of gnome – is used to parse and manipulate the XML data (`http://xmlsoft.org/`)

calculated using the co-ordinates of the two nodes, (x_1, y_1, z_1) and (x_2, y_2, z_2) to which the edge is connected. The path length $l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

2. **Type**: There are two different requirements necessary for the type of the path, namely the geometrical layout of the path and path restrictions for certain pedestrians. The geometrical layout of the path means architectural properties such as staircase, ramps, pathways, narrow path through a door, etc. These properties are automatically identified during the graph extraction. Path restrictions are derived from properties of the scenario, where the pedestrians are modeled. Properties such as private paths (restricted to certain group of pedestrians), emergency exits (not in normal use), paths inside rooms, restricted areas or ducts, etc., are not defined in the CAD model. Therefore, these restrictions are configured separately and are then included in the graph.

3. **Capacity**: The capacity is fundamentally driven by the area of the path. The area is calculated by using the length of the path and the distance between the walls throughout the path. Generally, the capacity of the path denotes the maximum number of pedestrians that can possibly fit in the given path (at $0.5m^2$/ped [Gip87]. The capacity is necessary to model the pedestrian behavior, which shall be discussed in chapter 4.

4. **Speed**: The walking speed of the pedestrian along the path is influenced by the characteristics of the pedestrian himself. Various statistical and experimental methods used to determine the walking speed of the pedestrian along the path will be discussed in chapter 4. However, the geometrical aspects of path also play a significant role in influencing the walking speed of the pedestrians. Physical experiments were made to measure the average walking speed along different path types in the reference model used here. For example, it was found that the walking speed along the ramps and flat pathways were more or less consistent whereas the walking speed along the staircase is roughly half the speed of walking along a pathway. Such parameters are also taken into account for pedestrian simulation.

3.2.1.2 Destination Properties

The destination properties needed for modeling pedestrian behavior are listed as follows

1. **Type**: The type of the destination here denotes the purpose of the destination (e.g., WC, shop, restaurant, etc.). The type does not have any relation

to the geometry of the data. However, the type information is necessary for a pedestrian to decide whether or not he has to visit the destination. The type also gives the pedestrian model an idea how long it would take to execute the task (the service time is calculated using the type value). Each node, which is a destination, is associated with a unique ID (or the room number). There can be more destinations associated with the same type. The handling of such destinations is dealt with in chapter 4. The type of destination cannot be automatically derived and they must therefore be defined manually using the Pathscan tool.

2. **Capacity**: Unlike the capacity of the path, the capacity of the destination does not denote the physical capacity, but instead the number of customers who can be served in parallel. Again, such information is dependent on the destination and therefore cannot be automatically generated. The capacity of the destination must therefore be defined manually using the Pathscan tool.

The path and the destination properties are embedded within the XML data of the graph and rooms respectively. The layout of the XML data that contains the graph and room data are presented along with some examples in chapter 6.

3.2.2 Graph Representation

For the reference model, we use a graph structure to represent the paths and destinations of a CAD model. The properties associated with each path and destination and path are also embedded into the graph. Pedestrian simulation and other applications such as path-search algorithms, navigation systems, etc., are performed using this graph. Figure 3.6 shows a snapshot of the graph of the reference model used here.

3.3 Pedestrian Navigation

So far, the necessary parameters required for modeling and simulating pedestrian behavior in the reference scenario were collected. In the reference scenario, the pedestrian is also navigated through a series of destinations. The geometry and the graph data can be used to build a pedestrian routing and navigation system. A pedestrian navigation system – using the geometry and graph data – is built in the following steps.

- Initially, the graph data is hierarchically structured in an octree.

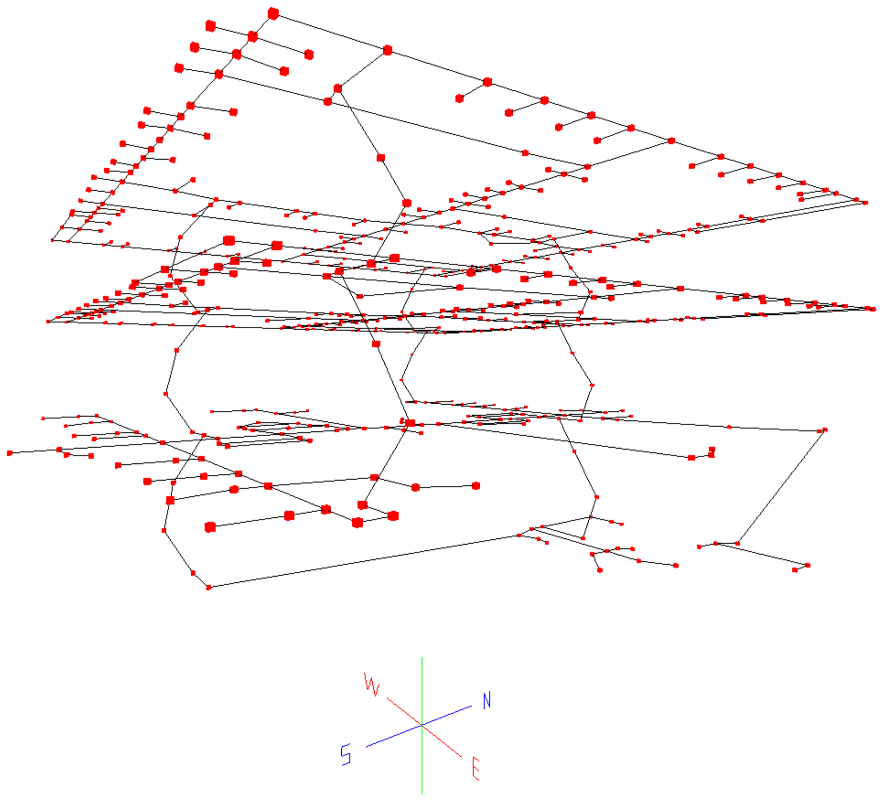


Figure 3.6: Snapshot of the graph of the reference model used here. The paths in the scenario are connected using the edges of the graph.

- The current position of the pedestrian, as well as the destination is then identified from the octree structure.
- Finally, by considering the pedestrian priorities, efficient path-search methods are used to determine the optimal path (or a route) to the destination.

In this section, we discuss each of the above listed steps in detail.

3.3.1 Hierarchical Modeling

The hierarchical modeling of a geometry structure (or space partitioning) is the process of partitioning the geometry objects into two or more disjoint subsets. The partition is done such that the geometry is partitioned into non-overlapping regions. Any point in the geometry shall then lie exactly in one of the partitioned geometry. The partition is done based on hierarchy and recursion. Some examples of such partitioning include BSP-Trees, Quadtrees, Octrees, kd-trees, triangulations, etc. In general cases, a tree structure is used to represent the partitioned structure. The data is then organized in the leaves of the tree. Such hierarchical geometry structures facilitate certain geometric queries such as collision detection of objects, neighbor search, etc. [Sam84] surveys the hierarchical data structures, specially quadtrees and their related structures.

Here, octrees are used to partition and structure the geometric model hierarchically. The tree structure is then used to identify the position of the pedestrian and navigate him to the destination.

3.3.1.1 Octrees

In section 2.1, the octree structure used to represent geometry data was presented. Here, we build a point-based octree of the graph by representing the nodes of the graph in an octree. The method to build the octree from a graph is explained as follows.

Octrees can be used to represent point data or set of elements that is spread across a 3-dimensional space $P \subset \mathbb{R}^3$. The points are inserted into the octree and for each point inserted, the octree is built recursively until each cell in an octree consists of utmost 1 element. During this process, octrees might contain empty cells. [Mun06] proposes an efficient method to insert the point data into the octree structure and [NRMB05] uses this method to structure the p-version of finite elements hierarchically using the octree.

The set of points P is structured in the octree using the Lebesgue curve sequence. The octant, where a single point $p \in P$ should lie is determined using a simple arithmetic operation. The algorithm uses binary form to identify the octant to which the point p belongs. Three bits (b_1, b_2, b_3) are used to represent the Morton index from '0' to '7' in the form of binary numbers $(000, 001, ..., 111)$. Figure 3.7 shows an octree of level 1 in the range of $[-1, 1]$.

The signs of the co-ordinates are used to determine the octant to which they point. For this purpose, the points in the point set are normalized such that the entire set of points can be represented in a cube space $f : \mathbb{R}^3 \to [-1, 1]^3$. Now, the co-ordinates of each point $(\pm x, \pm y, \pm z)$ lie either on the positive octant or the negative octant. The bits are set based on the sign of each of the co-ordinates (see algorithm 3)

Therefore, each point is then inserted into the corresponding (*node-number*) node of the tree. If the leaf node already contains a point, then an octree is built recursively within that node and both the points (old as well as new) are shifted to their corresponding octants. Figure 3.8 shows an example of a quadtree for P set of points.

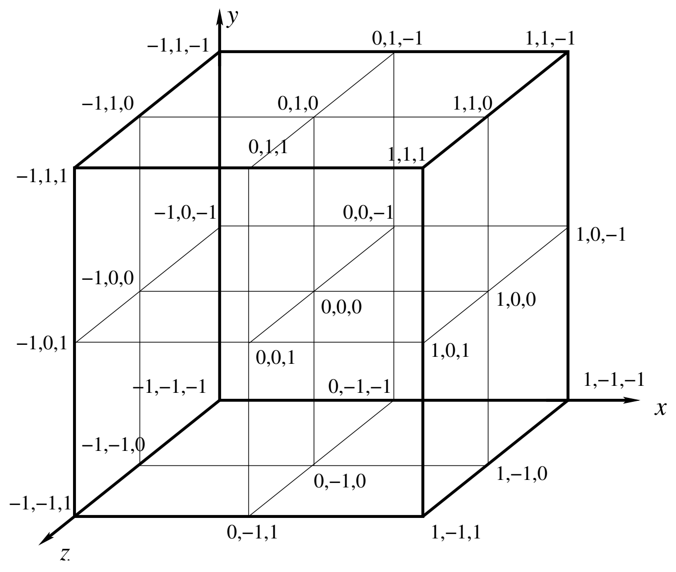


Figure 3.7: Octree with co-ordinates of each vertex in the range $[-1, 1]$.

The graph of the reference model used here is structured similarly in an octree structure. For this purpose, the nodes of the graph are parsed and the co-ordinates, along with the `NODE_ID` are extracted from the graph structure. These node points are then inserted into the octree structure. The octree is then used to identify the position of the pedestrian as well as search the neighboring nodes for possible destinations, which shall be discussed in the following section. The geometric model used in the reference scenario consists of about 540 node points. The algorithm takes 0.024 seconds to build an octree with a depth of 5 for the graph of the geometric model.

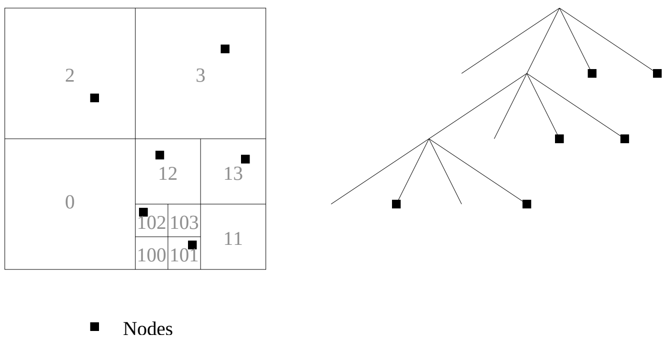


Figure 3.8: Quadtree and the tree structure consisting of P set of points.

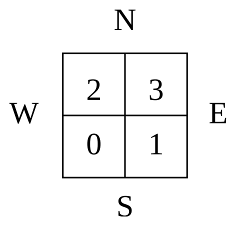
Algorithm 3 Method to determine the octant of a point with co-ordinates (x, y, z).

```
1: if $x \geq 1$ then
2:     set $b_3 = 1$
3: end if
4: if $y \geq 1$ then
5:     set $b_2 = 1$
6: end if
7: if $z < 0$ then
8:     set $b_1 = 1$
9: end if
10: convert $(b_1, b_2, b_3)_2 \rightarrow (\texttt{node\_number})_{10}$
```

3.3.1.2 Position Identification and Neighbor Search

Typical simulations that use octrees or similar structures will have their model partitioned into several parts. To handle these sub-models separately, it is important to efficiently identify the desired model in the structure. The term *Location Awareness* refers to the ability of identifying individual model components. In the reference model, the pedestrian arrives in the scenario with some mobile navigation device in hand. The position of the pedestrian must therefore be identified and translated to the graph data. Since the pedestrian can arrive in any of the octant of an octree, the neighbor cells must also be searched efficiently to identify graph nodes that are situated nearby.

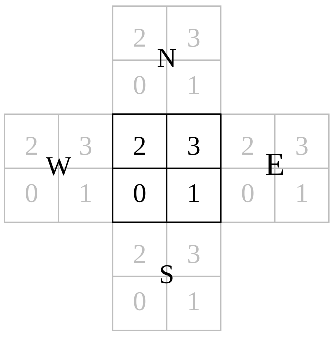
Identification of neighboring cells in an octree is possible with the use of neighbor-cell matrix [Fra00]. Table 3.1 shows the neighbor-cells in a matrix for a quadtree in all directions.



	0	1	2	3
W	-	0	-	2
E	1	-	3	-
S	-	-	0	1
N	2	3	-	-

Table 3.1: Neighbor cell coding (left) and neighbor cell matrix (right).

For example, the position code of the cell north of the cell with Morton index 1 is always a 3. In Figure 3.11, the position code of the cell north of 031 can be found using the table as 033. However, to determine a cell east of 031, we get $031 \xrightarrow{E2\mapsto -} 03 \xrightarrow{E1\mapsto -} 0 \xrightarrow{E0\mapsto 1} 1$. This means that the identification of the position of the node belonging to the same level is not possible. Therefore a complete neighbor cell matrix is required such that the position code of the neighbor cell can be identified even if the neighbor cell is outside the 4 quadrants [Ber02]. Table 3.2 shows the complete neighbor cells such that cells outside the quadrants can also be identified.



	0	1	2	3
W	1$\hookleftarrow$	0 ✓	3$\hookleftarrow$	2✓
E	1✓	0$\hookleftarrow$	3✓	2$\hookleftarrow$
S	2$\hookleftarrow$	3$\hookleftarrow$	0✓	1✓
N	2✓	3✓	0$\hookleftarrow$	1$\hookleftarrow$

Table 3.2: Complete neighbor cell coding (left) and neighbor cell matrix (right).

The ✓ shows that the neighbor cell code is definite and $\hookleftarrow$ shows that the neighbor cell code belongs to a different quadrant and it may or may not exist. Now, we determine the cell east of 031 we get 031 $\xrightarrow{E2\mapsto 0\hookleftarrow}$ 030 $\xrightarrow{E1\mapsto 2\hookleftarrow}$ 020 $\xrightarrow{E0\mapsto 1\checkmark}$ 120. Thus the complete neighbor cell matrix can be used to determine the position of the neighbor cell belonging to the same level. Similar approaches for neighbor search in an octree can be seen in [Bha01, FP02, YB06].

We use an octree-based system to identify the exact position of the pedestrian. Position identification or *Location Awareness* is increasingly becoming common with the cheap availability of location-sensing devices. In the field of ubiquitous computing, location awareness opens the possibility of improved services: Where is the next bus stop? how to find the next restaurant and also reserve a table? how to find my way home and how long will it take to drive? Several electronic devices exist that can determine the current geographic location, both indoors (RFID, WLAN, sensors) and outdoors (GPS[6] systems). There are three different techniques used in sensing the location [HB01a].

[6]GPS or Global Positioning Systems consists of a set of about 2 dozen satellite navigation systems orbiting around the earth. Initially launched for military purposes, GPS is now commonly used by civilians for a wide variety of applications. GPS consists of an atomic clock and a simple computer to synchronize and transmit its current position. A GPS receiver receives the GPS signal from at least 3 satellites to calculate its position using the triangulation method. Availability of a 4th satellite signal will be useful to compute the altitude as well. Modern GPS devices can have an accuracy of up to 1-2 meters by using differential GPS.

- *Triangulation* uses the geometric properties of triangles to compute object locations
- *Proximity* measures nearness to a known set of points
- *Scene Analysis* examines a view from a particular vantage point

A survey of different location systems is made in [HB01b]. In the reference model, we assume that a pedestrian arrives with some mobile navigation device in hand and we further assume that the location devices can deliver the co-ordinates of its current position $p_{(x,y,z)}$ in the scenario. We then use this data to identify the exact location within the graph using the octree.[7]

The octree containing the list of nodes of the graph is already generated. Now, the exact location of $p_{(x,y,z)}$ within the octree is identified and the position code of the octant to which it belongs to is identified. The search is made as follows. Let $p_{(x,y,z)}$ be a point $p \in P$. We now insert this point in the existing octree. Therefore, the new point is normalized within $[-1, 1]$ and by using the algorithm described above, the octant in which the new node is to be inserted is identified and the program terminates just before the insertion. We now have the position code of the octant to which $p_{(x,y,z)}$ belongs to. The node that already exists in the octant is then identified. There are two possible cases of identifying the existing nodes.

***Case 1*:** If $p_{(x,y,z)}$ belongs to an octant, where a node already exists, the current node is returned. This implies that $p_{(x,y,z)}$ lies in the voxel of the node that is returned (Figure 3.9).

***Case 2*:** If $p_{(x,y,z)}$ belongs to an octant, where no node exists, the neighboring octants in the same level must be searched. Therefore, the parent of the current node is searched and the children containing a node are returned (Figure 3.10). The resulting nodes are compared with $p_{(x,y,z)}$ to determine the closest node.

[7]It must be noted that there may often be errors in identifying the exact position using position identification devices. A WLAN-based position system for example uses the triangulation method to identify the current position of the device. The location of the access points within the scenario is known and the WLAN receiver measures the signal strength to identify the distance to the access point. The signal strength and noise depends on many factors such as wall thickness, construction materials used, other interferences, etc. Sufficient tolerance levels have to be built to improve the precision of the devices. [EFS03] surveys the available position identification devices and compares their performance and accuracy.

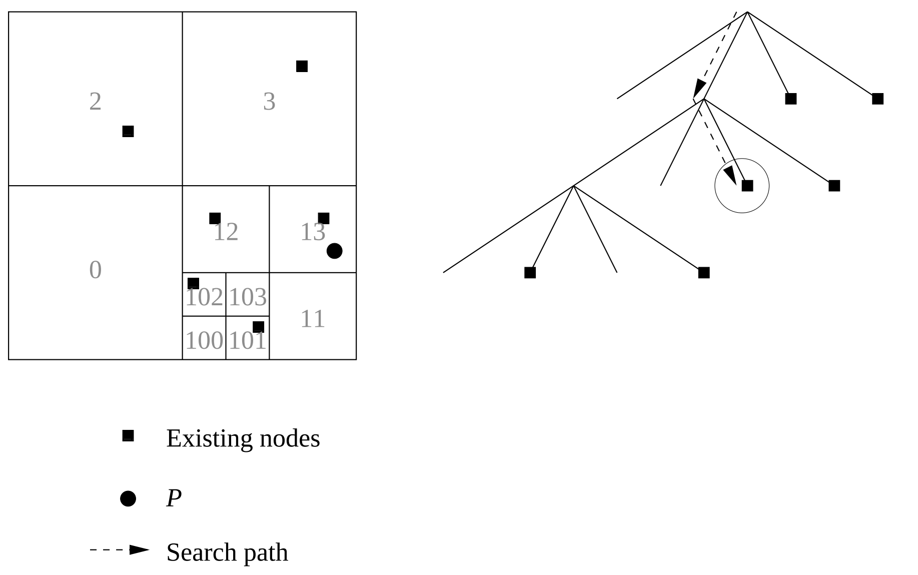


Figure 3.9: Tree search when $p_{(x,y,z)}$ lies in a quadrant, where a node exists.

However, it must be noted that the nodes stored in the octree structure are not necessarily positioned in the center of the voxel. This means that the position $p_{(x,y,z)}$ and the node determined from the above explained steps are not necessarily the closest neighbors. It is therefore necessary to parse the neighboring octants, determine the nodes on those octants and find the node closest to position $p_{(x,y,z)}$. The objective is to determine if there are other nodes closer than the identified node. To determine the boundary condition for searching the octree, a sphere is drawn with $p_{(x,y,z)}$ as origin and the radius is set to be the distance between $p_{(x,y,z)}$ and the closest node obtained so far. Since the octree structure resembles a cube, the resulting sphere is bounded by a cube since a search algorithm is much easier to implement when the search boundary resembles a cube. The octree is then traversed through recursively to determine the octants that lie within the search boundary. The nodes that lie in each of these octants are then extracted. The closest neighbor is then determined by comparing the distances between $p_{(x,y,z)}$ and the nodes within the search boundary. The position of the pedestrian $p_{(x,y,z)}$ will now be mapped to the closest node identified with this method. The algorithm 4 (also illustrated in Figure 3.11) shows the steps involved in identifying the nodes within the search boundary.

In the example (Figure 3.11), a quadtree containing certain number of nodes is generated. On searching the position of $p_{(x,y,z)}$, the octant with Morton index (122) is identified. In order to find the nodes that are located closer than the identified node, a circle with $p_{(x,y,z)}$ as center and r (distance between $p_{(x,y,z)}$

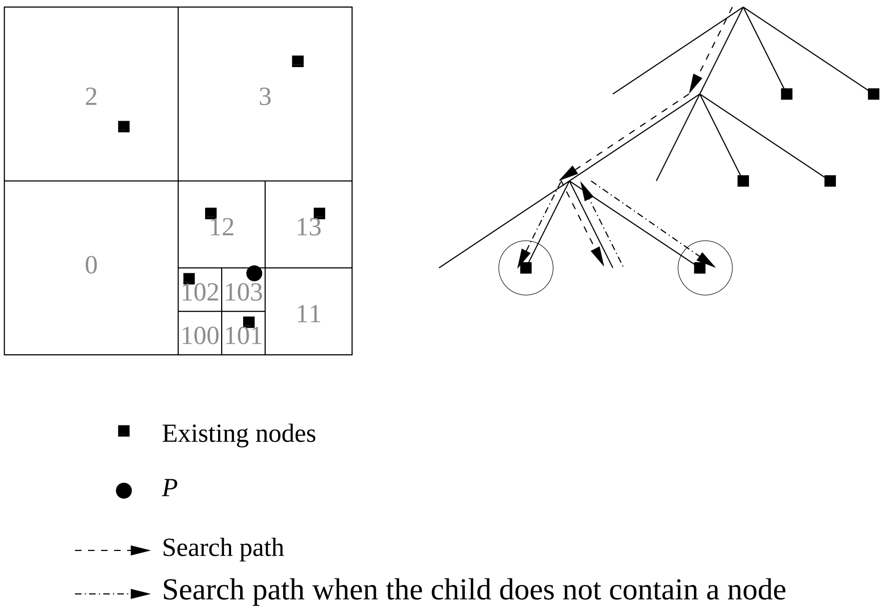


Figure 3.10: Tree search when $p_{(x,y,z)}$ lies in a quadrant, where a node does not exist. Therefore, the neighboring nodes of the same level are searched.

and the identified node) as radius is drawn. An enclosing square is set as the boundary box. Now, on parsing the quadtree to determine the octants and their respective nodes (if a node exist), the octants with Morton indices (033, 120, 122, 1230, 1232, 2, 300) are identified that lie partially or fully inside the square. Octants with Morton indices (031, 121, 301) are omitted since they do not contain any node. By comparing the distances between $p_{(x,y,z)}$ and each of the 7 derived nodes, the closest node is found in the octant with Morton index (2).

In comparison to a linear search (comparing the geometric distance of all existing nodes with $p_{(x,y,z)}$ and determining the closest node – computational complexity of $\mathcal{O}(n)$), the complexity of the node search using hierarchical data structures such as the octrees is $\mathcal{O}(\log(n))$. Since the neighboring nodes are also searched, the search is repeated until the nearest neighbor is found. The complexity therefore is $\mathcal{O}(m * \log(n))$, where m lies in the interval $[1, n]$. In the best case, the complexity turns out to be $\mathcal{O}(\log(n))$ and in the worst case, the complexity is $\mathcal{O}(n * \log(n))$ which is worse than a linear search. In normal cases, especially for large numbers of n as in our scenario, the complexity tends to be logarithmic.

Now that we have identified the position of the pedestrian, the pedestrian now wishes to execute his tasks. Let the next task be T_j, where j represents the type of tasks and there exists many destinations $d_i|\{i = 1, 2, \ldots, n\}$, where the

Algorithm 4 Method to determine the octants within the search boundary.

```
 1: get $p_{(x,y,z)}$
 2: determine node $n(p_{(x,y,z)})$ {closest node identified so far}
 3: set closest_distance = distance($p_{(x,y,z)}$,$n(p_{(x,y,z)})$)
 4: set bounding cube (of sphere) with end co-ordinates $(x_1, y_1, z_1), (x_2, y_2, z_2)$
 5: **Func**: nearest_node(node)
 6: start tree search
 7: **if** (child node $\neq \phi$) **then**
 8:    **if** (node inside cube boundary) **then**
 9:       call nearest_node(child node)
10:    **end if**
11: **end if**
12: **end Func**
13: get nodes of all identified octants
14: **while** there exists nodes to compare **do**
15:    get distance($p_{(x,y,z)}$,current_node)
16:    **if** (distance<closest_distance) **then**
17:       set distance=closest_distance
18:    **end if**
19: **end while**
```

tasks j can be executed. Thus the destination d that is closest to $p_{(x,y,z)}$ is to be identified. The identification of such a destination is again done using neighbor search in an octree. We first identify the set of destinations $d_k|\{k = 1, 2, \ldots, m\}$ such that $d_k \subset d_i$. For this, we define a search radius from the current position $p_{(x,y,z)}$ and obtain all nodes d_k that belong to the type d_i. The destination $d \subset d_k$ is chosen and the pedestrian is directed to d. This process will be explained in 3.3.2.

The performance and complexity of determining d_k is same as the algorithm to determine the closest neighbor. A larger search radius as compared to position identification method however increases the domain of search and therefore requires more time to determine d_k. Certain heuristics are however used to define the search radius such that the search complexity is reduced. One such heuristic is the search of destinations in the same floor. For this, the search radius is defined such that only the nodes along the $x - z$ plane is searched and the search along the y axis is restricted.

So far, the position of the pedestrian in the reference scenario (position on the graph) is identified. Also the list of destinations in the neighborhood is also identified. Now, the pedestrian needs to be navigated to his chosen destination. It must be noted that the closest geometrical destination need not necessarily be

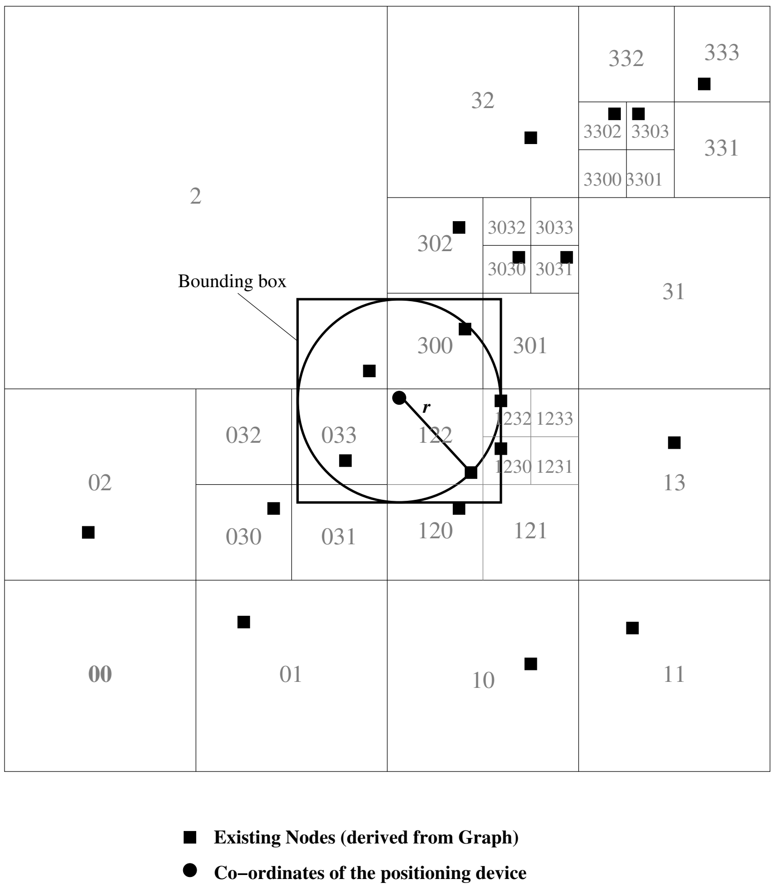


Figure 3.11: Method to determine the closest neighbor of $p_{(x,y,z)}$ in a quadtree example.

the closest destination to reach (a room located one floor above will be a couple of meters away from the current position but the path may be much longer). Therefore the destinations must be sorted according to the path length such that the pedestrian can choose the closest located destination to the current position. The path length can be determined from the path-search algorithm described in the next section.

3.3.2 Path-Search Algorithms and Navigation

A path-search algorithm (or the shortest-path problem) is used to determine the path between two points such that the overall cost (distance, congestion, toll roads) of the path is minimum. Path-search algorithms are widely used in street routing and navigation systems using a GPS device. Path-search algorithms are also used in logistics and transportation to calculate the time and cost of the service offered. With the use of a single-source shortest path (SSSP) method, all possible paths that leave the starting node is identified and the path with the lowest graph weight is the shortest path. The weights of the graph can be changed (set weight of toll-roads = ∞ such that toll-roads are avoided) such that the desired type of path is obtained. Furthermore, services such as a radar trap or real-time traffic messaging service (TMC) are offered such that alternate (cheaper) routes can be calculated using these services [Mei04].

The most commonly used method to determine the shortest path between two points is the Dijkstra's shortest path algorithm [Dij59]. The algorithm calculates the shortest path from the start node to all other nodes. As soon as the end node is reached, the algorithm terminates and the shortest path is determined. The algorithm uses a greedy search heuristic. During the search, every node of the graph is associated with the attribute – if the shortest path to it is determined or not. The algorithm works with the principle that the shortest path is the sum of shortest sub-paths leading to the end node (for implementation details, refer to chapter 2). With the graph of n nodes and m edges, the algorithm has a computational complexity of $\mathcal{O}(m + n \cdot \log n)$ when efficient data structures are used to store the intermediate nodes. Other algorithms that can be used to calculate the shortest path include Bellman-Ford algorithm (solves single source problem even for negative weighted edges), A* search algorithm (solves for single source shortest paths using heuristics to try to speed up the search) and Floyd-Warshall algorithm (solves all pairs shortest paths).

From the previous section, we have obtained a list of destination d_k from which a pedestrian may choose a destination d to execute the task T_j. The geometric distance between $p_{(x,y,z)}$ and each nodes of d_k may not necessarily identify the closest node to $p_{(x,y,z)}$ since the path distance and the geometric distance are different. Therefore, the single-source shortest path is used to determine the shortest path to each of d_k and the list d_k is sorted according to the increasing distance from the source. The list of possible destinations and the distance to it are presented to the pedestrian such that the pedestrian can choose his destination d (often the closest node). The path-search can also be improved by considering the priorities of the pedestrian. For example, wheel chair users cannot use the stairs and must therefore routed through the elevator. Since the graph extracted also contains the type of the paths, the edges that represent stairs can be searched and set to ∞. The path-search algorithm therefore avoids the stairs since they will never be a part of the shortest path. Numerical examples of destinations and path-search methods will be presented in chapter 6.

Once a path is chosen, the pedestrian can be interactively navigated to the destination. They can be done either in advance through some visualization environment or in real-time with the use of interactive navigation devices. A VRML (Virtual Reality Modeling Language) model or the visualizer tool developed for visualizing the pedestrian simulation (see chapter 4) can be used to view a virtual flight from the current position to the destination. Figure 3.12 shows a snapshot of VRML application for the shortest path-search performed in the reference model used here [GSG04].

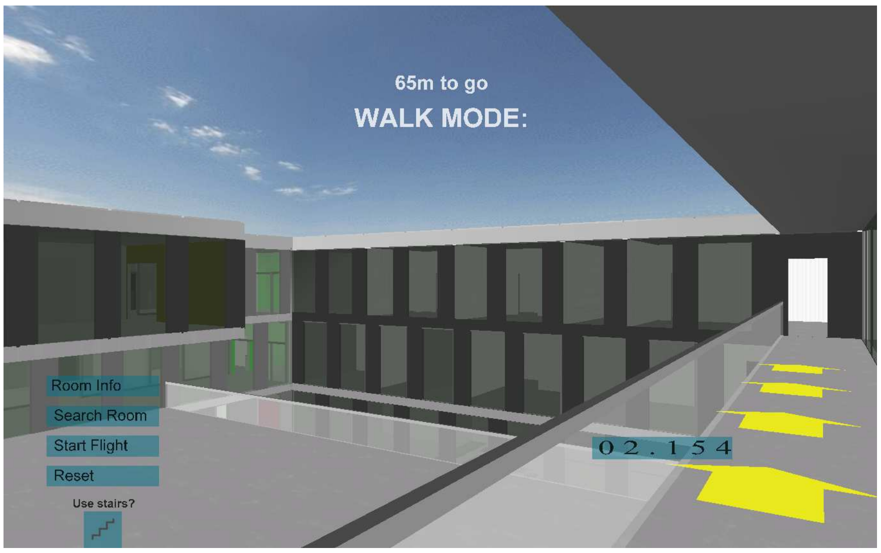


Figure 3.12: A snapshot of a VRML model for calculating the shortest path and guiding to the destination in the reference model (Computer Science building at the Universität Stuttgart)

3.4 Summary

In this chapter, the concepts of geometric modeling and representation as well as the extraction of the necessary geometry and architectural details are explained. The reference model and the parameters necessary for the pedestrian simulation in the reference scenario is also presented. Also, a method to support pedestrian navigation using the graph data is developed. In this method, the graph is partitioned into an octree, which again is used for position identification and location awareness.

For supporting a pedestrian navigation system, we assume that the pedestrians arrive with some kind of mobile navigation devices in hand, which can already identify the co-ordinates of its location within the building. We use the graph to transform the available co-ordinates to the position within the graph.

The major drawback with the system is that the octree model depends on the available co-ordinates of the position identification device. It assumes that the co-ordinates transmitted to the octree server is accurate. However, such devices have a certain degree of error and therefore, a certain level of fault tolerance must be built within the octree for an exact position identification.

Typical indoor position identification devices include the use of WLAN signals, sensors, RFIDs, BlueTooth, etc. In the case of sensors or RFIDs, the sensor devices are already coded with their corresponding location within the building. Therefore, the information from the sensors directly transmits the current location (room number or location in the graph) within the building to the pedestrian. Position identification using WLAN works on the principle of triangulation. The co-ordinates of the WLAN access points are known. Therefore, a client measures the signal strength to the available access points in the neighborhood and calculates its current co-ordinates. Generally, WLAN position identification devices use a cell-based approach to determine the exact location (e.g., room number) in the building. In a cell-based approach, several 3-dimensional cells are calibrated within the building and each cell receives an `ID`. The signal strength range within the given cell is known and whenever a position identification device is brought within this cell, the device automatically identifies the corresponding `ID`.

All the above-mentioned methods to identify the position suffer from a major drawback. That is, to calibrate a cell, a physical presence in the scenario is required and the WLAN access points and the receiver must be calibrated manually. This drawback can be overcome by the use of the octree model since it is only necessary to have a CAD model in order to build a location-aware system. Also, the octree model offers a more efficient search mechanism, especially for larger models.

Chapter 4

Discrete Event Pedestrian Simulation

Anyone who considers arithmetic methods for producing random digits is, of course, in a state of sin.
– **John von Neumann**

In the previous chapter, we identified the set of paths and the geometric parameters required to model and simulate pedestrian traffic, from the reference model. A method to identify the locations for a new pedestrian, as well as navigating the pedestrian to the destination has been described. Now, we need to analyze the pedestrian behavior in the reference model. Therefore, we build a framework, which allows us to model and simulate the pedestrian behavior using queuing systems and integrate the simulation within the geometry of the reference model.

Pedestrian movement is one of the most commonly seen activities in day to day life. Airports, shopping malls, football stadiums, exhibitions or any other similar facilities today attract more and more pedestrians. Since there is a continuous growth in the amount of pedestrian activity, modeling pedestrian behavior turns out to be complex and important. Recently, pedestrian simulation has received more attention, especially in safety design and evacuation process. Concerns in the environmental degradation and awareness of physical fitness[1] also causes an increase in the pedestrian activities, which in turn brings a necessity to improve

[1]This also lead to the concept of car free days. The world carfree day (September 22 of each year), launched by the world carfree network is once such initiative to encourage the use of public transport and pedestrian facilities by avoiding the use of a car.

pedestrian facilities. A sensible improvement of pedestrian facilities can be made possible only through pedestrian studies. Therefore, pedestrian modeling and simulation plays a significant role in increasing the standards of the pedestrian infrastructure.

Pedestrians are undoubtedly one of the most complex beings to model. Unlike vehicular traffic simulation (cars, trucks, etc.), pedestrian behavior is not disciplined. The pedestrian behavior pattern is very inconsistent among each other. Pedestrians involve themselves in a wide range of activities. Starting from daily commuters who rush to their destination to tourists who walk leisurely from shop to shop, pedestrians exhibit a wide range of behavior. [Tek02] shows that pedestrians have the maximum fluctuations in speed, acceleration, change of direction, spontaneous decisions, etc. A typical commuter during business hours would intend to get to the train as fast as possible. He would take the shortest path to the train and would rarely change directions. On the other hand, tourists tend to stop often for a window-shopping and to rest. Furthermore, architectural hindrances such as fountains or benches also tend to slow down the journey.

This chapter deals with modeling pedestrian behavior in buildings, specifically in a 3-dimensional scenario. The pedestrian movement is virtually 2-dimensional and the pedestrian movement in a (3-dimensional) building can be mapped to a 2-dimensional graph. However, the geometry of the building plays a significant role in the pedestrian simulation. For example, in chapter 3, it was shown that the geometry data is used to measure the height difference between 2 planes along the depth buffer to determine if a pedestrian movement can exist there. The pedestrian behavior is therefore modeled in a 3-dimensional scenario. The geometric modeling explained in chapter 3 provides the necessary geometric parameters for the pedestrian model. This chapter begins by explaining the various possible methods to model pedestrian behavior. The concepts of discrete event simulation and stochastic processes required to model and simulate the pedestrian behavior are then presented. An analysis of the pedestrian behavior and characteristics are then made. The input modeling and the stochastic functions used to translate the statistics of pedestrian behavior into input data for the pedestrian simulation are then presented together with the analysis of the pedestrian characteristics. The implementation and analysis of the simulation model within the available geometric scenario is then discussed. Finally, the components developed within the pedestrian simulation framework are summarized.

4.1 Pedestrian Simulation Methods

[May89] shows that pedestrian traffic simulation can be broadly classified as macroscopic and microscopic simulations, the former being commonly used whereas the latter is more complicated. Pedestrian simulation is often used to analyze issues such as evacuation scenario [Cas05], study of pedestrian dynamics [HBD02], urban planning, safety and security issues (crash simulation, collision with the car front, etc.), pedestrian surveillance analysis, etc. There exist several methods and strategies to model and simulate pedestrian activities. Following are some of the methods generally used to model and simulate pedestrian behavior.

4.1.1 Numerical Simulation

In a pedestrian simulation using numerical methods, flow- or particle-based simulation methods are used. Pedestrian behavior modeling using a numerical approach is proposed in [Hel90, Hen74]. A resemblance of the self-organization effects or the lane-forming effects in the pedestrian crowd as compared to gases or fluids was proposed by Henderson in [Hen71, Hen74]. Generally, a macroscopic simulation is performed to evaluate the density or throughput. Due to the non-constant acceleration, difference equations, instead of numerical integrations are used. A realistic fluid-dynamic theory for pedestrians must contain corrections due to their interactions (avoid collision, rapid acceleration and deceleration). Although such theory can be formulated [Hel92b, Hel92a, HB01c], such methods are unsuitable for microscopic pedestrian simulation in practical cases [HFMV02]. [Thi01] proposes the possibility of using Navier Stokes equations to visualize pedestrian flow as incompressible fluids, thereby simulate the flow movement of pedestrians.

In a social force model first proposed by Helbing [HM95], the pedestrian reacts to the social forces that motivates him. Pedestrians typically walk to their destinations comfortably by taking the shortest path and avoiding any collisions (walls, benches) on the way. The pedestrians also maintain a distance with other pedestrians to avoid collisions. Pedestrians also sometimes interact with other pedestrians along the way (friends, salesman). The effects of the motion of the pedestrians α shall be determined from these parameters. The velocity of the pedestrian is therefore calculated by the summation of the force model of the above mentioned characteristics $\frac{dv_\alpha}{dt} = F_\alpha(t) +$ fluctuations (random variations of the behavior).

4.1.2 Agent-Based Simulation

An agent-based approach to simulated pedestrian behavior is commonly used to study social behavior of the pedestrians. Agent-based simulation is a special form of microsimulation, where each pedestrian is treated as an entity or an agent with distinct state or behavior. In a multi-agent system, agents are active entities moving in an environment, where the environment for an agent consists of other agents and explicit environmental entities such as resources. The interactions among the agents are the central point of focus. Therefore, in an agent-based simulation, the individual behavior of the pedestrian (or the social aspects) are modeled and simulated with an intention to study the overall behavior of the pedestrians. Agent-based approaches to simulated pedestrian behavior are seen in [HW04, KHW01].

Agent-Based approach is also seen in simulation of pedestrian evacuation [TBdS06]. The advantage of an agent-based approach is that properties such as prior knowledge regarding the building layout (commuters vs. tourists), different mobility speeds (children, adults, elderly people), etc., can be defined to each entity. Therefore, the social behavior in a panic situation can be analyzed precisely.

Agent-Based simulation is also used in optimization of building layouts (for example, the usage of swarm-moves simulation to increase sales in supermarkets [UM06]), where the typical characteristics of the pedestrians are modeled within the entity and the impact of the model in the given environment can be studied. A computer model also allows the model to be re-run several times by altering the parameters to evaluate different situations [HFV00].

4.1.3 Cellular Automata Model

The cellular automata model was typically used in vehicular traffic simulation as proposed in [NS92]. Pedestrian simulation based on cellular automata was proposed in [GM85]. In a cellular automata method, the simulation of pedestrians is performed as entities in cells. The walkway is modeled as grid cells and a pedestrian, if present in the cell, occupies the cell. The occupancy depends on localized neighborhood rules, which are updated constantly. Pedestrians can change lanes or hop cells. In each time step, the cell can take two states (occupied, free). When a pedestrian changes the lane, the adjacent cells in the column are adjusted such that the gap between other pedestrians moving in the column is maintained. This is done by adjusting the speed in parallel to the cell change activity. However, the pedestrian behavior does not resemble a traffic movement and seems unsuitable to use cellular automata simulation for pedestrians. Nevertheless [Ba98] suggests a good idea of validating the cellular automata microscopic model.

A benefit cost cellular model, similar to the cellular automata method, is proposed in [Gip87]. It simulates the pedestrian as a particle in a cell. The walkway is divided into square grids and each cell can be occupied by utmost 1 pedestrian. A scoring system is used to each cell based on the proximity of the pedestrians. The score represents the repulsive effect of the nearby pedestrians (9 cells including the pedestrian himself) and has to be balanced as the pedestrian moves toward his destination. Benefit cost cellular model uses arbitrary scoring and the system makes the model difficult to calibrate in a real world scenario.

4.1.4 Queue-Based Simulation

A queue-based simulation is often developed for simulating evacuation scenarios [Lov94, OM93]. A queue model is built using a discrete event Monte Carlo simulation. Several nodes (rooms, intersections, etc.) and links between them exist in the evacuation scenario. A queuing system is built for each link between the nodes. Each pedestrian enters the queue through the node, wait in the queue and reach the end node of the queue. Each pedestrian shares the common goal of evacuating the building, i.e. reach the nearest exit door as fast as possible. The queuing systems are interconnected to form a queuing network. Each pedestrian, once he arrives at a node, selects his queue from n possible nodes using a weighted-random choice. The weight function is determined by the current population density in the queue. If the desired queue cannot be used, the pedestrian decides on a different queue. The initial time ($t = 0$) is the time the pedestrian arrives at the node. The pedestrian first experiences a delay in choosing the right queue and begins to wait at the *source* of the queue model at Δt. The total waiting time in the queue W is the time it takes for the pedestrian to reach the *sink* of the queue model. After that, he will stop the movement process. These steps are repeated until the pedestrian reaches his destination (the exit door).

A Queue is a natural process that occurs at destinations if the number of arrivals exceeds the throughput of the system. Such destinations can be modeled and simulated as a queue model. Since the movement of pedestrian resembles a queue (pedestrians arrive at a path, wait if the path is congested and walk once the path is free), a queue model can also be used to represent the behavior of pedestrians walking along the path. Therefore, we use a queue model to describe both movements of pedestrians as well as the behavior of the pedestrians at the queue. In the following section, we discuss in detail the implementation of a queue model and its usage for pedestrian simulation.

4.2 Discrete Event Simulation of Pedestrians

So far, various methods to model and simulate pedestrian behavior were presented. We have decided that a queue-based simulation shall be used to simulate the pedestrian traffic in the reference scenario. We therefore simulate the pedestrian behavior as a queuing system model, both along the paths and the destinations. It is possible to determine the overall system parameters such as the average waiting time in the queue, average queue length, throughput of the system, etc, from the queuing system. Since queuing systems can best be modeled and simulated using the discrete event simulation methodology, we use discrete event system simulation to simulate the pedestrian behavior.

In a discrete event simulation, the state variable changes only at discrete points in time. The system state changes on the occurrence of a new event. So it is sufficient that the system advances its state directly to the next state when the event would occur. One of the most common usages of a discrete-event simulation for pedestrian system is a pedestrian queue. For example, consider the situation of a pedestrian visiting a post-office counter to execute a certain task. The post-office queue contains 3 events, namely arrival event, service event and departure event. On the occurrence of any one of these events, the system state is changed, the system variables are updated and the next set of tasks are triggered. When many such queues exist at various locations, the location of the queue that correspond the post-office as well as the path to it are identified using the geometry model presented in the previous chapter.

The discrete event simulation for simulating pedestrian traffic has been suggested by many other researchers. For example, [JH94] proposes the use of discrete event simulation for traffic analysis in quick service restaurants. This approach combines the movement of pedestrians and vehicles (drive-in or take away counters) within the surrounding layout of the fast-food center. Also the use of discrete event simulation to model pedestrian evacuation can be seen in [BS06].

4.2.1 Queue Model

A queue model is one of the fundamental blocks used to model and simulate pedestrian traffic here. A queue model can be used to model many systems such as traffic, network flow, business processes, logistics, etc. A queuing system, as in a discrete event simulation, is driven by events. The events of a queue model are termed as processes. In our example, the processes of a queue model include, arrival and service processes. In modeling processes, the arrival time or the service is generated using existing values or random functions. In an arrival process, the random function generates the inter-arrival time between 2 consecutive arrivals. In a service process, the random function generates the time it takes to serve an entity. The arrival and service processes have normally no connection with each other. There are three broad classes of processes, namely deterministic (D) process (arrivals occur at a constant rate), Markovian (M) process (arrivals occur at a negative exponentially distributed time intervals with an arrival rate of λ) and general (G) process (arrivals occur at an arbitrarily distributed time intervals). Each event changes the system state and triggers the next event connected with the current event. Another component that plays an important role in a queue model is the entity e (here a pedestrian). An entity is a component or an object in the system, which requires explicit representation of the model (e.g., server, customer, machine). Each entity is associated with some attributes or properties such as priority in the queue, ability to choose a certain queue, behavior, etc. Several entities participate in the queue model. Therefore, there exists a logical structuring of the entities in the queue model (waiting line in the queue, FIFO, LIFO, etc.).

4.2.1.1 Components of a Queue Model

At first, to implement a queuing system, certain data structures are necessary to organize the entities and events i.e., to maintain the list of events and to logically organize the entities. A tertiary tree structure can be used to store both list of events and entities. In this tertiary tree, each parent node contains a left and a right child node. The middle node is connected to a doubly linked list (DLL), also represented using a tertiary tree (tree structure explained later in 4.2.1.3). A queue model consists of the following components (see Figure 4.1)

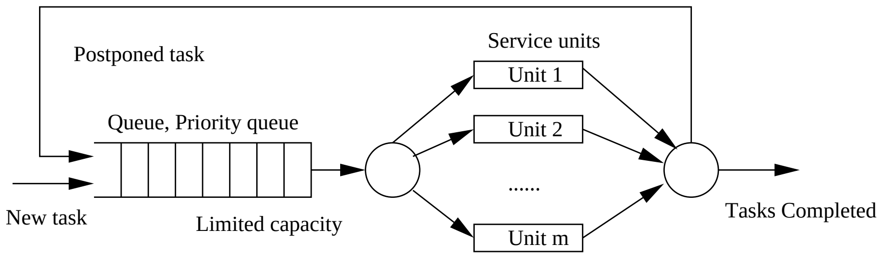


Figure 4.1: A queuing system with m service units and a (priority-based) queue.

- A queue Q, which may be a normal FIFO queue or priority queue or may even support preemptive service. A queue may have a limited capacity $Q_{\max}$.

- One or more service units, where one entity per service unit is processed at a time. The state of the service unit is either busy or idle $SU \in \{busy, idle\}$.

In a queue, entities wait before they are served. Whenever an entity arrives in the queuing system, the entity is appended (or prepended[2] depending on the priority) in the queue, even if the queue is empty. If a service unit $SU = idle$, the entity leaves the queue and joins the service unit. Each queue is modeled using the tertiary tree structure. The left and right nodes of the root node are set to null and the doubly linked list is used to sequence the entities. Inserting an entity to or removing from the DLL is straightforward. The behavior of an entity in the queue may however be different. Following are some of the queue behaviors modeled.
Queue Swapping: If more than one queue exist, an entity may decide to join another queue. Swapping of an event E from queue Q_1 to queue Q_2 can be modeled as

$$if(\text{condition}) \quad \{\text{remove}(E, Q_1);\ \text{add}(E, Q_2);\}$$

Balking: If the queue Q is too long, it avoids further addition of events E to the queue.

$$if(\text{size}(Q) < \max) \quad \{\text{add}(E, Q);\}$$

Reneging: Abrupt termination of an entity e from the queue Q. This situation normally happens if the alloted time exceeds or the entity waits too long. In that case, the event E is canceled.

$$if(T_{wait} > T_{\max}) \quad \{\text{search}(Q, e);\ \text{remove}(Q, e);\ \text{cancel}(e, E);\}$$

Service Preemption: Preemption occurs if an entity with a high priority interrupts a service. The new entity is served and the service event E for the entity with a lower priority is canceled and postponed to a later time T.

$$if(e_{priority}) \quad \{\text{cancel}(e, E);\ \text{postpone}(e, T);\ \text{serve}(e_{priority}, E);\}$$

The n number of service units are represented as resources. As long as a resource is available, the next waiting entity is removed from the queue and served. During service, the resource is set to be occupied ($SU_i = busy$). When the entity leaves the service unit, the resource is again freed ($SU_i = idle$). At any point during the simulation, it is possible to determine the number of resources that are available or occupied. The occupied resources denote the population p of the functional unit. The throughput μ is the rate at which the tasks are finished and the maximum throughput $\mu_{\max}$ is the maximum possible throughput of the functional unit.

[2]Prepend operation is made in case of a LIFO queue

Therefore, the system utilization $\rho = \frac{\mu}{\mu_{\max}}$.
Time Dependency: It is normal that the arrival rate varies as time progresses. For example, the arrival rate is high (the inter-arrival time is less) during peak hours and the arrival rate gradually decreases as time progresses. The time dependencies can be modeled by defining discrete time frames and specifying the arrival pattern for each time frame.

$$if(T_i < CLOCK \leq T_{i+1}) \quad \{\text{arrival}(\lambda_i);\}$$

4.2.1.2 Event List

In an event-driven simulation, an event E that occurs at time t changes the system state at that time. Therefore, the events must be organized in a chronological order in an event list $EL(t)$. The time is controlled and monitored by a clock function in a discrete event simulation. This time has no connection with the real world time. As seen before, there are different types of event (arrival, service, departure). Since one event is correlated with the other event, the events must be managed in the same list. Therefore, an event E is represented as $E = (E_a, E_t)$, where E_a is the type of the event and E_t is the time at which the event occurs. Every time there is a new event, the event is inserted into the event list and every time the event is executed, the event is removed from the list. An event is typically controlled by a task T. The tasks that enter and leave the queue are normally numbered in increasing order $(T_1, T_2, T_3, \dots)$. Each task has some attributes attached with it such as arrival time $T_{i,at}$, departure time $T_{i,dt}$, service time $T_{i,st}$ and waiting time $T_{i,wt}$. When a task triggers an event, the task is scheduled in the event list.

4.2.1.3 Event Scheduling

Scheduling of the events is one of the most complicated parts in a discrete event simulation. Event scheduling involves scheduling the events in the chronological order and updating the event list as the events are executed along the time progress. The events must be sorted in the increasing order of their occurrences. In some cases, the occurrence of certain events is dependent on its preceding event. For example, a pedestrian shall be served only after he arrives in the queue. If the resources are blocked and if there are new arrivals, the service event cannot be scheduled immediately. Such events are called conditional events. To simplify the scheduling of the conditional events, the “ABC” method or the “three phase” method is used to schedule the events in the event list [Tys99]. Whenever the events are executed, this approach always scans the conditional list and schedule them if there are no barriers. The control structure of the ABC approach consists of the following three phases:

- **A**: the simulation time **A**dvance phase,
- **B**: execution of event routines associated with all the **B**ound (or certain to happen) events that have the same earliest activation time,
- **C**: scanning of the **C**onditions that trigger successive conditional events.

Figure 4.2 describes the scheduling of events using a 3-phase approach. In the tertiary tree used for scheduling events, the left and right nodes are used to store the next event and the center node, connected to the doubly linked list, is used to store the conditional events. In short, the event list is represented as a sorted binary tree (BST) and any events that cannot be executed or scheduled is then stored in the doubly linked list. The conditional event list is a ring of entity records, all of which have the same activation time. As time progresses, the events from the binary sorted tree are first executed. Then, the node is checked for any conditional events that need to be executed. We use the tertiary tree structure to store both the conditional list and the event list, i.e. we use the tertiary tree to also represent a doubly linked list (see Figure 4.3). The right pointer is used for adding new entities that have a later activation time. Since the events with equal activation time are doubly linked, additions to both the beginning and end can be made efficiently (also seen in queues when using the tertiary tree as a doubly linked list). If the conditional events cannot be executed together with the parent event, the conditional event is moved to an event with the next activation time.

4.2.1.4 Queuing Notation

D.G.Kendall proposed a notational system for parallel server systems in 1953. Kendall's notation for classification of a queuing system has the format

$$A|S|s|c|p|D,$$

where

A represents the arrival process or the inter-arrival time distribution (e.g., Markovian or general process)

S represents the service process or the service time distribution

s represents the number of service units ($\{SU \geq 1 | SU \in \mathbb{Z}\}$)

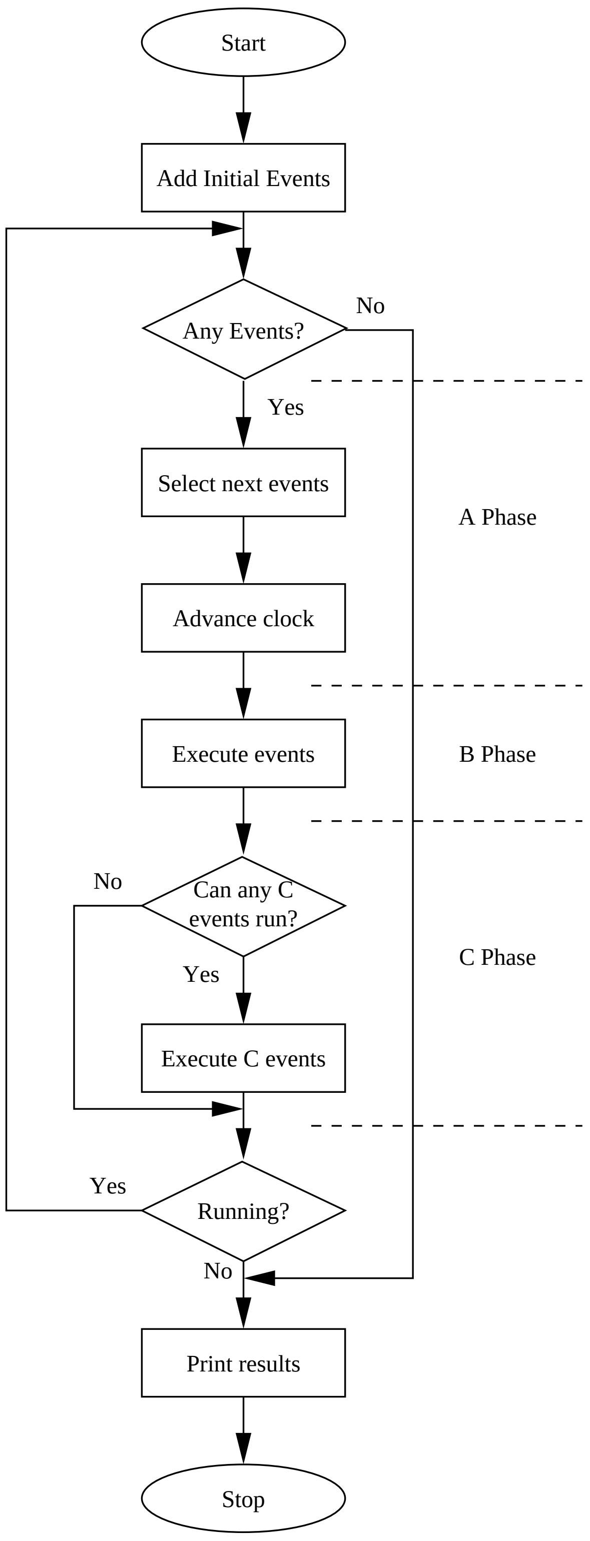


Figure 4.2: The ABC approach for event scheduling.

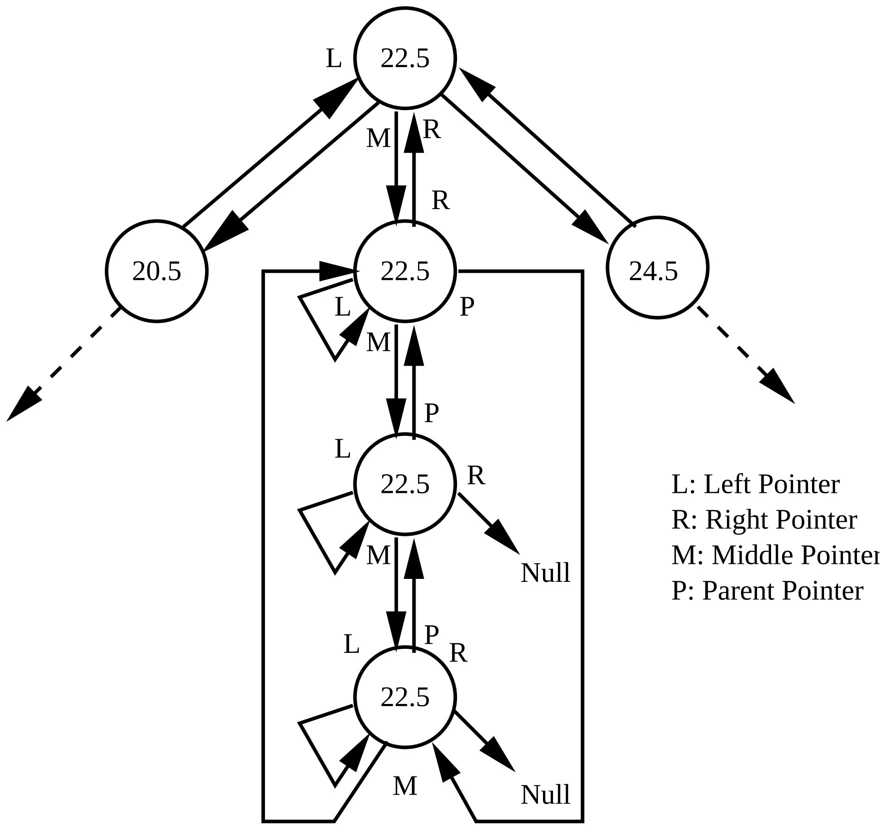


Figure 4.3: Tertiary tree structure with middle pointer pointing to a doubly linked list (also represented using a tertiary tree).

c (optional) represents the system capacity (by default ∞)

p (optional) represents the calling population or the maximum number of tasks that can arrive in the queue (by default ∞)

D (optional) represents the queue discipline (default FIFO)

The simplest queuing system that can be analyzed is the $M|M|1$ queue, where the arrival and service processes are Markovian and the system has a single service unit. The missing optional parameters would take the default value. In case the service process is an arbitrary process, the queuing system is denoted as a $M|G|1$ queue.

4.2.1.5 Stochastic Process and Random Variables

During the simulation of pedestrian traffic, the outcome of a process is generally not known in advance. This means that the process is a non-deterministic or a *stochastic* process. In a stochastic process, the set of outcomes of a random occurrence in a process is parameterized with time [Nel95]. The use of *random variables* describes the *state* or the condition of the stochastic process in a specific time. The state space S is a set of all possible states of a stochastic process. The stochastic models are typically represented using random variables, for example, pedestrian tasks waiting in the queue, the time the server is operational, etc. Random variables are functions that map a real value to every random output of the state space. Random variables, whose possible values are finite (or countably infinite) is called a discrete random variable (e.g., number of pedestrians arriving per hour is $X : S \rightarrow \{1, 2, 3, 4, 5, \dots\}$, where X is a random variable). Often, random variables, for a pedestrian arrival process (arrival time during the day), produce a continuous stream of random numbers in which case, it is called a continuous random variable ($X : S \rightarrow \mathbb{R}$).

There are two functions that characterize a random variable X, namely the probability density function (PDF) $f(x)$ and the cumulative distribution function (CDF) $F(x)$. $F(x)$ is non-decreasing such that $F(-\infty) = 0$ and $F(\infty) = 1$. If X is a continuous variable, then $f(x)$ is non-negative and $\int_{-\infty}^{\infty} f(x)\mathrm{d}x = 1$. If X is a discrete variable, then $f(x)$ is non-negative and $\sum_{x=-\infty}^{\infty} f(x) = 1$. If X is discrete (or continuous), the both its PDF $f(x)$ and CDF $F(X)$ are also discrete (or continuous).

Another important concept in a probability theory is the *expectation* of a random variable. The expectation of a continuous random variable X is defined as $E[X] = \int_{-\infty}^{\infty} x \cdot f(x)\mathrm{d}x$ provided the integral converges absolutely. The expectation of a discrete random variable X is defined as $E[X] = \sum_{\text{all } i} x_i \cdot p(x_i)$. The expected value $E(X)$ of random variable X is also referred to as the mean μ or the first *moment* of X.the quantity of $E(X^n), n \geq 1$ is called the n^{th} moment of X.

Other metrics that are often used in performance analysis are the *standard deviation* σ and *variation coefficient* c (or relative dispersion). They are used to quantify the variability of a random variable X. The variance of a population σ^2 is defined as the expected value of the square of the difference between x and the population mean μ, i.e.

$$\sigma^2 = E[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 \cdot f(x)\mathrm{d}x.$$

The standard deviation is the square root of variance, i.e $\sigma = \sqrt{E[X^2] - E[X]^2}$. The variation coefficient is defined as $c(X) = \frac{\sigma}{E[X]}$ [LK00].

Frequency distributions are useful means of indicating the type of theoretical distribution that best suit the statistical properties of the population under consideration. A careful study of the nature of physical situation from which the data are obtained will indicate the type of distribution followed by the data. Frequency distributions are classified into discrete or continuous distribution.

Poisson Distribution

The Poisson distribution, introduced by S.D.Poisson in 1837 [Poi37], describes many random processes. The selection of samples is often performed and related to a given time interval. *Poisson Experiments* yield numerical values of a random variable X during a specified interval or region. The Poisson distribution is described as

$$f(x) = p(X = i) = \frac{e^{-\lambda} \cdot \lambda^i}{i!}, \lambda > 0, i \in \mathbb{N}.$$

One of the important properties of the Poisson distribution is that the mean and variance are both equal to λ or $E(X) = \sigma^2 = \lambda$. The Poisson distribution is used to describe situations that are concerned with counting the number of times that a certain type of event occurs within a specified frame (e.g., number of pedestrian arrivals in one hour). Such an occurrence of events is termed as a *Poisson process.*

Exponential Distribution

One of the most studied continuous distributions is the *exponential* distribution. The density function of an exponential distribution is given as

$$f(x) = \lambda e^{-\lambda x}, x \geq 0$$

and the distribution function is given as $F(x) = 1 - e^{-\lambda x}, x \geq 0$. An important property of the exponential distribution is the *memoryless* property. This property plays an important role in system modeling since the value of an exponential random variable in a given moment of time t does not depend on its past values. Therefore, it allows us to analyze the system without keeping track of all past events. Thus

$$P\{X > s + t | x > t\} = P\{X > s\}, \ \forall s, t \geq 0.$$

The exponential distribution has been used to model inter-arrival times when arrivals are completely random and to model service times, which are highly variable. In these instances, the rate is λ: arrivals per hour or services per minute, which is a Poisson process. The times between arrivals $T_1, T_2, \ldots$ are independent exponential random variables with mean

$$1/\lambda : \ P\{T_1 > t\} = P\{N(t) = 0\} = e^{-\lambda t}, P\{T_2 > t | t_1 = s\} = e^{-\lambda t},$$

where $N(t)$ represents the total number of events that have occurred in the time frame.

Stochastic Processes

In a *stationary process*, the distribution is independent of time. Therefore,

$$E[X(t)] = E[X(t + s)] = E[X].$$

For example, in a lecture hall, where the pedestrians arrive just before the lecture begins, and once the lecture has started, there are no further arrival or departure processes. The distribution at the lecture hall stays constant until the process is complete. Generally, the time shortly before and after a stationary process experiences a rapid fluctuation at the source of the queue and the paths that lead to it. The inter-arrival times decreases rapidly just before the start of the lecture. Similar fluctuation is seen once the lecture is complete (see Figure 4.4). So during the start and end of the lecture, one can expect congestions near the lecture hall.

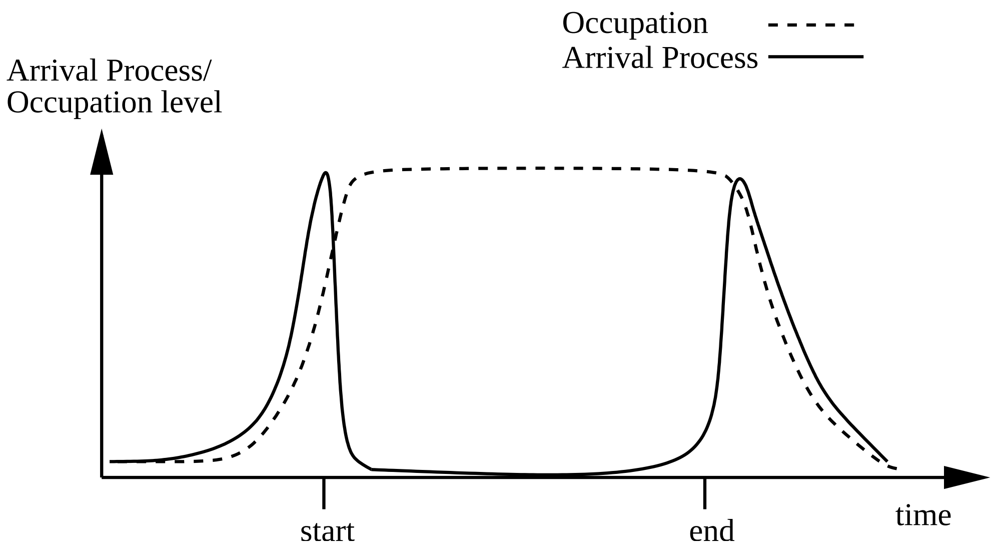


Figure 4.4: The arrival pattern before and after a stationary process.

In an *independent process*, there exists no dependence of the events with earlier results (e.g., pedestrian arrivals). Therefore

$$p(X(t_n) \leq x_n | X(t_{n-1} = x_{n-1}) = p(X(t_n) \leq x_n).$$

Markov Process

Markov Process is a random process whose future probabilities are determined by its most recent values and it follows the Markov property. Markov chains – named after Andrey Markov (1856-1922), a Russian mathematician who made contributions on the theory or stochastic processes – are classified as discrete or continuous [Mar]. A Markov chain describes the states of a system in successive times. A stochastic process $x(t)$ is called a Markov process if for every n and $t_1 < t_2 < \cdots < t_n$, we have

$$P(x(t_n) \leq x_n | x(t_{n-1}), \ldots, x(t_1)) = P(x(t_n) \leq x_n | x(t_{n-1})).$$

This is equivalent to

$$P(x(t_n) \leq x_n | x(t), \ \forall t \leq t_{n-1}) = P(x(t_n) \leq x_n | x(t_{n-1}))$$

[Pap84]. If the value of the next event depends on current value but not the previous values, the process is called a *Semi-Markov Process* (e.g., state transition diagrams). Therefore, the previous values cannot be used for making a future prediction. If in a Semi-Markov process, the predecessor dependence is constant, the process is called a *homogeneous Markov process.* Markov chains with repetitive structure occur when queue models are analyzed via their embedded Markov chains. The simplest queuing system is the $M|M|1$ queue. According to the Kendal's notation in an $M|M|1$ queue, both arrival and service processes are Markovian. Since there is no restriction on the system population and queue capacity, the state space of the embedded continuous time Markov chains is infinite. The population process however is a homogeneous Markov process. The state transition diagram is illustrated in Figure 4.5.

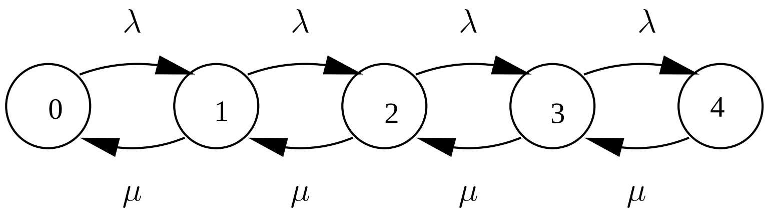


Figure 4.5: State transition diagram of a Homogeneous Markovian Process.

Generally, an $M|M|1$ queue is known as a *birth-death process* – semi-Markov chains, which increases or decreases only by 1. A homogeneous Markov process in discrete time is

$$p_{i,j} = p(X(t_n) = j)|X(t_{n-1} = i)$$

(see Figure 4.6) and in continuous time is

$$p_{i,j}(\Delta t) = p(X(t + \Delta t) = j|X(t) = i), \ \lambda_{i,j} = \lim_{\Delta t \to 0} \frac{p_{i,j}(\Delta t)}{\Delta t}$$

(see Figure 4.7).

In the transition diagram (Figure 4.5), each state of the system is distinguished by the number of jobs (population) in the system. The state space of the process is $S = \{0, 1, 2, \dots\}$. The mean arrival rate is λ and the mean service time is μ.

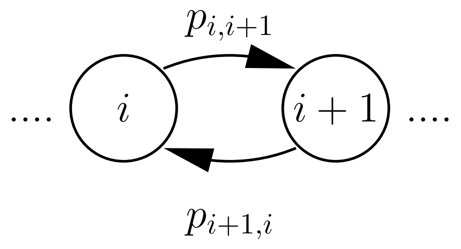


Figure 4.6: Homogeneous Markovian Process with transition probabilities $p_{i,j}$.

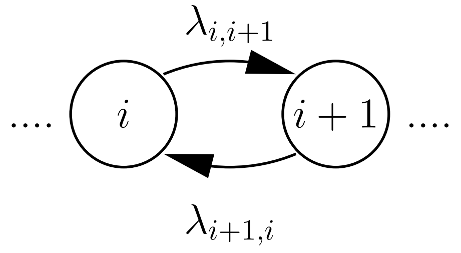


Figure 4.7: Homogeneous Markovian Process with transition rates $\lambda_{i,j}$.

The status of the population is represented as

$$\begin{aligned} p_0 \cdot \lambda &= p_1 \cdot \mu \\ p_1 \cdot \lambda &= p_2 \cdot \mu \\ p_2 \cdot \lambda &= p_3 \cdot \mu \\ &\cdots \end{aligned}$$

Therefore

$$p_i \left(\frac{\lambda}{\mu}\right)^i \cdot p_0 \overset{\sum p_i = 1}{\Longrightarrow} p_0 = 1 - \frac{\lambda}{\mu} = 1 - \rho, \text{ where } (\lambda < \mu).$$

Therefore, in a $M|M|1$ queue, the mean population is $\frac{\rho}{1-\rho}$, mean throughput is λ, mean service time is $1/\mu$, mean sojourn time due to Little[3] is $1/\mu(1-\rho)$ and the long term average waiting time is $\frac{\lambda}{\lambda(\mu-\lambda).}$.

4.2.1.6 Queuing Network

A single queuing system is clearly an elementary system to be used in general systems. The complex pedestrian model involves several queuing systems connected together as a graph. The edges represent the paths of tasks traveling

[3]The Little's theorem or the conservation equation [Lit61] states that the average population is same as the average sojourn times the average throughput

through the system and the nodes represent the queuing systems. A typical task involves traversing through several queuing systems. The entire network contains many processes that occur in parallel, each with a different set of tasks. The process typically contains a stream of tasks with precedence relation, i.e. when $T_i \prec T_j$ then T_i must be completed before T_j. The sequence can be stochastic or deterministic or both. Queuing networks can be classified into two different classes namely closed network, where the population is constant i.e. entities from within the queuing network move from one queue to another internally, and open network, where the population is variable i.e., new entities may enter the queuing network and old entities may depart the queuing network. Jackson networks, proposed by J.P.Jackson in 1957, is an example of an open network. Consider a queuing network with $J \geq 1$ queues, where pedestrians arrive from outside as independent Poisson processes, with the rate λ_l into queue l (where $1 \leq l \leq J$). The pedestrians require independent service times at each visit to the queues (similar to the reference scenario used here). The service times in queue l are exponentially distributed with rate μ_l. Also the service times are independent of the arrival processes. Upon leaving queue l, each customer is sent to queue m with probability p_{l_m} for $1 \leq m \leq J$ and leaves the network otherwise. The routing decision is independent of the past evolution of the network (or otherwise termed as Markov routing) since the probability of a routing decision depends only on the current position of the customer. Gordon-Newell networks, proposed by W.J.Gordon and G.F.Newell in 1957, are an analog to Jackson networks except that a Gordon-Newell network is a closed network.

Other methods to represent waiting nets include Petri nets [Pet62] and Bayesian network [Jen96]. Petri nets can be used for synchronizing (stationary) processes (e.g., Arrival of pedestrians to an elevator, where the elevator shall move only when the pedestrians enter the elevator).

4.2.2 Input Modeling and Random Functions

Input data provide the driving force for a simulation model. Random functions are used to generate input data for a simulation model. Algorithmically generated random numbers must be uniform and independent. A random variable X that generates a uniformly distributed random number in the range $[0, 1]$ is the fundamental building block of random variate functions. The Linear Congruential Pseudo Random Number Generator (LCPNG) proposed by D.Lehmer in 1948 used to generate uniformly distributed random numbers in the range $[0, 1]$. [L'E90] gives an overview of random numbers to be used in simulations. Often in practical situations, the input data is not uniformly distributed. Processes such as inter-arrival times, arrival rates, service times, etc., follow a specific pattern.

In principle, such distributions can be obtained by transforming uniform random numbers by using the inverse transform method. In many cases, finding an inverse function is difficult. Therefore several methods such as convolution, acceptance-rejection methods, etc., can be used to derive the random function. One type of non-uniform distribution often used for the input modeling in the reference scenario is the use of different probabilities among k group of alternatives. For example, a pedestrian decides to visit one among k different destinations in the scenario. Each destination has a different priority of visits. Therefore, the probability of choosing one among k alternatives is $p_1, p_2, \dots, p_k$. Then, the selection process would consist of generating a random variable R uniformly distributed between 0 and 1, followed by the comparisons of X against various values of probabilities such that

$$X = \begin{cases} 1, & \text{if } 0 \leq R < p_1, \\ 2, & \text{if } p_1 \leq R < p_1 + p_2, \\ \dots & \\ k, & \text{if } p_1 + p_2 + \dots + p_{k-1} \leq R < 1. \end{cases}$$

In this case, the probabilities must be adjusted such that $\sum_{i=1}^{k} p_i = 1$. This method can be replaced with a simple and fast general technique known as *alias method* [Wal77].

In a queuing system, the inter-arrival time is negative exponentially distributed, if the arrival process is a Markovian process. The negative exponential distribution can be determined by the inverse transform method. In an inverse transform method, if R is a random variable uniformly distributed between 0 and 1 and $F(x)$ is a strictly increasing distribution function, then the random variable $X = F^{-1}(R)$ has the cumulative distribution of $F(x)$, where F^{-1} represents the inverse function corresponding to F. In a negative exponential distribution, the function $F(x) = 1 - e^{-\lambda x}$. The inverse of

$$F(x) \text{ is } u = 1 - e^{-\lambda x} \Longleftrightarrow x = -\frac{1}{\lambda} \ln(1 - u),$$

where u is uniformly distributed between 0 and 1.

Another commonly used distribution (sometimes used to generate service times for the pedestrian) is the Gaussian or the normal distribution. The Box-Müller [BM58] method is used to generate normally distributed random numbers. Let $N(\mu, \sigma)$ be normally distributed with mean μ and standard deviation σ. Then,

$$x_1 = \mu + \sqrt{-2\sigma \ln(u_1)} \cdot \cos(2\pi \cdot u_2) \text{ and}$$
$$x_2 = \mu + \sqrt{-2\sigma \ln(u_1)} \cdot \sin(2\pi \cdot u_2),$$

where u_1 and u_2 are two random variables that are uniformly distributed between 0 and 1. In a Box-Müller method, one iteration results in 2 normally distributed numbers.

In the simulation of a queuing system, typical input data are the distributions of time between arrivals and service times. Such data mostly originate from real world systems (e.g., camera and sensors to detect the number of pedestrians entering the supermarket). These statistical data are then converted to mathematical functions that generate input data for the simulation, whose pattern is similar to that of the data collected from the real world system. Input modeling process is done in different stages.

- Statistical data collection from a real system.
- Identification of a probability distribution that best suits the real system.
- Parameter definition for the distribution function.
- Validation and evaluation of the distribution function with fitness tests.

The data collection, even though straightforward, is a tedious process. Sometimes it is not possible to collect the exact required data in which case expert opinion is sought to identify the function that best describes the system. Once the data is available, a distribution function must be chosen from the family of distributions. One method to do this is the use of histograms. In a histogram, the input data is ordered according to the frequency of occurrence to identify the shape of the distribution. For example, the probability of a visit to a specific destination varies during the simulation. Discrete time periods (1 hour) are taken and the number of visits for each hour is sorted as probabilities during the day. Figure 4.8 shows the probability distribution of visit probabilities to a restaurant during different times of the day.

Once the histogram is available, a suitable distribution is chosen that fits the histogram (or in other words the curve that fits the shape of the histogram). Literally hundreds of probability distributions have been created, many targeted for some specific physical process. Some distributions include:

Binomial Models the number of successes in n trials, when the trials are independent with common success probability p.

Poisson Models the number of independent events that occur in a fixed amount of time (e.g., number of pedestrian arrivals in an hour)

Normal Models the distribution of a process, which is more or less consistent (e.g., time required to execute the same task by different entities)

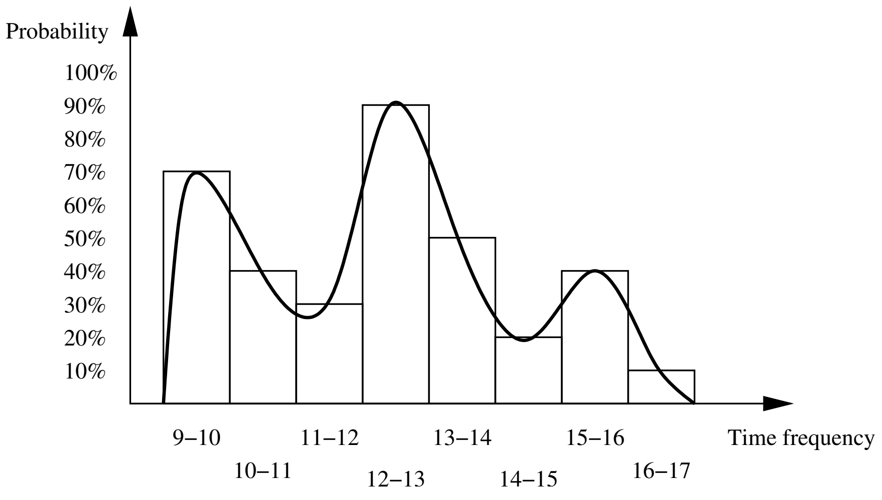


Figure 4.8: Histogram of the visit probabilities to a restaurant during different times of the day.

Exponential Models the time between independent events, or a process time with is memoryless (e.g., inter-arrival times of pedestrians)

Empirical Resamples from the actual data collected that is often used when no theoretical distribution is appropriate (e.g., decision of one among k destinations, each independent and with different probabilities)

However, it is not always possible to match the shape of the curve to an existing distribution function. The curve of the histogram in Figure 4.8 for example does not match any of the listed distributions. Therefore, curve-fitting functions are used to determine an appropriate probability distribution function for the histogram. The curve in the figure partially resembles a sinusoidal curve. Therefore, the resulting function can contain a combination of sin and cos functions. The curve function of the of the curve in the figure is determined to be $0.002 \cdot \sin(x) - 0.060 \cdot \cos(x) + 0.525$[4].

Several commercial tools such as Mathworks Curve Fitting Toolbox, CurveFit, etc., can take in values from data collection and give out appropriate functions to generate random numbers that best suit the situation. Finally, once a distribution is chosen, it must be validated. This means that the random numbers generated from the distribution function must be similar to the collected data and they must lie within the tolerant range. Goodness-of-fit tests are used to validate the distribution function.

[4]An on-line curve fitting function generator to translate histogram data to numerical functions is available at `http://www.softintegration.com/`

To model and simulate pedestrian behavior, different functions that generate appropriate random numbers were written. For the input modeling, several statistics regarding pedestrian behavior were collected and the random function was implemented by using the input modeling functions. The following section discusses the pedestrian parameters collected and the random number generators used to translate the pedestrian statistics to an input for the pedestrian simulation.

4.2.3 Analysis of Pedestrian Characteristics

In a typical pedestrian model, the scenario consists of several types of pedestrians, each with different characteristics and motives. Since the pedestrian behavior creates a significant impact on the model, a background analysis of the pedestrian behavior in a given scenario is necessary. Such an analysis shall provide us with the statistics and parameters necessary for modeling the input data for the simulation. The amount of required pedestrian behavior details depends on the actual situation of the model. For example, when modeling a cash counter in a supermarket, parameters such as the average number of customers in the queue (to measure queue length) and the average number of goods purchased by the customer (to measure the service time) are required. Other parameters such as the average time it takes for a customer to collect his goods, his walking speed, etc., are not directly related to the cash counter and can hence be ignored. In the past, several studies on microscopic pedestrian characteristics have been made and many statistics on pedestrian behavior have been published. For example, the Highway Capacity Manual and the journal of Transportation Research Record published by the transportation research board[5] are a good source of statistics of pedestrian characteristics and behavior. [Boa85, OM81] also gives us the necessary information on pedestrian characteristics. For the reference scenario, we choose some of the already available statistics that are best suited for the pedestrian simulation and use these data for the input modeling function. The parameters required for modeling pedestrian behavior as well as the statistics obtained from the respective sources are listed later in this section.

Apart from pedestrian analysis, pedestrian simulation also involves pedestrian data collection. Typically, pedestrian data collection is done manually (counting or use of cameras) or automatically (video surveillance and image processing). With the technological advance, use of computer methods and video devices have been increasingly common over the past decade. To facilitate design improvement of pedestrian facilities, surveys are often carried out at intersections, long

[5]`http://www.trb.org/`

pathways, waiting areas, crossings, etc. Many methods such as manual counting, photo beam, video processing, video surveillance, etc., [oTE94, LTP$^+$90, TSKT95] have been developed to automatically collect macroscopic flow characteristic data. Image processing methods[6] are used to recognize moving human beings. The tracked data is used to calculate the object position, path of movement, velocity and acceleration of the pedestrians.

In contrast to a vehicular traffic movement, pedestrians generally do not follow lane discipline[7]. Activities such as abrupt stopping, rapid acceleration and deceleration, unexpected change in direction, etc., make modeling a pedestrian behavior complicated. However, the graph used to model such a scenario is extracted in such a way that each path has a closed region around itself and only bi-directional movement is possible. We are concerned with basic properties such as flow, density and speed, and we shall ignore the other details. The fundamental characteristics of pedestrian traffic – flow, density and speed – can be analyzed at macroscopic (group of pedestrians) or at microscopic (each individual) level. In a macroscopic level, the flow rate Q (or the volume) is denoted by the number of pedestrians that pass through a horizontal line across the path for a specific period of time. The walking speed along a pathway was found to be Gaussian distributed [Hen71, HL72]. The average speed v can be computed either at the horizontal cross section line (called the *time mean speed*) or by calculating the average walking time for a specific length of the path (called the *space mean speed*). The density $k\ ped/m^2$ of the pedestrians along the path influence the pedestrian speed and flow. The fundamental traffic flow is denoted by $Q = v \cdot k$. The US Highway Capacity manual [Boa85] shows the relation of space, average speed and flow rate for different levels of service[8] (see Table 4.1).

Also, [Fru71a, Fru71b] suggests that pedestrians are able to walk at their characteristic speed if the density is below 0.5 ped/m^2. [O'F97] summarized the

[6]Movement can be tracked using the walking rhythm (spatial and temporal frequency of the foot movement to differentiate between pedestrians and other moving objects) of the object. Fourier transforms are then applied on the time series binary data. The components are then matched with the rhythm of the walking. For more information, see [YM94]. Similarly, making snapshots for frequent time interval can be used to identify the position displacement by calculating the difference between the images and performing edge detection on them (see [SRS$^+$95]).

[7]In crowded situations, the pedestrian movement resemble a swarm movement. An automatic lane formation can be seen in crowded pedestrian movements. However, in contrast with a 1-dimensional vehicular traffic movement, pedestrian lane formation is a 2-dimensional phenomena

[8]The levels of service, indicated with A,B,C,. . . , classifies the pedestrian activities based upon the area of occupation (or the density of the pedestrians). The levels of service does not mention anything about pedestrian mobility or safety. For more explanation to the levels of service, see `http://www.walksf.org/pedestrianLOS.html`

Level of service	space (m^2/ped)	Average speed (m/s)	Flow rate ($ped/s/m$)
A	≥ 12.077	≥ 1.321	≤ 0.196
B	≥ 3.716	≥ 1.270	≤ 0.383
C	≥ 2.230	≥ 1.291	≤ 0.547
D	≥ 1.394	≥ 1.143	≤ 0.820
E	≥ 0.557	≥ 0.762	≤ 1.367
F	< 0.557	< 0.762	variable

Table 4.1: Pedestrian Level of Service on Walkway (source [Boa85]).

walking speed along a wide pathway and has indicated that the average speed value lies in the range of 1.2 m/s to 1.35 m/s with a mix of pedestrian age groups. However, if the crossings are free of congestions, the average walking speed approximates to a free-flow walking speed of 1.6 m/s. For disabled and elderly pedestrians, the walking speed is found to be 0.5 m/s.

The pedestrian characteristics listed so far show that the behavior is different in different situations (fast movement when crossing the road to slow walking pace when doing window shopping). Certain physical experiments such as measurement of walking time along pathways, stairs, long walks, etc., were also made in the geometric scenario used here (in the computer science building) and the outcome were used to calibrate the available data and also to implement functions, where the existing data cannot be used. For example, it was found that the walking speed along the staircase is roughly 50% of the walking speed along the corridor.

The objective is to analyze the congestions caused by the pedestrians along the path and the waiting lines at the destination nodes. The following parameters are therefore necessary to model the pedestrian behavior.

1. **Walking speed**: The pedestrian walking speed is derived from the above listed statistics. The walking speed is the only parameter used in the reference scenario that is derived from external statistics. The pedestrian walking speed and acceleration plays a major role in causing congestions along the paths of the scenario. Walking speed is greatly influenced by the density of the pedestrians along the path, the type (age) of the pedestrians and the bi-directional movement along a given path. Since the path is extracted such that there is a closed geometry around it, the movement of the pedestrians is either uni-directional or bi-directional, i.e. only along the path.

[Tek02] suggests that the average walking speed of a pedestrian is normally distributed with $\mu = 1.38\ m/s$ and $\sigma = 0.37\ m/s$. The acceleration is normally distributed with an average of $0.68\ m/s^2$. From the above-mentioned statistics, [Tek02] also proposes that the density dependent speed is linear in case of a uni-directional movement and logarithmic in case of bi-directional movement. Furthermore, it was found that more than a few pedestrians of older age in the path already tend to reduce the walking speed of all pedestrians in that path.

In the reference scenario, we generate the walking speed of the pedestrian using a Gaussian distribution $N(1.38, 0.37)$ and induce a speed factor to it. The speed factor is determined by calculating the density along the path (for both directions), age and mobility factor of the pedestrian, and also the pedestrian characteristics, along with the above listed statistics.

2. **Arrival of the visitors**: The arrival of pedestrians is a Poisson process. The inter-arrival times are negative exponentially distributed. The arrival rate of the pedestrians may dependent on the time. Therefore, time dependency model, as explained in 4.2.1.1, was implemented with different arrival rates for different time frames.

3. **Type and number of visitors**: As mentioned earlier, in any given scenario, different types of pedestrians can be generally expected. Apart from that, each type of pedestrian will have restrictions such as where to go and where not. A flexible database that can store an arbitrary number of pedestrian profiles along with the necessary properties is developed. The pedestrian profile definition is addressed in the next section. The number of visitors depends on the scenario to be modeled and simulated. The parameter for the number of visitors, sorted according to their types, is made flexible such that any scenario can be simulated. Furthermore, the distribution of the different types of pedestrians can also be specified.

4.2.4 Profile and Parameter Modeling

When modeling the input data for the pedestrian simulation, it is also necessary to consider the objectives of each pedestrian in the scenario apart from his characteristics. Therefore, we define many classes of pedestrian profiles that suit different pedestrian characteristics. When defining the profile for use in the pedestrian simulation, the following parameters are taken into account.

- An unique `ID` to identify the pedestrian: To define and track the pedestrian movement throughout the simulation.

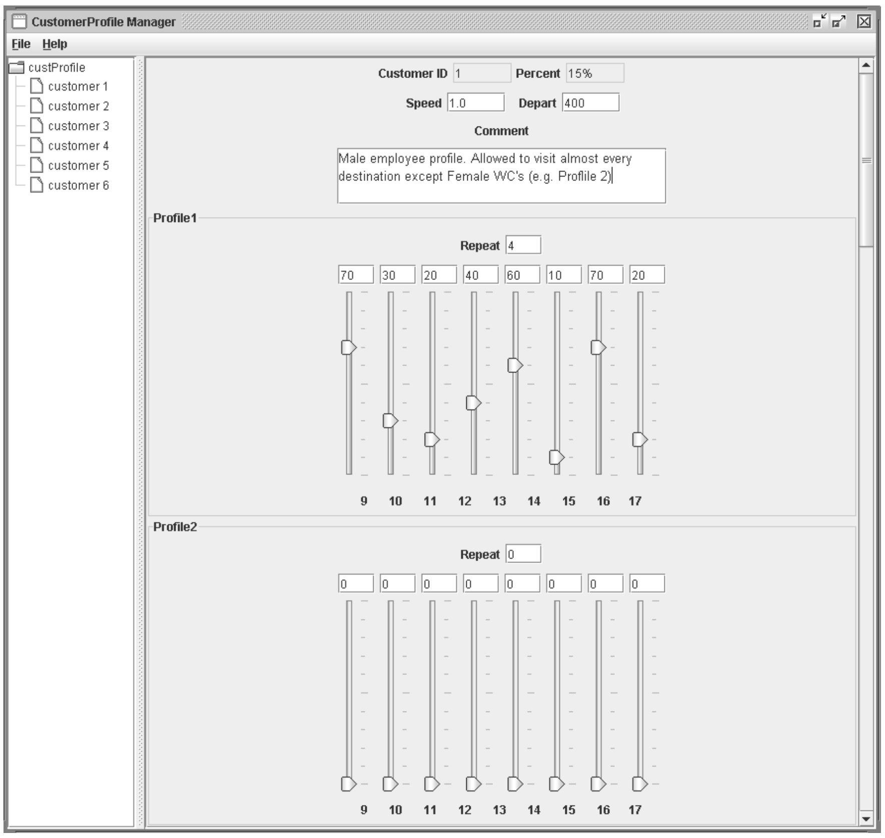
- Average duration of stay in the scenario: To differentiate short term and long term visitors (e.g., an employee tends to stay throughout the day whereas a visitor would stay for a shorter time).
- Age (speed) factor: To determine the average walking speed of the pedestrian.
- Restrictions imposed: A pedestrian may not be allowed to visit every destination or walk along every path.
- Tasks to execute: Each pedestrian intends to execute a certain number of tasks in the scenario. Therefore, the profile includes the destinations, which a pedestrian will visit, repetitions to the destination and the probability of visit during different times of the day. Certain tasks may be time dependent (stationary process). Therefore, such tasks are indicated with the time range the task should be executed as well as the service time of the task.
- Distribution of pedestrian types: Once a pool of profiles is available, the total number of visitors is set along with the distribution of the chosen profiles.

A pedestrian profile management interface was developed to define and store an arbitrary number of pedestrian profiles. With an exhaustive list of pedestrian profiles, it is possible to choose from the set of profiles and build the require scenario. Table 4.2 shows a hypothetical pedestrian profile distribution in an office building. Furthermore, additions or changes to the profile database are also possible.

Profile type	Distribution
Employee_1	10%
Employee_2	30%
...	...
Visitor_1	12%
Total	100%

Table 4.2: Hypothetical profile distribution in an office building simulation.

The pedestrian profile manager has a graphical interface for defining pedestrian profiles. Every time a new profile is created, the objective of the pedestrian (destinations to visit or the tasks to execute) are also defined together with the profile. An `ID` is automatically defined and is associated with the profile. Then parameters such as the average duration of the stay and the speed factor (representing the age) are also defined. The list of tasks executed by the pedestrian is determined stochastically from the profile. For this purpose, the probabilities of visit to each destination are also defined. The graph extracted from the CAD model, shown in chapter 3 contains the list of destinations and its types. Each destination was classified in one of the destination types listed. In the profile, for each destination type, the probability of visit during different times of the day is defined. By defining a probability of 0.0, it implies that the pedestrian will not visit the destination. The definition of the probabilities can be done using a graphical interface. Discrete values (in this case the average probability for each hour) are set and a histogram of the visit probability is obtained. Furthermore, the average number of repetitions for each destination type is also set. Figure 4.9 shows a snapshot of the pedestrian profile manager and the interface to set probability histogram for the destination. Similarly, several such profiles can be created and stored in the database.



Customer ID 1 Percent 15%
Speed 1.0 Depart 400
Comment
Male employee profile. Allowed to visit almost every destination except Female WC's (e.g. Profile 2)

Profile1
Repeat 4
70 30 20 40 60 10 70 20

Profile2
Repeat 0
0 0 0 0 0 0 0 0

Figure 4.9: A snapshot of the pedestrian profile management interface used to define, modify and create profiles for simulation.

Once a pool of profiles is available, the interface also allows us to define a specific scenario for simulation. In this interface, selected profiles that suit the necessary scenario are chosen and their distributions are also specified (similar to Table 4.2). The pedestrian management interface provides an option to specify the distributions of the chosen profile. By default, the distribution of each profile is set to 0%. The user then interactively defines the distribution of each of the profile such that the final profile distribution is adjusted to 100%. The final scenario for input modeling is then exported as XML data that can be easily parsed to produce the necessary input for the pedestrian simulation.

Another parameter necessary for pedestrian simulation is the service time at the destination. The service time at the destination is already embedded into the graph data (contained in the room properties) during the graph extraction.

During the initialization of the simulation, the XML data of the scenario is first parsed and the properties of each profile are identified. Every time a new pedestrian arrives, the type of pedestrian is chosen stochastically from the histogram model of the distribution listed in the profile data. This process is done during the simulation. Once a specific type of profile is chosen, the list of destinations and the repetitions are calculated immediately. During the simulation, when the pedestrian has to execute a task, the choice of destination is chosen stochastically based on the actual simulation time and the visit probabilities. As soon as a destination is chosen, the destination list attached with the pedestrian profile is updated.

4.2.5 Geometry Embedding of Queuing Systems

With the input data available so far, we model the pedestrian behavior using queuing systems and embed the model in the geometry context. As discussed earlier in 3.2, we model two pedestrian activities, namely the movement of the pedestrians along the paths, and the pedestrian behavior at the destination nodes. Each edge of the graph and destination identified from the CAD model is replaced with a queuing system.

4.2.5.1 Queuing System for Paths

The pedestrian movement along the path can be thought as a queue, where the pedestrians arrive at a path, wait if the path is congested and then move once the congestion reduces. The following parameters are therefore needed to model the queuing system.

Arrival: The initial arrival time to the scenario is negative exponentially distributed. The subsequent arrival time to each path is the time when the pedestrian leaves the previous path.

Service: We define service to be the actual movement along the path. Therefore, the service time is calculated using walking speed and distance of the path. The number of service units SU or resources is determined from the capacity of the path. As mentioned earlier, the walking speed is dependent on the density and the direction of movement along the path. Before scheduling the service event, the current status of the allocated resources is identified to calculate the walking speed. The acceleration parameters and also the characteristics of the pedestrians who are already present in the path are also considered for service time.

Queue: Since the number of service units is same as the capacity of the path, the queue size is theoretically zero. A pedestrian can only arrive if a resource is available. However, for modeling purposes, the queue capacity is set to be infinite $Q_{\max} := \infty$ and new arrivals are appended in the queue only if $SU = idle$. Once appended, the pedestrians are served immediately. If the number of pedestrians exceeds the path capacity, a backlog is automatically formed until the starting node.

4.2.5.2 Queuing System for Destinations

The pedestrian behavior at the destination is straightforward. The pedestrians arrive at the destination, wait if all the service units are occupied and then execute the task once a service unit is free. Following parameters are therefore needed to model the queuing system.

Arrival: The arrival time is the time when the pedestrian departs from the path just before the destination. Once the pedestrian arrives, he is immediately appended into the queue.

Service: The service time is generated stochastically based on the type of the destination (see chapter 3 and section 3.2.1.2). The service time is independent of the status of the resources as well as the queue. In case of stationary processes, the time of departure from the destination is taken into account to determine the service time. That is, if the pedestrian arrives earlier than the intended start of the activity and the queue is empty, then there is an additional service time until the start of the activity (waiting at the service counter).

Queue: The queue capacity is set to be infinite $Q_{\max} := \infty$. However, priorities and queue behavior are decided from the pedestrian profiles. But such behavior takes effect only when the pedestrian arrives at the queue.

Queue Behavior: Parameters such as balking, reneging and service preemption are decided from the customer profiles. In some cases, the type of destination is also used to influence the queue behavior. For example, if the waiting line at a restaurant is long, the pedestrian may consider returning at a later point of time.

4.2.5.3 Queuing Network

Once each path and node is represented using a queuing system, they are interconnected – using the graph – to form a queuing network. Whenever a pedestrian chooses his path, he transits through all the queuing systems along the path before reaching the destination. The customer profile generated from the pedestrian profile manager is used for the simulation. Once the pedestrian profile is initiated, the simulation starts with the arrival of the pedestrian. The destination to visit is determined at random by considering the probability distribution in the pedestrian profile. The pedestrian then chooses the path to the destination and walk through all edges (or wait at the corresponding queuing systems) until he reaches the destination. Once the task at the destination is complete, the next destination to visit is generated. This process is repeated until all tasks are executed or the departure time (terminating condition) has been reached.

4.2.5.4 Pedestrian Simulation

The pedestrian simulation begins with the initialization phase in the following stages.

- **Parsing of the geometry parameters**: The graph and room data are parsed from the XML data and a queue is initialized for each room (node that has a room defined to it) and the edge (path). In case of an edge, 2 queues for each direction are initialized. Since the direction is only counted for calculating the walking speed (service time for a path), the pedestrians in both directions pass through a single queuing system.

- **Pedestrian data**: Arrival of new pedestrians are not fixed in advance, but are generated dynamically during the simulation. Each time a new pedestrian is initialized, the type of the pedestrian (from the pedestrian profile) is decided and a list of destinations along with the probability distributions is attached with the object. The departure time is also fixed and the pedestrian leaves the scenario when the departure time has arrived or when there are no more tasks to execute, whichever is earlier.

Once the initialization phase is complete, the simulation starts with the arrival of the first pedestrian and the arrival event is *scheduled* at the time of arrival. For each arrival, a service event is bound to occur. Since the status of the service units is not known at the moment, the service event is set to *hold*. Just before the arrival, the first destination to visit is (stochastically) decided and the pedestrian is appended in the corresponding queue of the network. During the simulation, the current location of the pedestrian is continuously tracked such that the pedestrian is appended to the right queuing system – can be a path or a destination. After every visit, the pedestrian object, which contains the probability distribution and the list of destinations, is updated.

Arrival Event: In case of an arrival to a path, the pedestrian is appended in the path queue: $append(e, Q_{p_i})$. In case of an arrival to the destination, the pedestrian is appended to the destination queue: $append(e, T_i)$. To keep the simulation running, the next arrival is already scheduled each time an arrival event occurs. Algorithm 5 shows the implementation of the arrival event.

Algorithm 5 Arrival Class

```
1: **if** arrival type path **then**
2:     identify path queue $Q_i^p$
3:     identify direction $D$
4:     append($e, Q_i^p, D$)
5:     set $Q_i$ visited
6: **else if** arrival type destination **then**
7:     reset path list
8:     set current position (node)
9:     append($e, Q_i^d$) {*append to destination queue*}
10: **end if**
11: **while** new customers exist **do**
12:     schedule(arrival,$e, T_{at}$)
13: **end while**
```

Service Event: In case of a service process, the availability of resources (path density and service units at the destination for a service event in a path and destination respectively) is first verified. As soon as a service unit is available, the entity is removed from the queue and served. The service unit is then set to be *busy*. A departure event is then scheduled at $t_{\text{current}} + S_t$. If no resource is available, the event continues to be in *hold* state[9]. The algorithm 6 shows the implementation of the service event.

[9]A hold state takes a passive or active event and adds it to the conditional list. This event the fires after each activated event

Algorithm 6 Service Class

```
1:  if path then
2:      while $Q_i^p \neg empty \ \wedge \ SU_{p,i} = free$ do
3:          remove($Q_i^p, e$)
4:          $SU_{p,i} = busy$
5:          schedule(departure,$e, S_{p,t}$) {$S_t$ calculated using walking speed}
6:      end while
7:  end if
8:  if destination then
9:      while $Q_i^d \neg empty \ \wedge \ SU_{d,i} = free$ do
10:         remove($Q_i^d, e$)
11:         $SU_{d,i} = busy$
12:         schedule(departure,$e, S_{d,t}$) {$S_t$ calculated using destination properties}
13:     end while
14: end if
```

Departure Event: In a departure event, the service is complete and therefore, the resource is reset to available. If the departure occurs in a path, the next task is identified and scheduled. This may be the next path (schedule pedestrian arrival into the path) or the destination (append the pedestrian into the service unit) or the exit (remove the pedestrian). If the departure unit occurs at the service unit, the next destination and the path to it is determined, and the pedestrian is scheduled in the path. The algorithm 7 shows the implementation of the departure event.

As one can see, each event automatically triggers a new event. The cycle is complete only when the pedestrian leaves the scenario after executing his tasks. Since new arrivals are also scheduled at the start, the simulation executes the complete chain of events. A simulation, once started has a termination condition (e.g., terminate after 480 time steps (`simulate(480)`), where 480 time steps assume 1 minute per time step or 8 hours of real time). The simulation also terminates if there are no more events to execute. Throughout the simulation, output data concerning the queue size and congestions are collected. Once the simulation terminates, the output data gives us an overview of congestions and waiting times for each path and destination throughout the simulation period.

A flexible simulation framework is therefore built, with which one can define the necessary simulation profile, model and simulate the pedestrian behavior within the graph of the CAD model, and finally obtain the congestion and waiting status of the scenario for the simulation duration.

Algorithm 7 Departure Class

```
1: if path then
2:     release $SU_{p,i}$
3:     if next=path then
4:         schedule(arrival,$e$, $Q_i^p$)
5:     else if next=destination then
6:         schedule(arrival,$e$, $Q_i^d$)
7:     else
8:         terminate($e$)
9:     end if
10: end if
11: if destination then
12:     release $SU_{d,i}$
13:     identify next destination $d_{next}$
14:     calculate path $p$ to $d_{next}$
15:     schedule(arrival,$e$, $Q_i^p$)
16: end if
```

4.2.6 Analysis of Pedestrian Simulation

The major computational part in a discrete event pedestrian simulation is the event scheduling part. As mentioned earlier in section 4.2.1.3, events (also the conditional events) are inserted in a tertiary tree and the scheduling is done in chronological order. The insertion operation has an average complexity of $\mathcal{O}(\log(n))$ and a worst case complexity of $\mathcal{O}(n)$. For the conditional events stored in tertiary tree in the form of a doubly linked list, the insertion or removal of an event can be performed in constant time $\mathcal{O}(1)$. To schedule (and execute) the events, both the event list and the conditional event list is searched. Since the tertiary tree or the event list is represented as a sorted binary tree, the complexity of searching an event is $\mathcal{O}(h)$ where h is the height of the tree. However, if all previously occurred events are deleted from the list, the next event will always be located at the root of the tree. Therefore, the search for the next event in this case is performed in constant time. The run time efficiency of a discrete event pedestrian simulation depends mainly on the performance of the scheduler. The scheduler simply inserts the events in the tree and lists them out in the increasing order. Therefore, the discrete event methodology is fast and efficient. For example, consider a pedestrian simulation scenario with 2000 pedestrians visiting the scenario, with a maximum of 300 pedestrians present in the scenario, and with a graph of 600 edges and 540 nodes. The simulation took 61 seconds to execute the events and generate profiler data, without including the time to write out the simulation data. About 364,500 events were created. From the profiling data on the run time statistics of the simulation, it was found that the discrete event simulation consumes only 17% of the total computing time. An astounding 67% of the computing time was required to perform graph operations such as extraction of architectural data, path-search algorithm (consumed the most amount of time), etc. The remaining amount of computing time was utilized by the initialization phase where the XML data is parsed and the initial structures for the pedestrian simulation are defined.

Simulation for different problem sizes were performed and the memory requirement, as well as the computing time (only the simulation time) of the simulation, performed on a Intel Pentium 4 processor with 3 GHz processor speed and 2GB main memory, are listed in Table 4.3.

Maximum capacity of the model	Total number of pedestrians	Computing time	Memory required
100	359	7 *s*	
1000	2917	48 *s*	
10000	21437	230 *s*	200 MB approx.
100000	113052	31 *min*	< 2 GB

Table 4.3: Analysis of the computing time and memory requirements for a pedestrian simulation performed for different number of pedestrians.

4.3 A Pedestrian Simulation Framework

The concepts and components explained so far are integrated within a single pedestrian simulation framework such that a given scenario can be modeled and simulated to obtain an overview of possible congestions and waiting times across the scenario. The framework is designed to be as flexible as possible such that the user shall be able to define any generic scenario, define the input and environmental parameters, simulate the scenario and analyze the results from the simulation. In this section, the various components that are built within the framework as well as certain extensions to the framework are discussed. An outlook into similar pedestrian simulation environments existing elsewhere and a comparison with them is made.

4.3.1 Framework Components

The components integrated into the framework as well as its performance and functionalities are listed as follows.

4.3.1.1 Pedestrian Profile

The pedestrian management interface explained in 4.2.4 is generally used to define the pedestrian profile. The interface uses an XML parser to read from and transfer to the profile database. The profile manager manages a master database, where several pedestrian profiles are stored. New additions or changes to the existing profiles are done using the interface. When defining a specific pedestrian profile scenario for the simulation, selected profiles are chosen from the profile database and are exported separately. Along with the profile list, a profile distribution data, similar to Table 4.2 is also generated. The pedestrian simulation code then has access only to a subset of the master profile list. This process minimizes the read/write activities performed from within the simulation and thereby speeds up the simulation process.

4.3.1.2 Graph Profile

Another functionality added together with the pedestrian profile manager is a similar profile manager to edit and manipulate graph data. Since the graph alone is used for simulation, minor geometrical changes to the scenario can be made using this interface. The graph extracted from the CAD model does not contain information regarding room capacity, number of service counters, path restrictions etc. Since these data are additionally defined and stored into the graph, different scenarios can be simulated by simply adjusting the values from the graph. This method is particularly useful when optimizing a building plan structure. Interchange of two rooms can be easily done by just exchanging the room properties. Therefore, the layout and the capacities of the building can be quickly altered using the graph and room profile manager.

When optimizing the architectural layout of the building (before the construction of the building), the scenario can be simulated iteratively to detect bottlenecks along the paths determine an optimal path capacity for the given scenario. Even though the path capacities – based on the distance to the wall – are automatically extracted from the CAD model, changes can be made to the graph through the graph management interface by altering the path capacities. The optimized layout is then used to redesign the architectural model such that it suits the given scenario in that building.

However, changes in architectural design means changing the structure of the building. The structural engineering mechanism must also be considered when making such changes, which is currently not considered within this framework. One option is to use a common geometric modeling database for distributed applications such that each application has access to the same model. The geometry model is then modified in such a way that it is suitable for all applications that requires the geometry model [MB04].

The pedestrian and the graph profile manager together can be used to build and alter a generic pedestrian simulation scenario.

4.3.1.3 Pedestrian Simulation

The `SIM`: a C++ library for discrete event simulation [BE95] was partially used (event scheduler and the tertiary tree structure for queues and event list) to model and simulate the pedestrian behavior for reference scenario. The 3-phase approach shown in Figure 4.2 was implemented in C in the `CSIM` discrete event simulation library [Wat93]. Apparently, [BE95] implemented a C++ version of the `CSIM` library by integrating the `HUSH` graphics library [Eli95] such that graphical analysis of the simulation can be made. Appendix A demonstrates the method to build and simulate elementary queuing system (a single server queue, where pedestrians arrive, wait and execute their task) using the `SIM` library.

The individual queuing systems and thereby the queuing network is built automatically within the framework during initialization phase of the simulation. The parameters such as number of queuing systems, properties, etc., are determined from the graph data. Therefore, the implementation of the queuing network model is basically the same for any scenario that is simulated within the framework. When building the queuing systems, the program allocates the required amount of memory for the scheduler and the event list (tertiary tree). An event-based approach[10] is used to model and simulate the pedestrian behavior.

The queuing network follows a Markov chain process where each pedestrian makes a decision on the next destination and the decision is a Poisson process depending on the probability distribution of the pedestrian profile. Other stochastic processes such as stationary process can be implemented within this framework. Such processes are performed based on the definitions and information available

[10]In an event-based approach, we first identify the events (arrival, departure, service) in the model. The behavior of an event is implemented by deriving it from the class *event* and overriding the *function operator* of this class. On the other hand, in a process-oriented approach, the components of the model consists of entities, which represent the existence of some object in the system (pedestrians). An entity receives a user-defined phase that determines the behavior of the entity

from the pedestrian profiles.

The entire simulation is driven by the pedestrian profile that is used for simulation. The profile management interface shown in section 4.2.4 is used to define the pedestrian profile for the scenario. The profile interface is at the moment a separate component and is not integrated within the simulation framework. Even though the choice of pedestrian type, the decision of visiting a particular destination, etc., are determined stochastically within the simulation framework, the parameters to make such decision lie with the pedestrian profile. Properties such as queue priorities, visit priorities, queue behavior (reneging, balking), are available within the simulation framework and can therefore be activated when necessary. However, the profile management interface does not currently support such properties to be defined within the profile and must therefore be defined separately in addition to the profile, if such decisions have to be made by the pedestrians.

The reference scenario simulated within this framework assumes that each pedestrian will determine a list of tasks to execute and the task is executed in the order of occurrence $T_i \prec T_{i+1}$ (each task is generated dynamically, i.e. the next task is decided once the current task is executed). For tasks that must be executed at a specific time for a certain period of time $T_i(t_{start}, t_{end})$ (e.g., stationary process), a time parameter is set additionally such that the pedestrian interrupts the current task to execute the new task.

$$set(t_{\text{start}});\; if(t_{\text{curr}} > t_{\text{start}})then\ \{\text{interrupt}(T_i);\text{execute}(T_{i+1});\}$$

In such situation, the time taken to reach the destination is also considered when performing $set(t_{start})$.

It is assumed that every pedestrian who visits the scenario arrives with an intention to execute a certain set of tasks. However, in reality, certain pedestrians wander within the scenario aimlessly. Such pedestrians also create congestions within the scenario. Such statistics are not directly incorporated within the framework. Instead, these characteristics can be integrated within the pedestrian profile by defining extra spots where such pedestrians may gather. Such aimless wandering can therefore be mapped to a pedestrian profile of an existing format and these pedestrians can be simulated along with the system. In this framework, the example of occasional smoking activity on the outside areas of the reference model was considered in the pedestrian profile.

Other parameters that can be controlled within the simulation framework include the (time-dependent) arrival rate, maximum amount of arrivals during the simulation and the maximum number of pedestrians that can be simulated simultaneously (denotes the maximum capacity of the building itself).

As mentioned in chapter 3, an XML parser library (LibXML2) is used to parse and manipulate the profile data stored in XML format. During the initialization phase, following operations are done.

Graph Data: The graph data consists of nodes (also co-ordinates, rooms), and edges (also type, length, capacity). The node and the edge data from the XML are parsed and stored separately in a linked list structure. The two lists are then interlinked such that the nodes point to the corresponding edges and vice-versa. Once the list is created, the XML contents are removed from the main memory. The graph list is retained until the simulation terminates and all graph operations are performed on this list. Typically, the graph is stored only once in the memory and the graph stays until the simulation terminates. Even though the memory requirements are not high (just the required memory to store the graph), it can be seen in section 4.2.6 that the operations performed on the graph such as path-search, destination decision, geometry parameter extraction, etc., consume a lot of computing power.

Room Data: The room data is organized similar to the graph data. Two individual lists are created for rooms (ID, capacity) and room types (type of the destination and typical service times). Once the list is created, the XML contents are removed from the main memory.

Customer Data: The customer class manages one customer at a time. For each instance, the object parses the customer profile designed for the simulation, chooses a specific customer type based on the type distribution and collects the properties of the chosen customer type. All other remaining data are unnecessary and are therefore deleted. Once a customer is chosen, a list of tasks to execute, along with the probability of time of occurrence during the simulation, is also generated and stored along with the object. As and when a task is executed, the task list in the customer object is updated. Compared to the graph and room data, the memory requirements for the customer is higher. This is because, for each new customer, the customer profile has to be parsed fully and a decision on a specific customer type is chosen. The customer record stays in the memory as long as the customer stays in the scenario. The memory requirements of a customer data increase linearly with the maximum allowed capacity of the geometric model.

4.3.1.4 Data Collection and Analysis

Output data is to be collected from the pedestrian simulation regarding the behavior of the pedestrian and their impact in the scenario. The framework offers the following possibilities to collect results from the simulation

- **Path congestion**: During the simulation, the congestion that occurs along the path is continuously monitored. Snapshot at discrete time stamps[11] are made and the congestion data is collected (e.g., for a simulation time of 8 hours, 480 observations are made). The congestions can be measured in terms of time (time it takes to walk along the entire path) and congestion (pedestrian density at a path).
- **Destination Congestion**: The destination congestions are monitored similar to that of the path congestion data. The congestions are measured in terms of time (time it takes to execute a task, including waiting time) and queue size (the average length of the queue). Both the path and destination data gives an overview of the congestions that occur in the scenario throughout the simulation.
- **Customer Data**: For each customer initiated, there is a trace file attached to the customer object that keeps track of each activity of the customer (movement, task execution, node visits) along with the time stamp of each event. The customer data is used for the visualization tool described later in this section.

Examples and analysis of the simulation data collection are presented in chapter 6

4.3.1.5 Pedestrian Data Visualization

A visualization environment named Observer was developed to observe and validate the pedestrian movement within the simulation scenario [Wu06]. The reference model used here was created using Maya software from Alias (now owned by Autodesk) [GSG04]. The model was then exported to the OBJ format[12] supported by Alias Wavefront. The GLM [Rob] – an OBJ file loader developed with the GLUT (OpenGL Utility Toolkit) library – is used to parse and visualize the geometric model and the pedestrian simulation. The visualizer is written using OpenGL and Trolltech's Qt. The Observer tool offers the following functionalities

[11]In this framework, values are measured for the time interval of 1 minute and a mean is calculated. This time frame is valid for both path congestions and destination waiting times. The time interval can be flexibly changed within the framework.

[12]Object files define the geometry and other properties for objects in Wavefront's Advanced Visualizer. Object files can also be used to transfer geometric data back and forth between the Advanced Visualizer and other applications. Object files can be in ASCII or binary format. The OBJ file supports both polygonal and freeform objects.

- **Model Display**: The OBJ formatted geometric model is parsed using the GLM library and is displayed within the Qt window. The textures and the colors are retained with the model. The transparency level can also be activated such that the simulation across the entire scenario as well as the model is visible. Transparency is adjusted by activating the blending level and the alpha component is used to blend the colors to give a transparent effect.
- **Navigation**: Functions such as rotate and zoom have been activated such that the viewpoint camera can be controlled interactively.
- **Simulation Visualizer**: The customer data collected from the simulation is used to visualize the pedestrians. Each pedestrian is represented as a sphere and the sphere moves along the edges of the graph as defined from the customer data collected from the simulation. The customer data include the motion of the pedestrian along every single path as long as the pedestrian is present in the building and the speed of movement (represented using color values).
- **Congestions**: By activating the graph, the graph is displayed and the edge color represents the pedestrian density. Similarly, the current status of queue size and occupation is also displayed.

All the above components can be activated or deactivated through the graphical user interface. A real time interactive visualization of the reference model (geometry model with about 156,000 triangles and a customer data consisting of 1000 customers) was possible on a PC with 3D Graphics Card (ATI Radeon X800) having 128MB RAM. However, larger models consisting of many such buildings may not be possible to visualize. Therefore, an option of transferring the visualization into a video file is also built such that an off-line visualization and analysis is possible. A virtual flight is also possible by pre-defining the desired path over which a flight has to be made. All the components listed so far can be combined or activated individually. The Figure 4.10 shows a snapshot of the Observer visualization tool used to visualize the pedestrian movement in the reference model.

4.3.1.6 Parallel and Distributed Simulation

When scaling the reference scenario and the reference model, where the scenario consists of a number of buildings spread across a large region, several times more number of pedestrians can be expected in the scenario. Even though the simulation framework theoretically allows such scalability, each pedestrian entity and

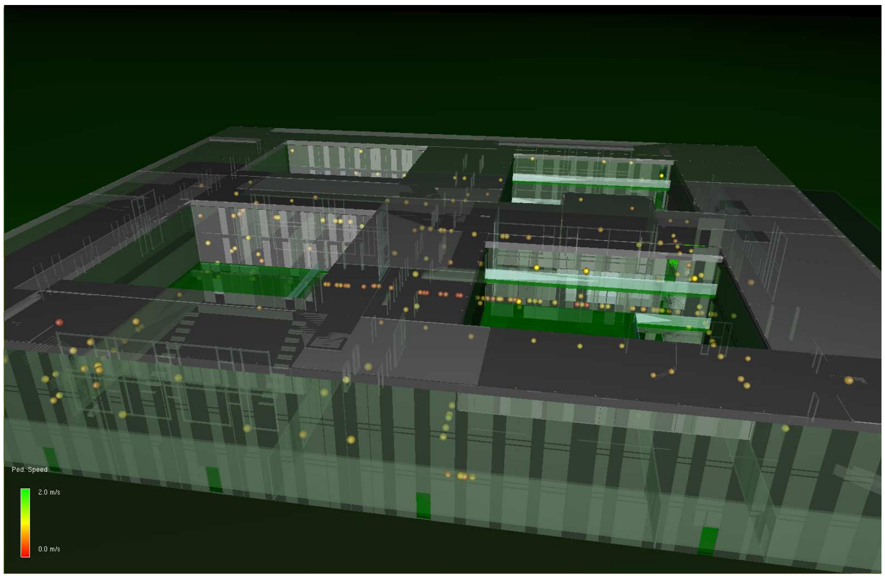


Figure 4.10: A snapshot of the visualization of the pedestrian movement (represented using colored balls where the color denotes the walking speed of the pedestrian) in the reference model shown as a translucent object.

the graph data require an enormous amount of memory (see analysis in 4.2.6). Therefore, parallel and distributed simulation methods must be used to tackle the computing restrictions.

Research has been done in the past to parallelize discrete event simulation [Fuj00, Fuj90, NF94]. Parallel Discrete Event Simulation (or PDES) broadly fall into two categories: conservative simulation [Mis86] and optimistic simulation [Jef85]. A conservative approach executes an event only if it is possible to avoid a causality error, i.e. the event should not be dependent on any future events. An optimistic approach on the other hand uses detection and recovery approach, i.e. the events are first performed and if a causality error is detected, the simulation is rolled back to its initial state or to the state where a causality error is detected such that the errors are corrected. Therefore, each state of the simulation must be saved and the amount of intermediate storage for an optimistic approach is very high. In both cases, the complexity of the methods are very high and for small improvement in performances, huge amount of efforts are needed. This is because, in a discrete event simulation, the events are scheduled in the event list in a chronological order and the events must be executed in the increasing order of the time. Some events that are already available in the future event list and are bound to certain conditions. Such events, even though available to be executed, cannot be executed since the status of the event depends on some other event that is not yet scheduled (or completed). Therefore, the amount of parallel executions in a PDES reduces drastically and the performance improvement is low and hence not used to implement queuing systems within this framework.

Alternatively, it is possible to partition larger graphs into smaller sub-graphs and simulated in parallel. For example, in a campus network of many buildings, each building can be stored in a separate sub-graph and each building is simulated separately. Since communication between the sub-graphs exists for pedestrians wishing to move from one building to another, the graph must be partitioned such that the connection between the sub-graphs is minimal.

Also, independent simulations such as a building plan optimization (simulation of several different scenarios within the same given CAD model) can be executed in parallel such that independent scenarios can be simulated on a parallel computer. Initial attempts were made within this framework to perform parallel independent simulations. A parallel code using MPI (Message Passing Interface) library was written to perform such parallel simulations. In the program, simulation specific parameters and the input data for independent simulations are stored separately. Since the simulation program is the same, the parallel code sends the input data for each simulation separately to different available processors. Once the simulation is complete, the results are collected and the simulation is terminated. This process is repeated until all the scenarios have been simulated. The parallel program schedules a new simulation scenario as soon as a processor is available and the data is collected.

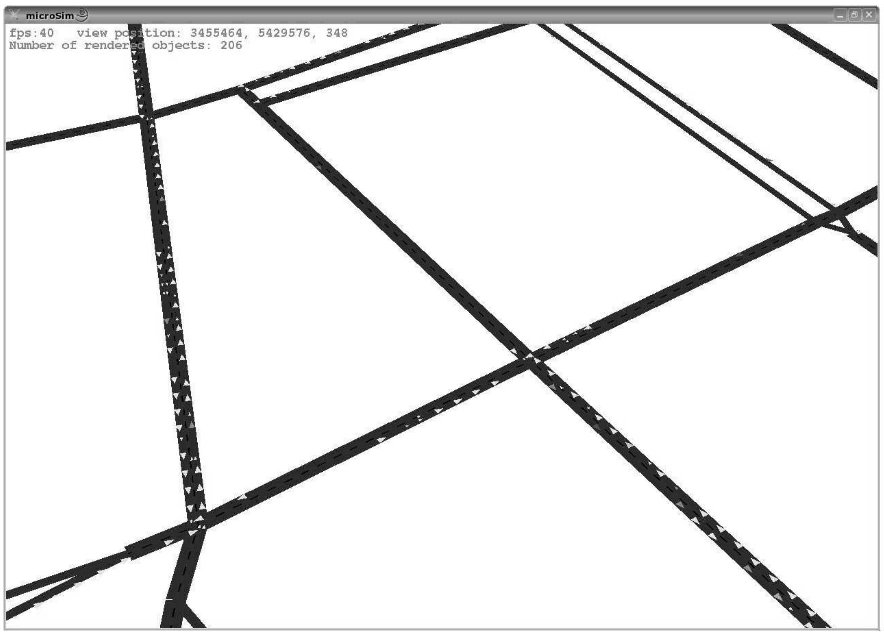
4.3.1.7 Geometric Modeling and Navigation

The component for the extraction of the graph from the CAD model as explained in chapter 3 is not included within the pedestrian simulation framework and is performed separately. However, the octree-based navigation system uses the data from the simulation to choose an optimal destination and navigate the pedestrian to that destination. In chapter 3, a list of destinations in the neighborhood is searched and the closest destination is chosen. However, the closest destination may not necessarily be the optimal destination. The congestion data obtained from the simulation can be used to sort the identified list of destinations based on the congestions along the path and the actual waiting times at the destination. The pedestrian can then choose an optimal destination that can be reached easily and also has the least waiting time. Numerical examples of such navigation systems that use the simulation data will be presented in chapter 6.

4.3.2 Pedestrian Simulation Framework Extensions

The simulation framework discussed here performs a pedestrian simulation for the defined scenario in the given geometric model. Different applications and connectivities are possible with this framework. In the reference scenario, we choose the application of a pedestrian navigation system to schedule the list of tasks for a new visitor. The simulation framework may also be used to simulate an evacuation scenario: normal evacuation once the closing time has reached (discussed with an example in chapter 6) or an emergency evacuation scenario, building plan optimization: optimize the architectural model by simulating the expected pedestrian scenario in the building, pedestrian visit estimate: where to expect more pedestrians visits and how to optimize the distribution of rooms in the model to optimize pedestrian movement, etc.

Apart from the different applications, the pedestrian simulation framework can also be integrated with the traditional traffic simulation such that the pedestrian movement (or a passenger in a vehicle) can be simulated over a larger region such as a city. Several approaches for simulating vehicular traffic movement have been proposed in the past. Of all the methods developed so far, microscopic traffic simulation using the cellular automata proposed in [NS92] is used widely. This model allows the simulation of a simple one-lane traffic. Several extensions have then been made to this model (e.g., multi-lane traffic simulation as seen in the real world). However, these models do not consider public transportation and heterogeneous traffic such as trucks, bicycles, cars, etc. Therefore, an environment for simulating multi-modal traffic on a microscopic scale was proposed and built within our group [MB06]. A simulator to simulate and visualize the traffic was also built with the option of later parallelization [WK06] (see Figure 4.11). The simulator is also connected to the timetables of public transportation. Therefore, the movement of a pedestrian or a passenger who uses any of the transportation modes can be simulated. The simulator tool also offers an interface to connect to the pedestrian simulation framework. In order to perform a full-fledged pedestrian and traffic simulation, the pedestrian simulation framework and microscopic traffic simulator must be integrated. The interface between the traffic and the pedestrian simulation involves transfer of pedestrians or passengers. The passenger, once arrives at a building through some transportation mode, carries an `ID` along with the profile and a list of tasks to execute. The current time of arrival is also noted. Based on the profile and the list of tasks to execute, the passenger is included with the existing pedestrians in the building and is simulated. Once the passenger completes his task, he is transferred back to the traffic simulator along with the `ID` and the time of task completion.



```
fps:40   view position: 3455464, 5429576, 348
Number of rendered objects: 206
```

Figure 4.11: A snapshot of the microscopic traffic simulator.

4.4 Summary

In this chapter, a generic flexible framework for performing a pedestrian simulation was presented. The framework models the pedestrian behavior (paths and destinations) as queuing systems and embeds the queuing system into the geometry of the reference model. The framework also offers an option to define several pedestrian profiles and simulation parameters such that a generic pedestrian scenario can be modeled and simulated within this framework.

Different commercial pedestrian modeling environments are also available to model and simulate pedestrian behavior. For instance, the commercial software Simwalk, developed by Savannah Simulations, offers an environment to simulate pedestrian behavior, evacuation scenarios, urban planning, etc. The Shopsim, also from Savannah Simulations offers an environment for simulating a shopping complex layout by defining pedestrian data and interchanging the locations of the shops within the layout. All such environments were not found to be suitable for embedding the simulation into the geometry context and providing an option for an intelligent pedestrian navigation. Therefore, the pedestrian simulation framework shown here was built from the scratch.

Different applications such as evacuation planning, building plan optimization, visit estimation, coupling with traditional traffic simulation, etc., can be performed using this framework. In the reference scenario, we choose to use this pedestrian simulation framework to build an intelligent pedestrian navigation system by optimally scheduling the list of tasks to be executed by a new visitor. In the next chapter, different methods to optimally plan and schedule a pedestrian visit that uses the data from the pedestrian simulation is discussed.

Chapter 5

Optimized Pedestrian Task Scheduling and Routing

We should forget about small efficiencies,
say about 97% of the time:
premature optimization is the root of all evil.
– **Donald E. Knuth**

So far, the reference scenario has been modeled and simulated and an overview of possible congestions and waiting times across the scenario has been collected. Now, it is possible to estimate the time it takes to move from the current location to a specific destination, the waiting time at that destination and the service time of the task to be executed. In our reference scenario, the pedestrian arrives with the intention of executing n different tasks as quickly as possible. In this chapter, we study the possibility of using the simulation data obtained from chapter 4 to schedule and route the pedestrian such that the tasks are executed in the shortest possible time (or within the given time-frame).

Scheduling and routing problems are often seen in transportation networks and logistics[1]: routing of shipments (packet pick-up and delivery), service routing (providing on-site technical services), passenger routing. Due to increasing availability of GPS and communication devices, dynamic routing has received a lot of attention. Whatever may be the situation, the goal of routing and schedul-

[1]Logistics is the technique of managing the flow of products, services, etc., from source to destination through the supply chain network. Supply chain management is the process of controlling such a supply chain. To quote an example of efficient supply chain management: The Tiffin Carriers in Mumbai, India (see `http://www.mydabbawala.com/`) are known to deliver and pick up lunch packets around the city with an efficiency of 99.9999% (1 error in 16,000,000 transactions) and achieve a six-sigma [HS00] performance virtually.

ing algorithms will be to minimize the cost and maximize the throughput (least traveling distance, least waiting times, maximum amount of task executions,...). However, when planning a schedule, parameters such as capacity of the system, fixed timings, pre-specified schedules, etc., must be taken into account.

Automated scheduling process is used in wide range of applications such as production scheduling: concerns the scheduling of jobs and control of their flow through a production process, space allocation: optimizing the allocation of physical space or capacities, timetable scheduling: planning and scheduling of airlines, events, university lectures, etc., personnel scheduling: scheduling of man power to optimally complete the set of tasks, and so on.

This chapter is structured as follows. The next section gives an overview of existing scheduling and routing methods. Then, the components of pedestrian task scheduling in the reference scenario are presented and various combinatorial optimization methods and heuristics are used to schedule the list of tasks, as well as route the pedestrian between the tasks. An analysis is then made on the performance of different methods. Based on the analysis, a decision on is made on choosing the appropriate optimization method that can be used to schedule the pedestrian tasks in the reference scenario. Finally, an outlook is made into other possible scenarios (apart from the reference scenario), where the scheduling methods can be used.

5.1 Task Scheduling and Routing

The problem of scheduling and routing involves planning a timetable wherein each process occupies some resource for a certain period of time. The scheduling problem or otherwise referred to as a job-shop scheduling, in its usual form, consists of m machines $M_j|j = 1, 2, \ldots, m$ and n jobs $J_i|i = 1, 2, \ldots, n$ such that each job is alloted one or more time intervals to one or more machines. A job J_i consist of a number n_i of operations $O_{i1}, O_{i2}, \ldots, O_{in}$ and each operation O_{ij} is associated with a processing time. There are several classes of scheduling problem and they are specified in terms of a three-field classification $\alpha|\beta|\gamma$, where α specifies the machine environment, β specifies the job characteristics and γ denotes the optimality criteria [GG75].

The job characteristics are defined with the set β consisting of the following elements.

- **Preemption**: It indicates whether the job may be interrupted and resumed at a later point of time.

- **Precedence Relation**: Sometimes there may be a specific order in which the jobs can be executed and if such an order exists, it is defined by a precedence relation.
- **Release Date**: A tentative time when the job shall be completed is defined. The release dates may or may not be defined.
- **Processing time**: Certain jobs may have restrictions with the processing times or restriction with the number of operations each job contains.
- **Deadline**: If there exists a deadline to finish the job, the job may not be finished later than the specified deadline.

The following criteria must be considered when optimizing a schedule.

- **Throughput**: The schedule must be prepared such that maximum possible jobs are completed within the least amount of time. Most methods improve the throughput by increasing the efficiency of the job execution. The resources are utilized to the maximum possible extent such that the throughput is higher.
- **Fairness**: It is important that each resource is fairly allocated. Situations, where a particular job blocks a resource forever such that other processes continuously wait for the resources must be avoided.
- **Deadlines**: If the jobs must be completed within the allocated time, the schedule is planned such that the deadlines are met.

Apart from the above-mentioned criteria, several other constraints may be specified and these constraints must be considered when scheduling the jobs. For example, the schedule must be planned such that the make span is minimized or the maximum tardiness is minimized. More information on scheduling algorithms can be found in [Bru04].

In our reference scenario, an optimal schedule must be planned such that the pedestrian executes all his tasks within the allotted amount of time. The scheduling must take into consideration the congestions and waiting times that occur in a system. Execution of each task involves a transition time (the time required to physically reach the destination). Therefore, the pedestrians have to be routed to each task. So with each task, there is also a routing operation performed.

Dynamic Shortest Path Problem

A routing operation in this case refers to the transition between two different destinations. Therefore, routing involves movement of the pedestrian from one destination along the path to another destination. The objective of routing is to determine the optimal path between the two destinations. The term "optimal" can denote the shortest-path, fastest-path, path with the least amount of congestions, path based on some constraints (no stairs when using a wheel chair) or path with the least cost. Of all the listed types of paths, the shortest-path is the only path that takes a static value (distance is always the same). The other types of paths are often dynamic, i.e. the value changes over time. A majority of the published research on shortest path algorithms has dealt with static networks that have a fixed topology and fixed costs. This was due to the enormous amount of computing power needed to perform a path-search in the early days. [EJ90] in 1990 for example reports several hours to calculate an all-to-all shortest path for just 250 nodes of static network and several days for a 16,000 nodes of a large-scale network. Therefore the early attempts made by [CH66, Dre69] to compute the shortest path for dynamic networks were not successful.

One way of dealing with dynamic networks is splitting continuous time into discrete time intervals with fixed travel costs, as noted by [Cha97]. Building on the research from path finding algorithms in static networks, [Cha97] suggests that a time-space expanded graph network of the discrete dynamic data can be used to calculate the dynamic optimal path between two points. Hence, depending on how time is considered, dynamic shortest path problems can be subdivided into two types, namely discrete and continuous. In the discrete case, if using a 1-minute time interval for a period of 8 hours, the time-expanded graph will contain $m \times 480$ time discretizations, where m is the number of edges in the graph.

[Cha98] lists the following types of dynamic shortest path problems depending on

- fastest vs. minimum cost (or shortest) path problems
- discrete vs. continuous time representation
- FIFO vs. non-FIFO network (a pedestrian departing later may arrive earlier than other pedestrians)
- waiting vs. no waiting at nodes
- one-to-all for a given departure time or all departure times, and all-to-one for all departure times

- integer vs. real valued link travel costs

[PPR$^+$03] investigates the possibility of solving a dynamic and stochastic shortest path problem by modeling link travel times as a continuous-time stochastic process. The aim is to estimate travel time for a particular path over a given time period. A survey on dynamic shortest path problems can be found in [Hud00].

Scheduling and Optimization Methods

Apart from performing a routing operation (a path-search operation), the tasks must be scheduled in a specific sequence such that the time taken to complete all the tasks, including the transition between the tasks, is minimized. A typical scheduling problem (or a job-shop scheduling problem), which involves scheduling i number of jobs in j number of machines, where each job may have multiple operations to be performed on different machines, is one of the most difficult classical scheduling problem. In most cases, a shop scheduling problem is classified as $\mathcal{NP}$-complete and very simple special cases of shop scheduling problems are already classified as $\mathcal{NP}$-hard. For instance, a 10×10 job scheduling problem (10 jobs on 10 different machines), has been an open problem since 1963 [MT63] and was solved 25 years later [CP89] using the branch and bound method. Shop scheduling problems are classified into several types such as a single machine scheduling problem, parallel machine scheduling problem, special cases such as open shop, flow shop or mixed shop problems, etc. A shop scheduling problem is represented using a disjunctive graph model. A disjunctive graph $G = (V, C, D)$ where V is the set of nodes representing the operations of all jobs, C is the set of directed conjunctive arcs (reflect the precedence relations between the operations) and D is the set of undirected disjunctive arcs (for operations that must be performed on a same machine). Different combinatorial optimization methods and heuristics are used to solve a job-shop scheduling problem. Mixed integer linear programming, branch and bound method, approximation methods such as a local search, bottleneck-based heuristics, etc., are used to optimize the scheduling problem. Due to the complexity of the scheduling problem, it is often sufficient to find a feasible solution such that the schedule is complete within a given time.

Much research has been done in optimizing such scheduling problems. For instance, [SS99] proposes an optimal stochastic scheduling method to schedule tasks in multi-class parallel queues. [Hoo06] combines mixed integer linear programming together with constraint programming to minimize the maximum tardiness in planning and scheduling problems through a Benders scheme [Ben62].

Different heuristics are also used for dynamic job scheduling problems. For example, [MRea02], uses genetic algorithms and evolutionary methods to optimize dynamic scheduling problems. Also the use of tabu search method for dynamic job shop scheduling problem can be seen in [WBHW03].

Traveling Salesman Problem

In the reference scenario, the pedestrian must visit n of destinations and complete the task in each destination in the shortest possible time. The problem resembles a traveling salesman problem. The major difference between a traveling salesman problem and the problem in the reference scenario is that the elementary traveling salesman problem is static. That is, the distance between any two given cities is constant and it is assumed that the time taken to travel between two cities is also constant regardless of the time of travel. Also, the traveling salesman problem does not consider the dynamic time it takes for the salesman to execute a task at each city. Research has been made in solving dynamic time dependent traveling salesman problem. Before, we discuss the solution to the dynamic problem, let us have a short look into solving a regular traveling salesman problem.

In a traveling salesman problem, a salesman visits n of cities spread across a 2-dimensional space. The objective of the salesman is to minimize the total traveling distance. That is, determine the visit sequence such that the distance covered is minimum. The solution for a traveling salesman problem is found to be $\mathcal{NP}$-hard and the decision problem (if there exists a route r' that is cheaper than the existing route r) is found to be $\mathcal{NP}$-complete. Several heuristic approaches have been proposed to solve a traveling salesman problem. Determination of an exact solution however is difficult and requires a lot of computing power. For example, in March 2005, the traveling salesman problem of visiting all 33,810 points in a circuit board was solved using CONCORDE:[2] a tour of length 66,048,945 units was found and it was shown that no shorter tour exists. The computation took approximately 15.7 CPU years. For more information on computational methods to solve the traveling salesman problem, refer to [Rei94].

In a dynamic traveling salesman problem however, the values change over time. Different proposals are made to optimize a dynamic time dependent traveling salesman problem. For example, [BBO+01] optimizes the time dependent traveling salesman problem using Monte Carlo methods. The paper attempts to optimize a "real" traveling salesman problem by considering the rush hours or traffic jams during traveling. The simulated annealing method is used to optimize the route of the traveling salesman problem and a time-distance matrix is used to represent the changes in the time it takes to travel. The paper uses specific

[2]`http://www.tsp.gatech.edu/concorde/index.html`

traffic information together with the `TSPLIB95`[3] to locally optimize the traveling salesman problem. Similarly, [SCM06] compares several formulations of a production scheduling problem with sequence-dependent and time-dependent setup times on a single machine. The problem is viewed as a time dependent traveling salesman problem, where the travel time between two nodes is a function of the departure time from the first node. Linear programming with lower bounds, as well as branch-and-cut algorithms are used to solve instances with up to 20 jobs within reasonable amount of time.

5.2 Pedestrian Visit Planning

The pedestrian simulation of the reference scenario in chapter 3 so far gives us an overview of the congestion status of each path and destination at different simulation time stamps. Now, in the reference scenario, a new pedestrian arrives in the scenario and visits a sequence of n destinations before leaving the scenario. The entire visit involves $n+1$ transitions to each destination (n transitions from the starting point to each successive destination and $(n+1)^{\text{th}}$ transition back to the exit) as well as an activity (execution of the task T) at the destination. The overview obtained from the simulation enables us to estimate the time it takes for a pedestrian to move from the current location to a specific destination at a given time t and also estimate the waiting time at the destination at time $t+t(P_i)$ (where $t(P_i)$ denotes the time it takes to walk along the i^{th} path)[4]. The waiting time in a queue and the actual service time at the service counter i are denoted by $t(WQ_i)$ and $t(ST_i)$. The time necessary to execute a task is therefore denoted as $t(T_i) = t(P_i + WQ_i + ST_i)$. A task can denote any arbitrary activity that a pedestrian would perform in a scenario (shopping, working, relaxing). Using the statistical data obtained from the pedestrian simulation, we investigate the possibilities to schedule and navigate the tasks to be executed by a pedestrian.

In the reference scenario, a pedestrian arrives with an intention of executing n tasks optimally. For experimental purposes, we consider that the visit must be completed in the shortest possible time such that the schedule $S = \min\{\sum_{i=1}^{n} t(T_i)\}$ or within the given deadline such that $t_S \leq t_{\max}$ (for example, before closing time). The simulation data obtained from the previous chapter gives us the time it takes to walk across each path and the waiting time at each destination during different times of the day, which makes the data dynamic.

[3]`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[4]The index i is used to represent the i^{th} task T_i out of n tasks. Since each task involves the transition to the task, waiting before the task and service of the task, the index i shall be used to also represent the path, queue and service station of the respective task.

Therefore, a task T_i executed at times t_1 and t_2 require different amount of time. Let list of tasks[5] be $\mathcal{T}\{T_i|i = 1, 2, \ldots, n\}$. The visitor starts his visit at *source*, visits n destinations and traverses to the *sink*. Therefore, the set of paths, denoted as $\mathcal{P}\{P_i|i = 1, 2, \ldots, n, n + 1\}$, consists of n paths to each destination and the last path to the *sink* (see Figure 5.1).

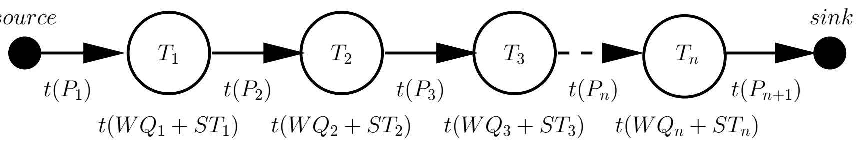


Figure 5.1: Set of tasks to be executed by a pedestrian.

The pedestrian visit planning problem consists of two components, namely scheduling the tasks and path finding between each task. In the reference scenario considered here, the pedestrian would wish to obtain a feasible schedule either in advance, e.g., through a web-based interface, or dynamically via some hand-held devices. In either case, it is important compute a feasible itinerary within a short span of time. In this section, the various techniques that were used to solve these two components are presented.

5.2.1 Pedestrian Routing: Fastest Path Problem

The task scheduling problem involves both sequencing of the tasks and movement of the pedestrian between the tasks. To reach a destination node, the pedestrian must find his way from his current position to the destination node. A path must be found such that the pedestrian is able to walk easily without congestions and reach his destination in the shortest possible time. The path-search algorithm makes use of the congestion data obtained from the pedestrian simulation. Even though the edge value is dynamic, the pedestrian simulation records the edge data as discrete and not as continuous values. A generalized static version of the Dijkstra's algorithm can be used to calculate the fastest path between two points in a discrete dynamic graph [Cha98, Dea99]. This algorithm works in the same way as the static algorithm and has a complexity of $\mathcal{O}(m + n \cdot \log(n))$. However, in reality, the computational needs and the running time for a discrete dynamic shortest-path algorithm is high. This is because of the construction and use of a time-expanded network (the edges of the graph should contain all discrete data). Also in a one-to-all shortest path problem, such data network is

[5]To differentiate between a single task and a set of tasks the calligraphic font is used to denote a set of tasks. The same applies to the set of paths ($\mathcal{P}$) and set of sequences ($\mathcal{S}$).

necessary even though the actual number of destinations is much lesser than the actual number of nodes present in the model (30% in our reference model). Also due to the stochastic nature of the simulation, the edge values are an estimate of the path congestions and not exact values determined from physical sensors. Therefore, computational efforts needed to build a time-expanded graph network for determining the fastest path are unnecessary and must be avoided.

Alternatively, we use a much simpler algorithm to approximate the fastest-path without a time-expanded network by carefully manipulating the graph with certain stochastic parameters. The fastest path is determined as follows. Initially, the weights of the edges of the graph are replaced with the time it takes to cross the path (depending on the average walking speed, type of the path and the length of the path). The walking time along the edge e_i is determined by $t(e_i) = d_{e_i}/(v * speed\ factor)$, where d_{e_i} is the length of the edge and v is the static average walking speed ($1.38\ m/s$). The speed factor is determined based on the type of the path as explained in 4.2.3 (for stairs $0.5 \cdot v$, for downslope ramp $1.2 \cdot v$). With the time weighted graph, the shortest-path algorithm now gives us the fastest path between two points.

Now, these edge values must be replaced by the actual time it takes to walk across the path when there is a congestion. This is because the congestions that occur along the path will reduce the total time needed to reach the destination. For this, we use the edge data obtained from the pedestrian simulation. Physical experiments have been made within the reference model and it was found that the maximum amount of time to transit between two farthest points in the graph (longest walk) takes no more than 10 minutes. It was also found that the congestions along the path generally remain consistent or change only gradually. Therefore, the mean value of actual transit times (obtained from the simulation) is computed for the next 10 minutes for each edge of the graph. This process has a complexity of $\mathcal{O}(n)$. The mean value is then compared with the existing static value of the edge and the greater value is retained $\forall\ E$ in $G(V, E)$, $e_i = \max\{t(e_i)_{static}, t(e_i)_{mean}\}$. Now, the static shortest path algorithm shall yield the fastest path from v_1 to v_2.[6]

The manipulation of the graph by stochastic means may not work in all cases. In certain cases, for example before and after a stationary process, rapid changes

[6]Since only the longest walk takes 10 minutes, most other paths will take much lesser time. Moreover, the approach mentioned here is specific for the reference model. A generalized solution will therefore be to first find the shortest path from v_1 to v_2, compute the time (k minutes) it takes to traverse this path, use the value of k to identify the mean value, and recompute the shortest path. In that case, the shortest path is calculated twice, first to find k and second to find the actual shortest path. We do not use the generalized method here since it is faster to use the experimental data rather than to calculate the shortest path twice.

in the congestion are seen along the paths that lead to the destination of a stationary process. These fluctuations will be ignored when computing the stochastic mean of the weights of the graph edges and a pedestrian might end up in a blocked path due to the fluctuations in the congestion. Therefore, the stochastic graph manipulation method is used only after ensuring that there are no rapid changes in the edge weights for the next 10 minutes.

5.2.2 Pedestrian Task Scheduling: Task Sequencing

We now have a generalized path-search method that identifies an optimal path between two nodes. The next stage involves finding a schedule (sequencing the tasks) such that the total time taken to complete the visit is within the allotted time limit. The schedule S involves sequencing of set of tasks $\mathcal{T}$ and we assume that the time taken to complete each task T_i includes the $t(P_i)$, $t(WQ_i)$ and $t(ST_i)$.

5.2.2.1 Brute-Force Search

Theoretically, the only accurate solution will be to use a brute-force search. A brute-force approach identifies the optimal sequence of tasks by determining the time taken by all possible sequences $\mathcal{S}\{S_i | i = 1, 2, \ldots, n!\}$ and identifying the sequence which takes the least time.

A brute-force search is a trivial but a very general problem solving technique, where each possible solution is evaluated to determine if the solution suits the problem statement. For example, to solve the traveling salesman problem, i.e. to determine a route through n cities such that the distance covered is minimum, it is necessary to identify every possible route through n cities to identify the route that covers the shortest distance. The brute-force approach is straightforward and guarantees the correct solution. However, a major drawback of brute-force search is the computation effort, especially when n is big.

The brute-force approach for scheduling the tasks have a computational complexity of $\mathcal{O}(n!)$. Even for small n (e.g., $n = 10$ or $n! = 3,628,800$ comparisons), modern computer requires many hours of computation. However, for very small problems (e.g., $n = 4$ takes just about 0.2 s), brute-force search can be used, especially when a large scenario can be decomposed into smaller domains. Since a pedestrian requests for a schedule on-the-fly, brute-force search is inappropriate for pedestrian task scheduling if n is big.

Due to the stochastic nature of the simulation data, there also is no guarantee that the schedule determined from a brute-force search is indeed the fastest and the most optimal schedule. This is because, the simulation attempts to capture the congestions and waiting time situations in a macroscopic level and a precise behavior modeling of each pedestrian is therefore not possible.

> *It is not possible to predict the occurrence of a slow moving group at a specific time and path that would decrease the walking speed.*

Also, if considering such negligible details, a small change could create an impact to the final schedule. That is, if it is assumed that the pedestrian would start at time t, arrive at the destination at $t+\Delta_1 t$, start the task execution at $t+\Delta_1 t+\Delta_1 t_2$ (where $\Delta_1 t$ represents the transition time and $\Delta_2 t$ represents the waiting time), an unexpected change in the Δ value will then render the schedule invalid and require a new computation.

5.2.2.2 Greedy Heuristic

A greedy heuristic is a very commonly used method for finding feasible solutions of combinatorial optimization problems. In a greedy heuristic, the algorithm follows the problem solving meta-heuristic of making the locally optimum choice at each stage with a hope of finding the global optimum. For instance, applying the greedy strategy to the TSP yields the following algorithm. "At each stage, visit the unvisited city nearest to the current city". Similarly, a greedy heuristic can be used for scheduling the tasks for a pedestrian visit.. There are three ways to perform a greedy algorithm for generating an itinerary and they are listed as follows.

***Greedy path-search*:**
In the first approach, the algorithm always looks for the task that can be reached fastest. The generic path-search algorithm is used to measure the time it takes to walk between the starting point and all destinations. The destination that can be reached fastest is chosen. Let T_j be a task j such that $T_j \in \mathcal{T}$. The next task $T_j = \min\{t(P_i)\} \; \forall i$. The greedy path-search algorithm has a complexity of $\mathcal{O}(n)$ for the first time and $\mathcal{O}(n-1), \mathcal{O}(n-2), \ldots$ and so on for consecutive searches. The major disadvantage with this method is that there exists no control in choosing the destinations. Therefore, the pedestrian might choose a destination that is most crowded currently.

***Greedy Sequencing*:**
In contrast to the greedy path-search method, the next approach chooses the next task that can be executed the fastest. The algorithm identifies the current waiting time at each destination, and chooses the task, which requires the least waiting time. Let T_j be a task j such that $T_j \in \mathcal{T}$. The next task $T_j = \min\{t(WQ_i)\} \; \forall i$.

Note that the service time ST_i is not considered for scheduling since service time of a task T_j is generally consistent at any point of time. On the other hand, when including the service time to compute the schedule, the tasks that take longer to execute will be postponed. The computational complexity is same as the greedy path-search method. In a greedy sequencing algorithm, the location of the destination (distance from the current position) is ignored. Therefore, a pedestrian might have to walk long distances between each task.

***Greedy Routing and Scheduling*:**
This approach essentially combines both greedy path-search and greedy sequencing methods to choose the appropriate destination by counting the length of the path as well as the waiting time at the destination. In this method, the sum of waiting time and walking time $t(P_i + WQ_i)$ is calculated for each destination and the destination with the least walking and waiting time is chosen. Let T_j be a task j such that $T_j \in \mathcal{T}$. The next task $T_j = \min\{t(P_i + WQ_i)\}\ \forall i$.

It is interesting to note that for different simulation data, each of the above mentioned greedy methods perform differently. Different test scenarios were simulated and greedy search heuristic were performed on the simulation data. The results were compared with the sequence obtained from the brute-force search. It was found that, the greedy sequencing approach often produces a sequence whose time of completion is longer than the original sequence generated without any optimization. This was because the greedy sequencing chooses the destination with the least waiting time and it does not differentiate marginal differences in the waiting time (e.g., if a and b are two destinations and even if $0 < t(WQ_a) - t(WQ_b) < x$ for very small values of x such as 1 minute, the destination a is chosen immaterial of its location). Therefore, the pedestrian ends up spending a lot of time in transition. On the other hand, the greedy path-search approach often – if not always – produced the sequence same as that of the brute-force search, when the waiting times are low. This is because the waiting times were generally the same at different points of time and due to the architectural structure, the fastest reachable neighbor often results in the fastest path through all the destinations. Therefore, for the reference model, where the waiting times are generally low (or constant) throughout the given time, greedy path-search approach may be used. Of all the greedy search approaches, the greedy routing and scheduling method yielded the most stable results. More analysis will be presented in 5.3 and numerical examples will be presented in chapter 6. Since the future states of the destinations are not taken into account and since the data is dynamic and not static, the greedy approach does not always guarantee us the optimal solution. For example, the time of a task $t(T_j)$ may be at its minimum at time t. But the task may be postponed since $t(T_j)$ is not the minimum among all other tasks in $\mathcal{T}$.

5.2.2.3 Stochastic Optimization

The greedy search mentioned above is a local search technique. The drawback of a greedy search can be avoided by iteratively determining a schedule and checking if the new schedule is faster. However, a greedy search always chooses the fastest task and any number of iterations will result in the same schedule. Therefore, a probabilistic approach is made to determine if a faster schedule exists. The method of simulated annealing [KGV83] is used to determine the fastest schedule.

Simulated annealing originates from theoretical physics, where Monte-Carlo methods are employed to simulate phenomena in statistical mechanics. The term annealing involves a technique of heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The process of heating causes the atoms to become loose from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy. The slow cooling gives the atom the possibility of finding configurations with lower internal energy than the initial one. Each time the annealing is performed, the states are chosen randomly and this value depends upon the previous temperature value.

In a simulated annealing method, several iterations are made to determine the optimal solution. At each step, the simulated annealing method considers some neighbor state s' of the current state s and decides on choosing s or s' as the solution base on probabilistic methods. The probabilities are chosen such that the system automatically finally tends to move to a lower state of energy. Generally, the process is iterated until the solution seems to be optimized (with a termination condition).

Simulated annealing method can be used to determine the optimal schedule for the pedestrian visit. Initially, the pedestrian arrives with a set of n tasks to execute. The total time taken to execute the tasks in the order of listing is computed and is set as the initial solution. The algorithm tries to identify a new sequence S' at random and S' is accepted as optimal if it is faster than the original sequence. This process is iterated as long as an optimized solution is obtained. The algorithm 8 shows the implementation of using simulated annealing method to optimize the schedule

The algorithm 8 is a general approach of simulated annealing. In the algorithm, the sequence S' is chosen completely at random. That is, two tasks T_1 and T_2 are identified at random such that $T_1 \prec T_2$. S' is derived by removing T_2 from the sequence list and inserting it immediately after T_1. Alternatively, it is possible to minimize the number of iterations by improving the efficiency of the random function. This can be done as follows. The pedestrian profile used for

Algorithm 8 Simulated Annealing method for Pedestrian Scheduling.

```
 1: determine initial schedule S, set initial temperature $\vartheta > 0$, set decrement
    factor $r$
 2: while stopping criteria not reached do
 3:    for $r$ times do
 4:       perform some random change of sequence and obtain sequence $S'$
 5:       determine $\Delta = t(S') - t(S)$
 6:       generate random number $x$, $0 \leq x \leq 1$
 7:       if $\Delta < 0 \vee x < \exp(-\Delta/\vartheta)$ then
 8:          set $S = S'$
 9:       end if
10:       update $\vartheta$ and $r$
11:    end for
12: end while
13: decide on $S$
```

the pedestrian simulation consists of the distribution of the profile types (x% of Profile 1, y% of Profile 2, etc.), as well as the probability distribution of the time-based destination visit of each profile (for destination type 1, the visit probability between 9 and 10 am is z%). In the schedule S, each task is associated with a destination type. Therefore, it is possible to extract the probability distribution of the congestions that occur during the given time period. Now, the random function identifies one task at random, identifies the time when the destination shall expect the least crowd, identifies another task in the schedule which lies at this time period, and swap with the new task. The algorithm 9 shows the implementation of the random function used to determine the new schedule S'.

Algorithm 9 Procedure to determine S' in a Simulated Annealing method.

```
1: while $S'$ not identified do
2:    randomly identify $T_j \in S$
3:    identify from distribution time $t$ when $T_j$ can expect least queue
4:    identify if there is a task $T_k \in S$ at time $t$
5:    swap($T_j, T_k$)
6: end while
```

During the initial iterations, S' is determined often. As the number of iterations increase, this algorithm may require more and more repetitions to compute S'. This is because the tasks are already swapped to suit a better execution time. Therefore, a breaking condition is set and as long as this condition is reached, S' is determined at with probabilistic methods using the profile data. After that S' is determined at random as explained in algorithm 8.

Compared to greedy search heuristic, simulated annealing method produced a very stable task sequence. Although the solution obtained from simulated annealing was never the same as the optimal solution obtained from brute-force search, the results were close to optimal schedule and they were consistent in all cases. Additional analysis will be discussed in 5.3 and sample results are presented in chapter 6.

5.2.3 Constraints and Preconditions

In typical shop scheduling problems, the job characteristics β are considered for scheduling. Similarly, the pedestrians arriving at a scenario may have different priorities and preferences (referred to as constraints) such as precedence relation for tasks $T_i \prec T_j$, path preferences (avoid stairs or escalators), queue priorities (preemptive service, queue balking and reneging), and a specific time of execution (execute task T_i at time δ). Such constraints are defined within the pedestrian profiles and are incorporated when scheduling the tasks.

Apart from pedestrian constraints, a careful analysis of the pedestrian simulation data can give us specific situations such as times when certain paths are heavily congested or times when certain destination nodes are free of any activities. Such preconditions can be used together with the scheduling methods discussed above. In general cases, the congestions in a path or destination changes continuously and gradually. Rapid fluctuations are seldom seen along these paths. However, due to stationary processes, such fluctuations along the paths or nodes are possible. A stationary process is a stochastic process whose probability distribution at a fixed time or position is the same for all times or positions. As a result, parameters such as the mean and variance also do not change over time or position. However, the time before and after a stationary process experiences a rapid fluctuation near the geographical location of the process. A classical example of a stationary process is a lecture hall. In case of a lecture hall, the probability distribution between the between the start and end of the process (a lecture) remains constant. Therefore, pedestrians arrive shortly before the start of the process and the inter-arrival times reduces rapidly before the process. This process causes congestions along the paths in the vicinity of the destination (lecture hall). Similar fluctuation is seen once the process is complete. From the simulation data, it is possible to identify stationary processes and thereby identify the congested paths at a given point of time.

Similarly, long waiting times that occur in the scenario can be detected from the simulation data. These observations can be used to ensure that the task is scheduled such that lengthy waiting queues are avoided. In one of the test scenarios that was simulated, the following parameters were considered.

- About 3000 customers visit the building, each with a specific list of tasks to execute (derived from the customer profile).
- Utmost 1000 customers are present in the building at any given point of time.
- A restaurant, with a capacity for 40 people is chosen for analysis, and simulations are run to study its occupation during different times of the day.
- The probability of the time of visit is determined from the customer profile. In general, the probability of visit is high during noon.
- It is assumed that the entities wait in the queue as long as they are served. Also, it does not deter further additions (queue reneging and balking are deactivated).
- The simulation is run for a period of 8 hours (9am to 5pm).

The plot in Figure 5.2 shows the number of customers (entities in the queue and the entities being served) during different times of the day. From the plot, it can be seen that the restaurant is overloaded shortly after noon and takes more than an hour to fall below the threshold line (less than the maximum capacity). Such information can be used as a precondition when planning a schedule. For instance, if the schedule includes a visit to a restaurant, the visit can be fixed shortly before noon.

A *tabu* list that contains preconditions similar to that listed above as well as a list of constraints to from the pedestrian profile is generated. The tabu list is then used by the different scheduling methods to schedule the task. Each time a task (or the sequence) to be executed is identified, the tabu list is used to accept or reject the decision made by the scheduling algorithms.

5.2.4 Nearest Neighbor Search

The nearest neighbor search attempts to find the next closest destination (in terms of distance) to visit. Nearest neighbor search is similar to greedy path-search method excepting that the nearest neighbor search is a static algorithm. In practical situations, it can be observed that the customer tends to stay in the region and finish the tasks located within the region before moving to the next region. The classical example is a theme park, where the attractions are grouped

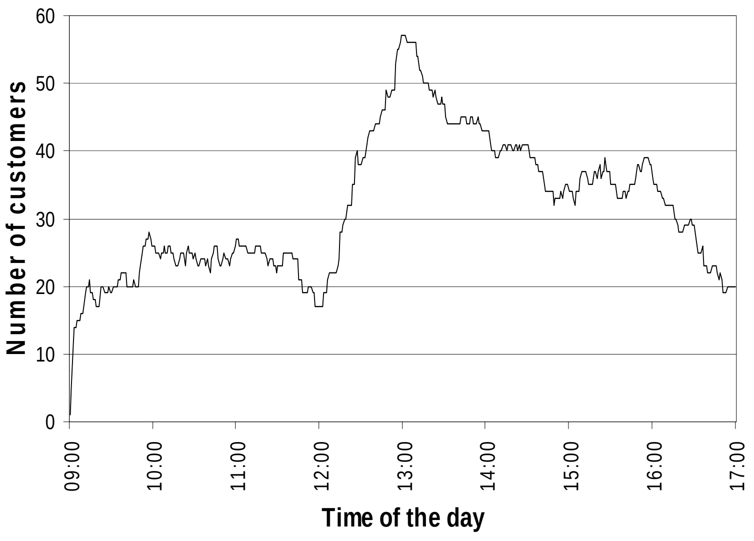


Figure 5.2: Number of visitors for a sample restaurant (capacity=40) recorded during the simulation.

and sorted according to the theme and a visitor normally makes a tour from the first area to the next closest and so on. The nearest neighbor search makes an attempt to localize the search for optimal path by splitting the graph into several subgraphs (regions), optimizing the plan for each subgraph and finding the shortest connection between each subgraph. The scheduling methods explained earlier are used to schedule the tasks optimally within the subgraph.

Due to the geometric background of the scenario, the nodes are spread over a 3D space. These nodes are decomposed into smaller regions and each region is interlinked with a specific sequence. Some of the decomposition methods include space filling curves, graph partitioning methods (octrees, Kd-Trees), Voronoi diagram, Delaunay decompositions, convex hull, etc. For the position identification of the pedestrian in the graph, octrees were used to partition the graph nodes and to identify the position of the customer. By varying the level of hierarchy, octrees can be used to decompose the graph into smaller domains such that these domains are optimized locally. For more information on graph partitioning techniques, refer to [AL97, BG95, CGT96].

5.3 Analysis of Scheduling and Routing Methods

Several hypothetical scenarios were simulated and data regarding the path congestions and queue waiting times were collected throughout the simulation period. In the reference scenario, the pedestrian arrives with an intention to execute n tasks. Initially, we list out the tasks and compute the time it takes to complete the set of tasks. For this purpose, a random list of tasks were generated and with the congestion data, the time it takes to complete all the tasks in the order of the random generator were estimated. Let the task sequence from the random generator be denoted as S_{init}. Now, we optimize S_{init} using the above listed optimization methods such that $t(S)$ is minimum.

In one of the hypothetical scenarios simulated, the scenario consisted of a total of 5000 visitors with a maximum of 500 visitors present in the reference model at any given time. Each visitor executed an average of 10 tasks and the average service times were in the range of 5-20 minutes per task. The scenario was simulated for a period of 8 hours (9am to 5pm) and the simulation took 117 seconds to complete on a Pentium 4 processor with 3GHz speed and 1 GB main memory. The data obtained from the simulation was found to contain generally low waiting times due to the availability of several destinations and short service times.

For the new pedestrian entering the scenario, about 10 destinations, where a task can be executed, were generated at random. Different problem sizes were chosen (the first 4,6,8 and 10 tasks) and the scheduling methods were implemented for each problem set. Let the task sequences obtained from the scheduling methods be denoted as S_{brute} (sequence obtained due to brute-force search), S_{gpath} (sequence obtained due to greedy path-search method), S_{gwait} (sequence obtained due to greedy sequencing method), S_{gtask} (sequence obtained due to greedy scheduling and routing method), and S_{stoch} (sequence obtained due to simulated annealing method). Let the time taken to execute the tasks be denoted as $t(S_{\text{init}})$, $t(S_{\text{brute}})$, $t(S_{\text{gpath}})$, $t(S_{\text{gwait}})$, $t(S_{\text{gtask}})$, and $t(S_{\text{stoch}})$ respectively. Figure 5.3 shows a comparison of the results (the time taken to execute the tasks).

It can be seen that in certain cases $t(S_{\text{gwait}}) > t(S_{\text{init}})$. This is because the greedy sequencing approach did not differentiate with marginal differences in waiting times. Therefore, greedy sequencing method is found to be inappropriate in pedestrian task scheduling. The results from the greedy path-search method was often close to the best result $t(S_{\text{gpath}}) = t(S_{\text{brute}})$. However, there existed inconsistencies in several other experiments performed $t(S_{\text{gpath}}) > t(S_{\text{init}})$. Of all the greedy search methods, the method that combined the path-search and waiting time S_{gtask}, performed most consistently. But, for large problem sizes,

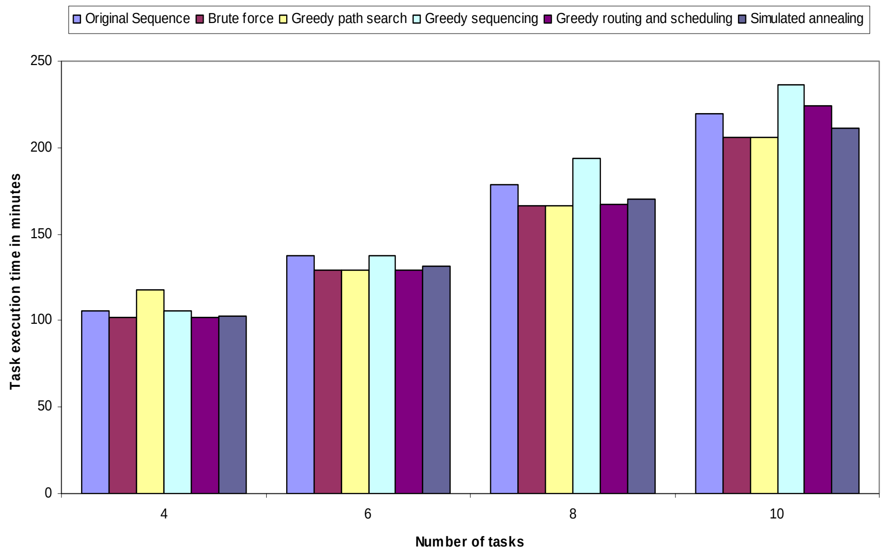


Figure 5.3: A comparison of the time taken to execute varying number of tasks $\{4, 6, 8 \text{ and } 10\}$ with and without the optimization methods to schedule the task. The bar graph shows a comparison of the quality of the output sequence obtained due to the use of the scheduling methods.

inconsistencies were noticed. The simulated annealing method produced very consistent results for different scenarios and problem sizes. The results were close to the optimal solution but never the same as the optimal solution $t(S_{\text{init}}) > t(S_{\text{stoch}}) > t(S_{\text{brute}})$.

The computing times were measured for each method and different problem sizes. The greedy search had the least complexity and therefore took the least amount of time. The computing time for all greedy search methods were linear in the order $\text{speed}(S_{\text{gpath}}) < \text{speed}(S_{\text{gwait}}) < \text{speed}(S_{\text{gtask}})$ with respect to the problem size (see Figure 5.4). Due the search mechanism, the greedy approach takes the least time.

The brute-force search always yields the optimal solution but there is a factorial increase in the running time (see in Figure 5.5). For a problem size of $n = 10$, the brute-force search lasted about 18.5 hours in comparison with just 0.2 seconds for $n = 4$. The computing time of the simulated annealing method depends on the number of iterations performed. In this example, between 100 and 230 iterations were performed for different problem sizes. The computing time for simulated annealing method increased linearly.

From the analysis, it was found that the greedy search approach is often unstable whereas the stochastic optimization using simulated annealing method

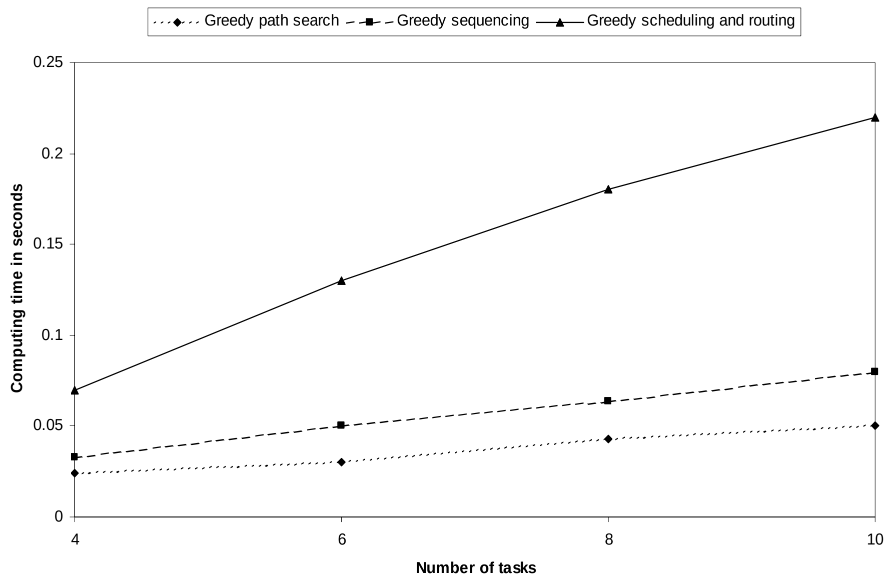


Figure 5.4: Comparison of the computing speed of the greedy methods used to schedule the tasks.

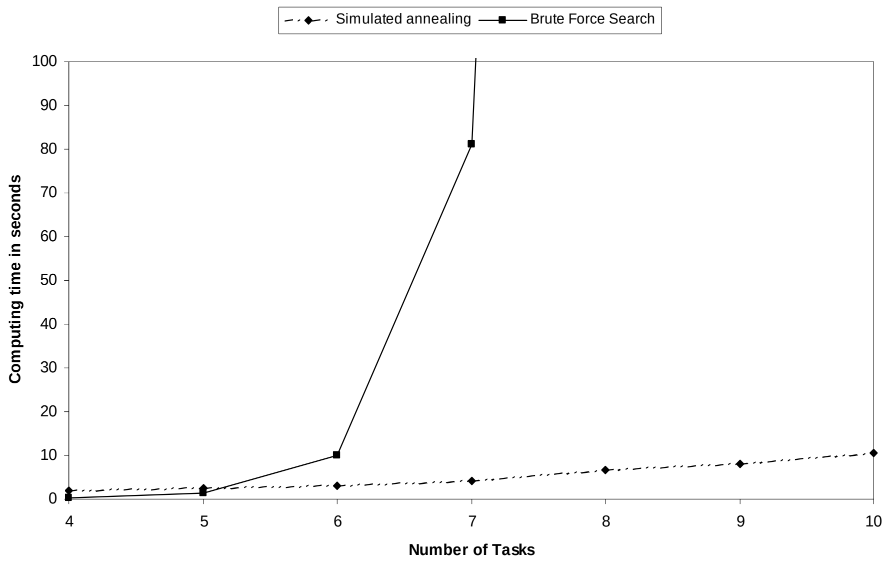


Figure 5.5: Computing speed comparison of brute force and simulated annealing.

produced stable results in reasonable amount of time (see Figure 5.6). Compared

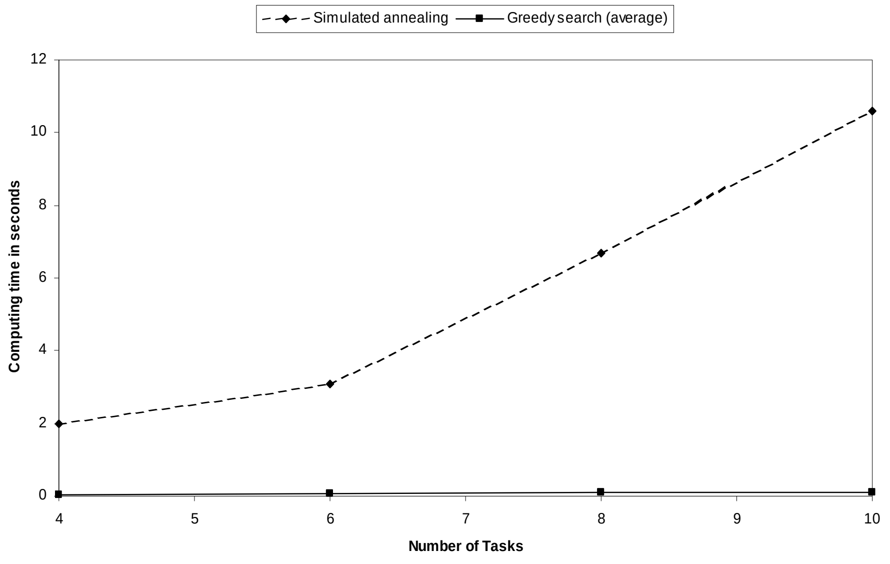


Figure 5.6: Computing speed comparison of (average) greedy search and simulated annealing. Even though, the greedy search is much faster than the simulated annealing method, considering the quality of the output, it can be seen that the simulated annealing computes the schedule still in a reasonable amount of time.

to the simulated annealing method, the brute-force search requires a very long time to compute. However, when the number of tasks $n \leq 5$, the schedule is determined faster using the brute-force search is computed faster. Reasonable amount of time is taken by the brute-force search to determine the optimal schedule for $n = 6$. Considering the superiority of the simulated annealing method over the greedy search method and considering the computing times of the brute-force search and simulated annealing method, it has been decided to use brute-force search for $n \leq 6$ and simulated annealing method for $n > 6$ tasks to schedule the list of tasks for the pedestrian in the reference scenario. Numerical examples for task scheduling in the reference scenario will be presented in chapter 6.

Finally, when providing itineraries for several new pedestrians, these pedestrians must in turn be inserted in the existing congestions and waiting queues such that the simulation data is constantly updated. Any new schedule computed would therefore consider the congestions caused new pedestrians who have entered the system. However, re-simulating the entire scenario to update a single pedestrian data requires lot of computing time, especially when repeating the simulation for several pedestrians. This cumbersome procedure can be avoided by directly inserting the new schedule in the available simulation database. That is, use the existing simulation data as the initial condition and simulate the task execution of a single pedestrian. The output data will then automatically contain the movement of the new pedestrian.

5.4 Summary

So far, various optimization methods have been proposed to schedule a pedestrian visit. It was found that the visit planning involves both optimal task sequencing and optimal path-search method. For finding the optimal path, a generic Dijkstra's algorithm shall be used (static version) with a time weighted graph. An average value of the weights is calculated if path congestions are consistent (or change only gradually) over time. If not, the time-expanded graph network is used. For sequencing the list of tasks optimally, a brute-force search shall be used if $n \leq 6$ and a stochastic optimization using simulated annealing shall be used if $n > 6$. The greedy search methods were found to be too inconsistent and hence not used to sequence the tasks. Since the task sequencing automatically includes the optimal path-search method, the path-search algorithm need not be performed separately. In the next chapter, the components built so far will be integrated to realize the reference scenario using these optimization methods along with some numerical examples.

The aim of these methods is not to impose a control on pedestrians such that each pedestrian chooses the suggested optimal path, but rather to provide a service to the pedestrian by planning the visit – either in advance before the visit or in real-time through mobile navigation devices. The methods addressed here are based on the data produced from the pedestrian simulation. Since, a human behavior is extremely complex to model, minor situations such as abrupt stopping, slow moving pedestrians, etc., which last for a short duration are not possible to model. In most cases, such situations hardly make an impact in the total time taken to execute the tasks. Therefore, such situations are neglected.

Such scheduling methods can be useful in many other situations. Some examples include: visit to a theme park, where a visitor would like to visit as many attractions as possible within the day, navigation of disabled people can be improved based on the congestions that occur along the path (difficulties with wheel chair if the path is congested), planning a schedule in a large clinic or hospital, where a patient must be routed through several departments such as blood test lab, X-ray lab, registration, OP, ward, etc.

Spontaneous changes such as emergency situations, accidents, breakdown, etc., are not computed through this simulation since the real time data is not available to the simulation. However, such situations can make an impact to the overall pedestrian schedule. Research in the field of context aware applications that use the pervasive[7] or ubiquitous[8] computing methods is being made to exploit information about a users environment to provide improved services [HKL$^+$99]. The use of sensors to identify the local information as well as situations such as emergency situations, accidents, breakdowns, etc., is widespread in pervasive and ubiquitous computing. These sensors transmit the data using communication systems to a central server such that the context aware applications make use of these data to provide services accordingly. Coupling the pedestrian model together with such servers can help update the status of the geometry model and thereby make necessary changes to the simulation database such that the scheduling algorithm considers these parameters for visit planning. At the moment, this interface however still remains open.

[7]Pervasive Computing is a term for the strongly emerging trend toward numerous, casually accessible, often invisible computing devices that are frequently mobile or embedded in the environment and connected to an increasingly ubiquitous network structure.

[8]Ubiquitous computing enhances computer use by making several computers available throughout a physical location, while having them effectively invisible to the user.

Chapter 6

Support for Intelligent Pedestrian Navigation

Any sufficient advanced technology
is indistinguishable from magic.
– **Arthur C. Clarke**

So far, we have discussed various methods to model and simulate pedestrian behavior, integrate the simulation in a geometric scenario, optimally schedule a visit and navigate the pedestrians within the reference model. In this chapter, we realize the reference scenario mentioned in chapter 1 with some hypothetical parameters by developing an environment for an intelligent pedestrian navigation. This is done by connecting all the components available so far within the pedestrian simulation framework such that the scenario can be modeled and simulated and by using the simulation data, the pedestrian can be optimally guided within the scenario to execute his tasks.

Over the last decade, the use of mobile navigation devices is becoming increasingly common. This is due to the cheap availability of GPS receivers and portable low-power computers. Navigation devices are not just restricted to cars plying on the streets but are also commonly used by pedestrians both indoors (e.g., guided navigation through a museum) and outdoors (e.g., hiking trials with 3D topography information). Apart from guided navigation system, many service providers offer additional dynamic data for navigation such as traffic information, radar traps, deviations and construction sites, cheap restaurants and shopping possibilities, parking situations, etc. Research has been done to establish a clear communication and information interchange between the hardware devices and service data such that the customer benefits to the maximum extent possible.

This chapter is structured as follows. Initially, the specifications of the reference model as well as the details regarding the graph and architectural data extracted from the reference model are presented. Then, the parameters concerning the simulation such as pedestrian profiles, speed parameters, simulation settings, etc., are discussed. Simulations on hypothetical scenarios and the various parameter settings are performed and the simulation data is collected. Then, the reference scenario, where a pedestrian wishes to execute many tasks, is realized using the data from the simulation. Finally, a possibility of providing pedestrian navigation using the graph and the simulation data is investigated.

6.1 Geometry Model and Parameters

As mentioned earlier, the new computer science building at the Universität Stuttgart (see Figure 6.1) is used as a reference model in this thesis. The CAD model of the computer science building [GSG04] was built using the Maya software from Alias (now owned by Autodesk). The final CAD data consisted of about 156,000 triangles and the data including textures and colors consumes about 36 MB of memory. The dimensions of the building measure approximately $67m \times 79m \times 16m$ $(l \times b \times h)$ including basement and the building consists of a total of four floors. As explained in chapter 3, the Pathscan tool is used to extract and modify the graph from the CAD model as well as identify the destinations, where a pedestrian can visit. The initial graph extracted from Pathscan consisted of 2920 edges and 2277 nodes. The reduced graph however consisted of 600 edges and 544 nodes. In the reference model, the customer has a choice of about 250 destinations to visit. The path capacities are determined from the surface area available for each path. The room capacities however depend on the type of the room and not based on area. The path types such as stairs or ramps are partially identified automatically. Path types such as emergency exits, private paths, etc., were defined manually using the Pathscan tool. For test cases, certain paths where defined with specific types hypothetically. Each room receives a `ROOM_ID` and the ID is used for communication with the room and pedestrian profiles. The complete graph data is collected in an XML data, which is then used by the simulation to simulate the pedestrian behavior.

The graph data extracted contains the set of edges and set of nodes. Each node data is identified by its co-ordinates (x, y, z), the `ROOM_ID` if the node lies in the room and the ID of the edges to which the node is connected. Similarly, each edge is identified by the nodes (`NODE1` and `NODE2`), length of the edge, the capacity of the edge, specifications (private, stairs, etc.). Figure 6.2 shows

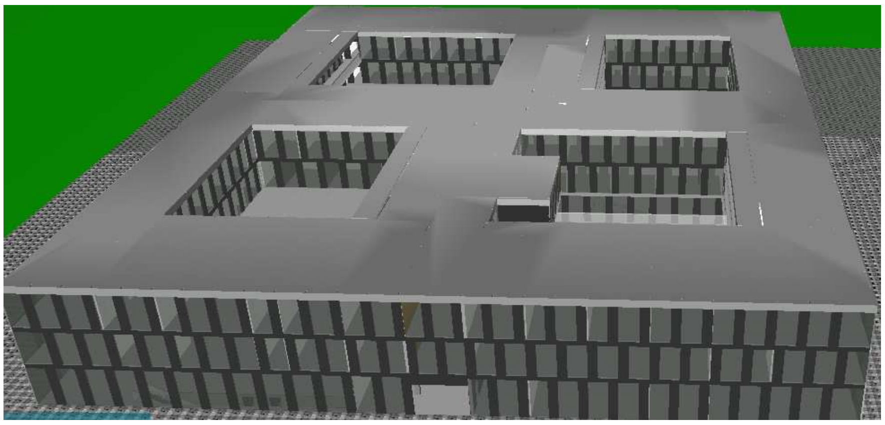


Figure 6.1: Snapshot of the VRML model of the new computer science building at the Universität Stuttgart.

the DTD (Document Type Definition)[1] representation of the XML file used to structure the graph data. A specific coding is defined to specify the path types and the codes, together with the specifications are listed separately.

Similarly, the room data extracted consists of a list of possible rooms together with the capacity, type of the destination (properties are defined such that the destinations resemble different types such as restaurant, lecture hall, etc.) and the service time. The service time is defined with a specific time period (minimum and maximum) and the type of distribution is also specified. The distribution value instructs the input modeling function to choose the appropriate distribution to determine the service time. Figure 6.3 shows the DTD representation of the XML file used to structure the room data. About 10 different room types were created and each of the 250 rooms within the reference model are classified among the 10 room types.

The room and graph data are parsed within the simulation program and the data are stored within a nested linked-list structure. From the DTD specification, it can be seen that each component is referred to each other (nodes $\Leftrightarrow$ edges, nodes $\Rightarrow$ rooms, and rooms $\Leftarrow$ room type). So using any of the ID, it is possible to extract the necessary geometry details.

[1]A DTD is one of the several SGML (Standard Generalized Markup Language) and XML schema languages. A DTD is used to express a schema through a set of declarations that conform to a particular markup syntax. DTD describe a class or type of SGML or XML in terms of the structure of the document

```
<?xml version="1.0" encoding="utf-8"?>

<!ELEMENT GRAPH ((NODE|EDGE)*)>

<!ELEMENT NODE (EMPTY)>
<!ATTLIST NODE
  ID ID #REQUIRED
  X CDATA #REQUIRED
  Y CDATA #REQUIRED
  Z CDATA #REQUIRED
  ROOM IDREF ""
  EDGES IDREFS ""
>

<!ELEMENT EDGE (EMPTY)>
<!ATTLIST EDGE
  ID ID #REQUIRED
  NODE1 IDREF #REQUIRED
  NODE2 IDREF #REQUIRED
  DIST CDATA #REQUIRED
  CAPACITY CDATA #REQUIRED
  SPEC CDATA #REQUIRED
>
```

Figure 6.2: The DTD listing of the XML file that stores the graph structure.

6.2 Pedestrian simulation

The next step involves modeling and simulating the pedestrian behavior within the geometric scenario and collecting the output data from the simulation. Several input parameters such as pedestrian data, pedestrian profiles, arrival rate and the number of pedestrians, pedestrian characteristics, etc., are required for modeling the queuing network and by using the geometric parameters obtained from the previous section, the queuing system is integrated within the graph of the reference model. This section lists out all the parameters needed to model the reference scenario, simulates the scenario and collects the data required to perform task scheduling for a new visitor.

```
<?xml version="1.0" encoding="utf-8"?>

<!ELEMENT ROOMDATA ((ROOMTYPE|ROOMLIST)*)>

<!ELEMENT ROOMTYPE (EMPTY)>
<!ATTLIST ROOMTYPE
  ID ID #REQUIRED
  TYPE CDATA #REQUIRED
  MINTIME CDATA #REQUIRED
  MAXTIME CDATA #REQUIRED
  DISTRIBUTION CDATA #REQUIRED
>

<!ATTLIST ROOMLIST
  ID ID #REQUIRED
  CAPACITY CDATA #REQUIRED
  RTYPE IDREF ""
>
```

Figure 6.3: The DTD listing of the XML file that stores the room data.

6.2.1 Input Modeling

In 4.2.3 we listed out the different pedestrian characteristics that are necessary for modeling the pedestrian behavior. In 4.2.4 we built a pedestrian profile management interface to create pedestrian profiles necessary for the simulation. We also specified that the pedestrian walking speed would be computed based on existing statistics of pedestrian behavior. We now build a function that can translate these profiles and statistical data into input data for the simulation.

6.2.1.1 Pedestrian Walking Speed

The walking speed of the pedestrian is necessary to determine the time it takes for a pedestrian to walk across an edge or a path. The walking speed is influenced by three parameters, which are

- pedestrian characteristics: active pedestrian, elderly or disabled pedestrian, tourists, commuters, etc.
- path situation: the density in both the directions of movement

- path type: stairs, wide ramps, pathways with slopes, etc.

At each queuing system representing a graph edge, the service time of the pedestrian in the queue is determined based on the length of the edge and the walking speed. While the length of the path remains constant, the walking speed varies based on the above listed parameters. The walking speed is determined as

$$v = N(1.38, 0.37) * \text{density factor} * \text{path factor} * \text{speed factor},$$

where N is a normally distributed function and v represents the pedestrian walking speed.

The speed factor represents the ratio of the speed (as compared to a the normal walking speed of an adult), which is based on pedestrian characteristics as explained in 4.2.3. The speed factor is defined within the pedestrian profile. The path factor represents the ratio of the walking speed (as compared to the walking speed on a wide ramp), which is based on path type. Each edge of the graph also contains the type of the edge and certain types have a different path speed factor (e.g., stairs=0.5*wide ramp). The density factor represents the value of the speed based on the density and the direction of movement on the edge. It was found that the speed decreases linearly with increasing density for uni-directional pedestrian movement and logarithmically for bi-directional pedestrian movement (see Figure 6.4). The density factor is determined by the following formulas [Tek02].

$$\text{density factor} = \frac{-1.7x + 62.7}{60} \quad \text{for uni-directional movement}$$

$$\text{density factor} = \frac{-13\ln(x) + 62}{60} \quad \text{for bi-directional movement}$$

where x is the pedestrian density for the given path. During the simulation, it is possible to determine the actual status of the path and thereby determine the pedestrian density along the path. Therefore, before serving each pedestrian, the current path density is used to determine the density factor and thereby the actual pedestrian service time.

6.2.1.2 Pedestrian Profile

The pedestrian profile is specified using the pedestrian management interface. A single pedestrian profile consists of the following parameters.

- speed factor: to specify speed based on pedestrian characteristics (e.g., 0.4 or 40% for elderly pedestrians)

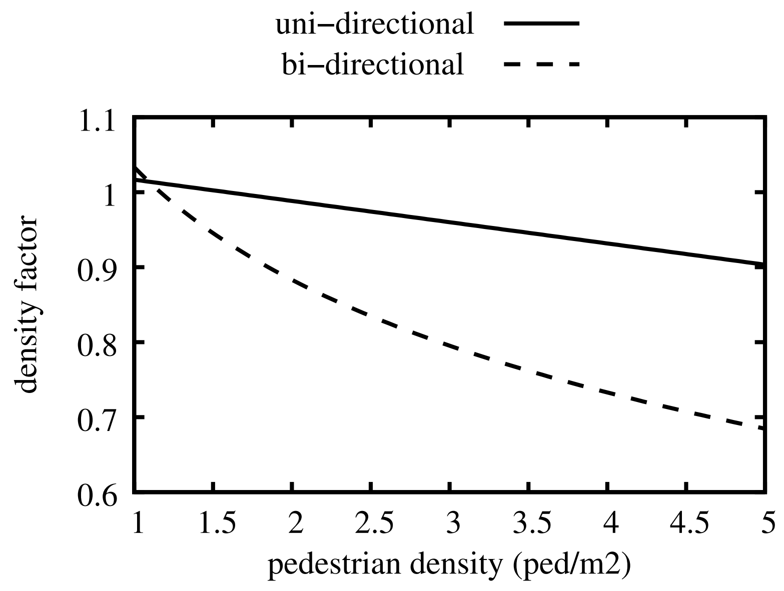


Figure 6.4: Density factor depending on the density and direction of movement.

- length of stay: specifies the average time until when the pedestrian remains in the scenario (e.g., 100 minutes)
- visit probability: the probability of visit to each destination type based on the time of the day (e.g., for type restaurant, probability of visit between 12pm and 1 pm is 0.83)
- repetition factor: average number of repeated visit for each destination type (e.g., repetition value of 2 for type cafeteria)

For both repetition factor and length of stay, stochastic functions are used to determine the exact values for each new visitor. Also if the pedestrian does not (or is not allowed to) visit a certain destination type, the repetition factor, as well as the visit probability for the destination type is set to 0. The pedestrian profile is represented using the XML format and Figure 6.5 shows the DTD representation of the pedestrian profile.

Several such profiles are created using the pedestrian management interface and are structured in an XML format. In order to simulate the given pedestrian scenario, certain profile types that fit the scenario are chosen from the master profile set. Using the pedestrian management interface, the selected profiles are exported separately together with their distribution. For test purposes, the following profile distribution was chosen (see Table 6.1).

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT CUSTOMERPROFILE (CUSTPROFILE*)>

<!ELEMENT CUSTOMER (EMPTY)>
<!ATTLIST CUSTOMER
  ID ID #REQUIRED
  SPEED CDATA #REQUIRED
  STAY DURATION CDATA #REQUIRED
>

<!ELEMENT PROFILE (EMPTY)>
<!ATTLIST CUSTOMER
  DEST TYPE CDATA #REQUIRED
  REPETETION CDATA #REQUIRED
  PROB DISTRIBUTION CDATA #REQUIRED
>
```

Figure 6.5: The DTD listing of the XML file that stores the pedestrian profile.

Profile type	Stay duration	Distribution
Employee1	400	15%
Employee2	400	15%
Visitor1	200	30%
Visitor2	200	30%
Elderly Visitor1	100	5%
Elderly Visitor2	100	5%

Table 6.1: Hypothetical test profile distribution chosen from the available profile set.

The different profile types (Employee1, Visitor1) vary only by visit probability distributions, visit restrictions and of course, speed factor. Finally, some of the destination types are time dependent and have a fixed service time (stationary processes). The queue priorities and the pedestrian behavior in the queue is not defined within the profile. Therefore, such parameters are activated during the initialization of a new pedestrian during the simulation.

6.2.2 Simulation Parameters

Apart from the pedestrian profiles, certain general parameters are used to initialize the pedestrian simulation. These parameters include

- average number of visitors for the entire simulation period
- maximum capacity of the geometric model (maximum number of pedestrians present in the scenario at any given point of time)
- time dependent arrival rate of the pedestrians
- pedestrian constraints (queue behavior and restrictions with path type)
- terminating conditions

All the above listed parameters are initial settings made before the start of the simulation. The actual number of visitors is variable and are dependent upon the allowed capacity of the geometric model and the arrival rate of the pedestrians. The pedestrian arrival rate depends on the actual time of the day. The arrival rate is negative exponentially distributed and the parameter λ is assumed to take values between 1 and 3.5 arrivals per minute for different times of the day. As a terminating condition, we perform the simulation for an actual period of 8 hours (or 9am to 5pm). We monitor the simulation time steps in terms of minutes and therefore, we execute the command `simulate(480)`. The simulation automatically terminates once the simulation clock reaches 480 or if there are no more events left over to execute (the maximum number of expected visitors has reached). For test purposes, we set consider different values for the expected number of visitors and the maximum capacity such that a comparison of the performance and the simulation data can be made.

6.2.3 Output Data and Analysis

During the simulation, the status of all the queuing systems, i.e the congestion along the paths and the waiting time situation at each destination is observed.

Apart from that, the movement of each pedestrian is also tracked during the simulation. The framework offers different possibilities to capture the congestion status for paths and destinations, namely

- path: transit time, actual number of pedestrians present, density level, and
- destination: queue size, waiting time, time taken to complete the task.

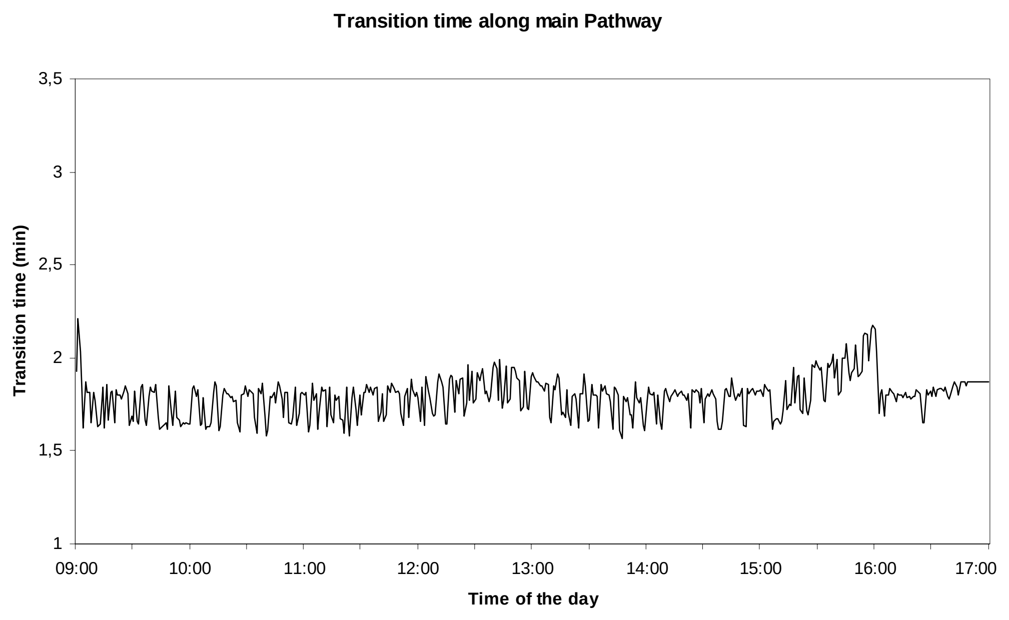
We shall now analyze a sample path and destination by simulating a hypothetical scenario with the above mentioned simulation parameters. For test purposes, we take the sample path to be the main pathway (north-south entrance/exit) and analyze the time taken to cross the entire path. The sample destination is chosen to be the one that resembles a library whose capacity is 80. The service time lies in the range of 10-30 minutes and the visit probability is such that more guests are expected around noon, followed by the morning hours. We perform two simulations with different problem sizes for a period of 8 hours (9am to 5pm).

The scenario 'A' consisted of a maximum of 1500 visitors during the simulation period, where utmost 300 pedestrians can be accommodated in the building at any given time. The simulation and the data collection took 155 seconds to complete. Scenario 'B' consisted of a maximum of 5000 visitors during the simulation period, where utmost 800 pedestrians can be accommodated in the building at any given time. The simulation and data collection of scenario 'B' took 292 seconds to complete. Figure 6.6 shows the average path transition time during the simulations.

From Figure 6.6, it can be seen that the congestions along the path remain more or less consistent. During the simulation, a certain closing time was assumed for the hypothetical scenario. Once the closing time was reached, no new visitors were allowed into the reference model. All existing tasks were interrupted and the pedestrians were evacuated from the building (a non-panic evacuation). Due to the higher number of pedestrians in scenario 'B', it can be seen that the congestions along the path increase rapidly in scenario 'B' during the closing hours. Similarly, rapid congestion increase is also seen immediately after the start of the simulation. In a sample scenario where about 500 pedestrians were present in the building, it was found that a complete (non-panic) evacuation of all the pedestrians took about 17 minutes.

Figure 6.7 shows the average waiting times observed in the library during the simulations.

From Figure 6.7, it can be seen that the congestions increase very rapidly for higher number of visitors in scenario 'B'. Since queue balking or reneging was not activated, the pedestrians wait as long as they are served. The waiting times are



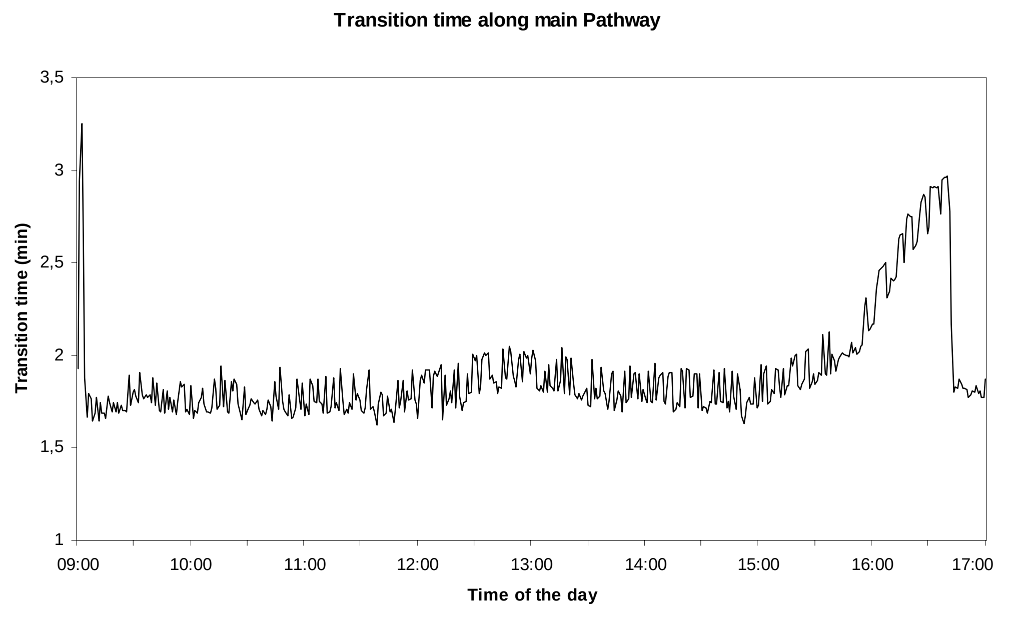
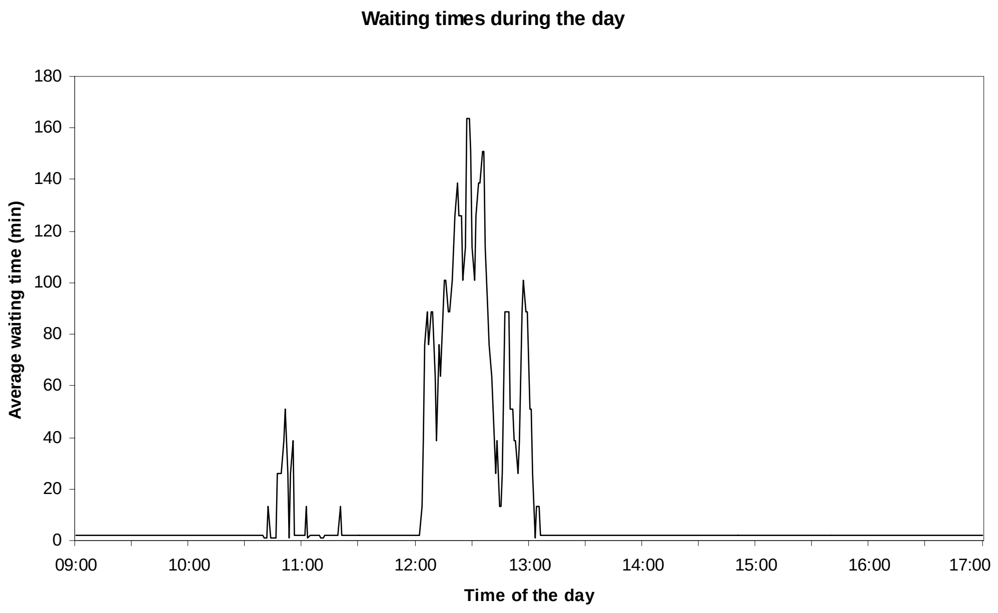


Figure 6.6: Average path transition time for scenario 'A' (top) and scenario 'B' (bottom).



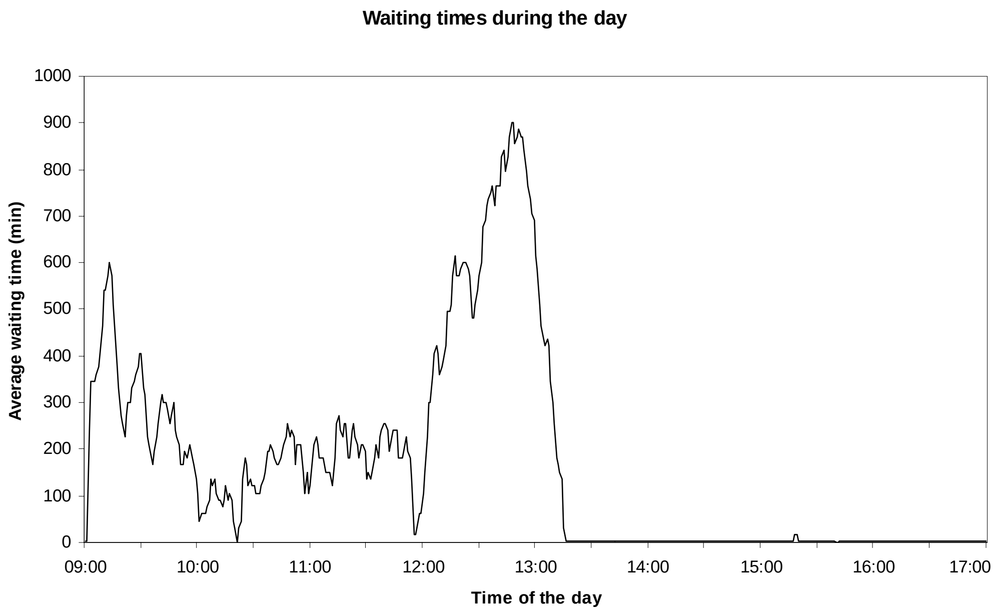


Figure 6.7: Average waiting times at a destination in scenario 'A' (top) and scenario 'B' (bottom).

estimated based on average service time and not the actual service time of each pedestrian. It can be seen that the waiting time increases to 900 minutes during noon but then drops very rapidly. This is either because of shorter service times of each pedestrian or the departure time for a pedestrian has been reached.

Similar data for all paths and destinations for various simulation scenarios were generated and the congestion data were stored in a simulation database. Using this database, a visit by a new pedestrian is planned and scheduled.

6.3 Pedestrian Visit Planning

So far, a database containing the waiting times at each destination and the transit times for each path has been created for each simulation time stamp. In chapter 5, various methods to determine an optimal schedule was presented and it was decided that for a task set with $n \leq 6$, the brute-force search will be used and for $n > 6$ the simulated annealing approach will be used. In the reference scenario, a pedestrian arrives with an intention of executing n number of tasks. Let us assume that the list of destinations and their corresponding `ROOM_ID` is known. We shall take two examples of such an hypothetical list and compare the results with and without the use of optimization methods. For test purposes, we generate a set of 10 tasks at random. We assume that the pedestrian arrives in either one of the scenarios mentioned in the previous section and executes the first 5 or all 10 tasks. Table 6.2 shows the amount of time it takes (in minutes) for a pedestrian to execute different sets of tasks. It can be seen from that table that for generally low congestions, the time difference between the optimized schedule and the default schedule chosen by the pedestrian is low. That is, the default schedule chosen by the pedestrian is as fast as an optimized sequence. However, when the scenario is crowded, it can be seen that the use of optimization methods improves the task sequence. It can also be seen that the time difference between a schedule with and without optimization methods increases for increase in problem size (more number of tasks).

6.4 Octree Modeling for Pedestrian Navigation

So far, the reference scenario has been realized and the pedestrian simulation framework is able to identify an optimal sequence and path between the tasks for a new pedestrian. However, the sequencing information must be transformed to a navigation service such that the pedestrian can guided through the various tasks. With the use of such service, dynamic changes to the sequence such as adding

	5 Tasks		
	Without Optimization	Brute-Force	Sim. Annealing
Scenario 'A'	13.708	9.889	11.488
Scenario 'B'	14.187	10.234	11.818

	10 Tasks	
	Without Optimization	Sim. Annealing
Scenario 'A'	24.854	19.653
Scenario 'B'	42.174	18.627

Table 6.2: Time taken (in minutes) to execute the tasks with or without optimization methods both in scenario 'A' and 'B'.

a new task or canceling an existing task from the list can be made. Dynamic changes are possible only if the location and the neighborhood of the pedestrian is known. For this purpose, the octree model to identify the position and search the neighborhood (as presented in chapter 3, section 3.3) is used. An example of the position identification of the pedestrian, as well as searching the neighborhood for an optimal destination is presented as follows.

Initially, the nodes of the graph are hierarchically structured in an octree. Figure 6.8 shows the octree structure for the CAD model of the reference building architecture used here.

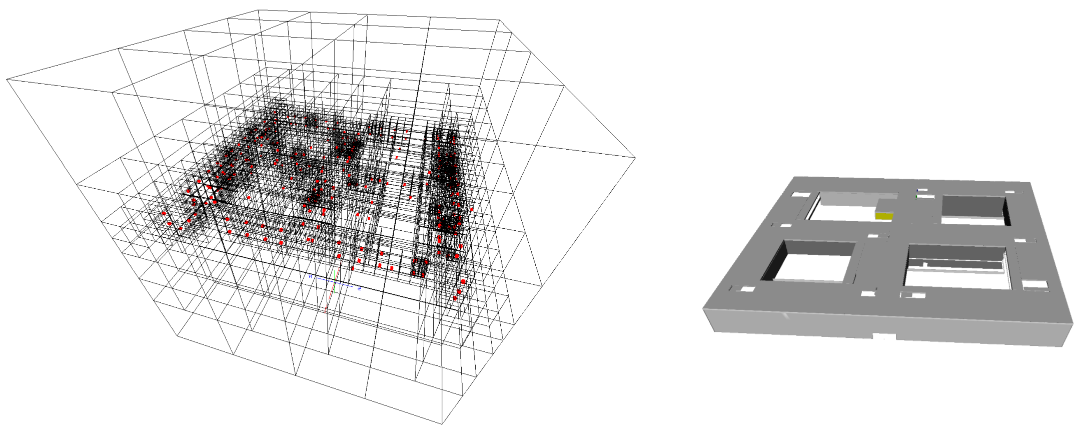


Figure 6.8: Octree structure (left) for the nodes of the graph and the actual CAD model (right).

By using the neighbor search methods, the given co-ordinates are transformed to the position on the graph. For example, the co-ordinates ($x = 36.42, y = 4.12, z = 32.05$) translates to the node with an `ID` 153 (as seen in Figure 6.9)

Now, once the position of the pedestrian within the graph is identified, it is possible to route the pedestrian to the choice of destination. Let us assume that the pedestrian would now wish to visit a destination whose type is y. Initially, the octree is rebuilt containing only the destinations whose type matches y. In other words, all nodes of the graph, which is associated with a `ROOM_ID` and the room type y, is extracted and structured in the graph. The current position of the pedestrian is also known. Therefore, with the use of neighbor search algorithm, a list of destinations in the neighborhood is identified. For simplicity reasons, the search was restricted to 2-dimensional space, i.e within the same floor. Figure 6.9 shows an example of the position identified by the octree and the list of possible destinations in the neighborhood.

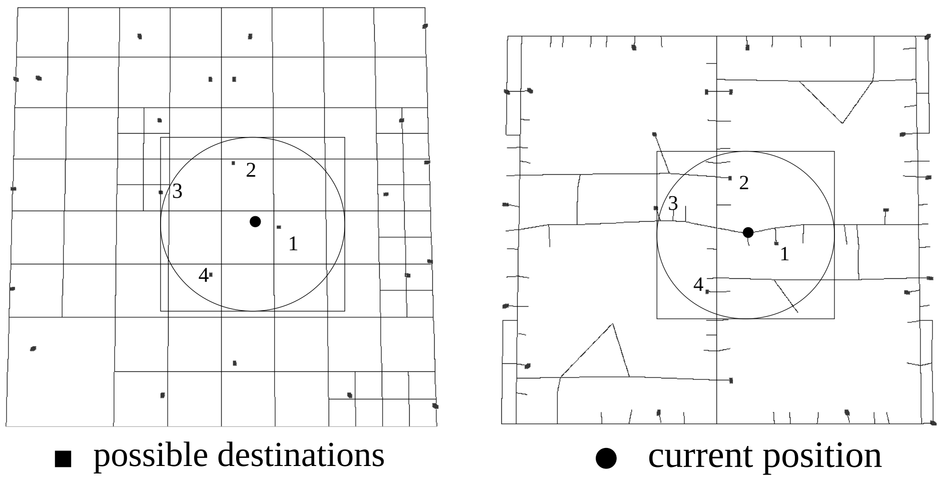


Figure 6.9: List of possible destinations positioned in the octree (left) and the graph (right).

There are now two possible ways for the pedestrian to make the choice of a destination. They are

1. static path: based purely on the distance to the destinations from the current location
2. dynamic path: based on the time it takes to reach a destination and the actual waiting time at that destination

For the static path, path-search algorithm is performed and the distance to each destination is identified. The list of destinations are then sorted according

Destination	Distance
1	7.37 m
2	12.13 m
3	14.76 m
4	17.49 m

Destination	Time until service
1	1.453 min
4	1.462 min
2	11.427 min
3	16.105 min

Table 6.3: List of destinations sorted according to distance (left) and time taken until served (right).

to the distance from the current location. For the dynamic path, the time it takes to reach each destination as well as the current waiting time at each destination is computed and the list of destinations are sorted according to the time it takes until the task can be executed. Table 6.3 shows a comparison of the list of destinations sorted according to distance and time. For the dynamic destination search, the scenario had about 200 visitors present in the building at the time the search was performed.

Similarly, a path-search algorithm can be performed also on a pre-decided schedule from the optimization methods and the path, together with the directions to the destination can be listed to the pedestrian for him to make a choice.

6.5 Summary

In this chapter, the possibility of integrating the components presented so far to realize the reference scenario and facilitate an intelligent pedestrian navigation system. With the sufficient availability of statistical information and the geometry data, it is therefore possible to prepare a system to provide navigation services to the visitors. The major advantage of such a system is that a physical presence in the scenario or a physical existence of the building is not necessary to prepare such a data. This is mainly due to the embedding of simulation within the geometry context. In fact, it is possible to build such an environment even before the construction of such a building. This also gives us an opportunity to optimize any of the parameters before the model and the scenario is realized in reality.

The system also suffers some minor drawbacks and they are listed as follows. First of all, the performance quality of the system depends on the quality of input data. Care has been taken to prepare the simulation framework as flexible as possible such that minor details can also be included into the simulation. But the collection of high quality statistical data from a real system is often a complex task. Secondly, due to the stochastic nature of the simulation framework, the calculations such as waiting time until served, path transition time, etc., are mere estimates. Even though the parameters can be fine tuned such that the estimates are as accurate as possible, the system can offer no guarantee that the time taken to complete the schedule will be exactly same as the calculated value. Finally, spontaneous changes such as elevator breakdown, accidents, emergency situations or other rare events, are not simulated within this framework. However, these problems can be tackled by the usage of sensors and communication devices to obtain such information and update the simulation accordingly. In the next chapter, a recap of all the components presented so far as well as an outlook into possible improvements and extensions are discussed.

Chapter 7

Conclusion and Outlook

Science is always wrong.
It never solves a problem without creating ten more.
– **George Bernard Shaw**

It is interesting to note that the embedding of the simulation (or the queuing system) into the geometry context of the scenario results in a totally new approach of simulating a system[1]. Due to the geometry coupling, the environmental (architectural) parameters for the simulation are naturally transferred from the geometry model to the simulation. So far, we have modeled and simulated the pedestrian behavior by embedding the queuing system into the geometry context. This opens up several possible applications that use the pedestrian model. The application discussed in this thesis was to provide an intelligent navigation service by considering the congestions and waiting times across the geometric model. Such a navigation services can find its use in a commercial center: a pedestrian wishing to visit many shops and offices within the allotted time, a theme park: a visitor wishing to visit as many attractions as possible until the closing time, or a large clinic complex: a patient is routed through the various formalities such as registration, X-ray labs or other labs, medical clinics, ward, etc. The pedestrian model can be used even for evacuation simulation to identify the bottleneck that lies in the geometric model: *does a pillar block the evacuation process?* The framework that was built provides a possibility to model and simulate the given scenario and use the outcome of the simulation to provide a navigation service – either in advance through some web-interface or in real-time with the use of some navigation device.

[1]In mobile communications, mobility of mobile agents within the geometry such as cells, cities, buildings, etc., was modeled and simulated. However, the approach described in this thesis integrates the simulation into a microscopic geometry model.

7.1 Summary of Contributions

This section summarizes the targets achieved and the contributions made by this thesis. The objectives of this thesis are realized in three main parts, namely the geometric modeling, discrete event pedestrian simulation, and pedestrian task scheduling and optimization. Throughout the thesis, a reference scenario, where a pedestrian arrives in a commercial building or a similar model to execute n set of tasks as early is possible, is used and the components needed to realize the reference scenario is built within the three concepts listed below.

Geometric Modeling

In chapter 3, the concept of geometric modeling, for the purpose of pedestrian simulation, was introduced. In the reference scenario mentioned throughout this thesis, the pedestrian simulation is strongly coupled with the geometry of the scenario, or in other words, the reference model. Due to the strong pedestrian interaction with the geometry of the scenario, several geometry and architectural parameters are extracted from the geometric model. There are two components modeled in a pedestrian simulation, namely movement of the pedestrians along the path and waiting of the pedestrians in the queue. Therefore, the paths where a pedestrian moves and the destinations where pedestrians wait are extracted and interconnected to form a graph. Properties such as type of the path, capacity of the path, architectural dimensions, location of the destinations, capacity of the destinations, destination purpose, etc., are specified within the graph.

Apart from preparing the geometric model for the pedestrian simulation, the geometric model is also used to develop a method to guide the pedestrians through different destinations in order to execute the tasks. For such a system, the graph extracted from the CAD model is hierarchically stored in an octree structure. An octree is chosen here because octrees have the natural property of storing 3-dimensional data hierarchically. Also the efficiency of location awareness (identify the position of the given co-ordinates in the graph) and neighbor search algorithms is found to be better than a linear search algorithm. Using hierarchical data structures, it is possible for a pedestrian to search the neighborhood efficiently and identify the best choice of destinations to visit.

Discrete Event Pedestrian Simulation

The pedestrian behavior in chapter 4 – movement along the paths and behavior at the destination – is modeled using the discrete event simulation methodology. The pedestrian behavior at the queue is clearly a queuing system model. Similarly, the pedestrian behavior along the path can be modeled as a queuing system, where the pedestrian arrives at the path, wait if there are any congestions and walk along the path once the necessary space is available. A queuing system is best modeled with the use of discrete event simulation. Therefore, one queuing system for each path and destination is modeled and the resulting queuing systems are integrated into the graph to form a queuing network that spans across the reference model.

The input data necessary for the pedestrian simulation was also modeled. In a queuing system embedded into the path, the event of walking along the path is determined using the walking speed of pedestrians. Certain existing statistics on pedestrian walking speed, along with the experimental data obtained from the reference model, were used to compute the service time of the pedestrians along the path. The pedestrian walking speed computed also based upon many other factors such as path density, characteristics of the pedestrian himself, type of the path, uni-directional and bi-directional movement along the paths, etc. Other parameters such as the set of tasks a pedestrian executes, typical pedestrian profiles, arrival rate, etc., are also defined in the simulation. Finally, the scenario is simulated and an overview of congestions and waiting times along the paths and destinations are obtained respectively. In this work, a flexible framework is built that can define and simulate the scenario by embedding the queuing systems into the graph derived from the CAD model, and generate the congestion information in the scenario.

Pedestrian Task Scheduling and Optimization
The reference scenario mentioned often in this thesis consists of a pedestrian arriving at a commercial center or a similar infrastructure with an intention to execute n number of tasks in the shortest possible time. Since the congestions and waiting times across the scenario can be estimated with the use of the simulation data, chapter 5 makes an attempt to use certain heuristics and combinatorial optimization methods to plan such a pedestrian visit efficiently. The visit planning consists of two parts, namely optimal path-search between two destinations and identification of a proper sequence of tasks such that the time taken to execute the tasks is minimum and the conditions of the pedestrians are also met.

The path-search algorithm uses the standard Dijkstra's shortest path algorithm to determine the shortest or the fastest path between two points. However, due to the dynamic nature of the congestion situation, a time expanded graph network is used with the Dijkstra's algorithm to function as a discrete dynamic shortest path algorithm. For each task execution, the transition to the destination is also taken into account. Therefore, the sequencing of the tasks automatically implements the path-search algorithm between the tasks.

In order to determine the optimal sequence of the tasks, different optimization methods and heuristics such as brute-force search, greedy heuristic and simulated annealing were used. It was found the simulated annealing yielded most stable results. However, it was also found that the brute force method could be used for sequencing the tasks if the number of tasks $n \leq 6$.

7.2 Outlook

Even though many methods and techniques regarding pedestrian simulation and its applications were studied, several more extensions and alternative methods are always possible. This section lists out such possible extensions during each phase of the realization of the reference scenario and further possible directions of research.

Geometric Modeling
It was shown that the Pathscan tool produced an extensive graph, which further needed to be reduced. Also, several architectural parameters such as the capacity of the destination, path restrictions, etc., were later defined manually. A possible extension in the geometric modeling to extract the graph will be to automatize the entire graph extraction as much as possible such that fine details, which were manually defined, are automatically included in the resulting graph.

Pedestrian Simulation
Even though the pedestrian simulation was performed at a microscopic level (simulation of each entity or pedestrian), the pedestrian movement was restricted to a bi-directional movement. The reduced graph was suitable for a building complex but a bi-directional pedestrian movement will not exist in open areas such as fields. Therefore, a more detailed implementation of the pedestrian simulation by considering the collision of pedestrians, a 2-dimensional pedestrian movement and implementation of additional pedestrian characteristics such as maintenance of a distance with other pedestrians, spontaneous changes, etc., can be implemented such that more generic scenarios may be simulated within this framework.

The pedestrian profiles used for input modeling consists of the probability distribution of visiting certain destination types during different times of the day. At the moment, discrete values for each hour during the day are considered and the resulting curve is translated into a numerical function, which in turn is used for the pedestrian simulation. Availability of better data collection methods such as video image processing or sensors and a finer discretization of the sampling data would increase the quality of the input data. Also, pedestrian properties such as queue behavior, task restrictions or priorities can also be included within the pedestrian profile.

Scenarios that cover a very large region (e.g., city) can also be simulated within this framework. However, each queuing system requires a certain amount of memory and processing power, and a simulation of a large graph is practically not possible in a normal PC. Therefore, parallel and distributed programming methods can be used to enhance the performance of the simulation and make the simulation framework scalable. It was shown that parallelization of the discrete event simulation does not yield promising results due to the complexity of parallelizing the event scheduler. However, attempts were made to execute independent simulations in parallel such that different scenarios can be simulated in parallel and the output data can be analyzed together. For very large graphs, efficient graph partitioning methods can be used such that independent sub-graphs are obtained and the communication required between two sub-graphs is minimum. For example, consider a campus with several buildings, each connected with some pedestrian paths. Generally, the pedestrian movement in building A is independent from building B. The only relationship between building A and B is the movement of pedestrians between these two buildings. But for a well-partitioned graph, the connecting edges between the two sub-graphs are low.

Scheduling and Routing of Pedestrians

The scheduling and routing of pedestrian tasks are performed using the data produced by the simulation. Even though the simulation tends to imitate the real system as accurate as possible, many events may not be simulated. Events such as unexpected increase in the number of visitors, failure of certain destinations, etc., make an impact with the schedule of the customers. Use of sensors or real-time local information (information from other pedestrians) is a good possibility to improve the quality of planning and scheduling pedestrian tasks.

Also, the use of additional optimization methods such as branch and bound technique, genetic algorithms, etc., may produce a good schedule in some cases. The problem of scalability was also discussed here. If a pedestrian needs to execute many tasks that are spread across a very large region, domain decomposition methods can be used to partition the large graphs into smaller sub-graphs and the schedule can be optimized locally.

Location- and Context-Aware Computing

In this thesis, a pedestrian navigation environment, which uses the congestions and waiting times obtained from the simulation, is built. In the octree-based position identification system, the position in the graph is identified using the co-ordinates given by a navigation device. Also, the use of sensors and communication devices to transmit the current status of the scenario such that spontaneous events (elevator breakdown, emergency situations) can be dynamically simulated and the service information can be updated. In order to transform the pedestrian navigation system into a full-fledged project, it must be strongly coupled with location- and context-aware computing systems. Several ongoing research projects including the Nexus project (Spatial World Models for Mobile

Context-Aware Applications) [HKL+99] at the Universität Stuttgart offer such information and services required by the simulation framework. Different possible applications can be built if such an interface is available.

First, the use of position identification devices to identify the exact location of the pedestrian in the scenario. Position identification and navigation devices are commonly used both indoors and outdoors. Typically, the usage of GPS is often combined with other position identification methods such as sensors, RFID, WLAN, etc., such that these systems can be used whenever available or whenever the GPS signal is not available. The octree-based navigation system assumes that the exact co-ordinates are available from such position identification systems. But, as discussed in 3, certain tolerance level is necessary for the system to function. Combining the octree-based position identification system with the existing navigation systems, errors in the exact location identification can be reduced and the efficiency of the system can be improved due to the hierarchical storage offered by octrees.

Secondly, discussions were also made on using location- and context-aware computing systems to provide dynamic information on spontaneous changes and the actual status of the scenario. By including such information, the simulation database can be constantly updated and the task scheduling can be performed more effectively by considering these system changes.

Third, the pedestrian simulation framework can be coupled together with pervasive computing applications. For example, consider a scenario where a pedestrian, equipped with a PDA and a navigation device, walks to the railway station to catch the train to his destination. In his PDA, the pedestrian has mentioned that he would wish to buy a pair of shoes. The application running on his PDA senses that the pedestrian still has a lot of time to get to the railway station, and the navigation device also senses that a shoe store few meters away currently offers good deals on shoes. The PDA application advises the pedestrian to go to the shoe store to look for a pair of shoes. However, the PDA application so far lacks the capability to estimate the congestions to the shoe store and the waiting time at the cash counter. Therefore, an interface with the pedestrian simulation framework can enhance the capabilities of the location- and context-aware applications.

Another issue to be discussed is the topic of privacy and security of such systems. For example: In the fictional novel of the Harry Potter series (the third novel – Harry Potter and the Prisoner of Azkaban [Row00]), Harry Potter receives a map of the Hogwart's school of witchcraft and wizardry called Marauder's Map. The map shows the movement of all witches and wizards present in the school campus with their exact location. Apparently, the MIT (Massachusetts Institute of Technology) liked the idea and implemented a similar map of the campus to display the Wi-Fi users accessing the MIT network. With these maps, it is possible to see the users logged on to the network along with their user names and the exact location in the campus. Such system is a classical violation of privacy and security. Of course, the MIT offers the user to be registered into this map system if the user wants himself to be displayed in the map. Similarly, in the framework discussed here, a pedestrian avails the service of the system to plan a visit. Thereby, the system virtually keeps track of the pedestrian as to where the pedestrian actually is located in at any given point of time. Privacy and security issues are to be considered strongly when transforming such a framework into real system.

Similar Applications using the Framework
In this thesis, we use the simulation framework to model and simulate the pedestrian behavior in a given geometric scenario and thereby use the simulation data to provide an intelligent navigation service to a pedestrian wishing to visit certain number of destinations within the building. Several other applications can be implemented using this framework.

Pedestrian evacuation simulation is one such application that can be simulated within this framework. Due to the strong coupling of the geometry model, the evacuation scenario can be simulated by considering all architectural parameters. Any architectural hindrances or bottlenecks can be detected by such a simulation. This provides an option to change the architectural layout of the building even before constructing the building such that the safety aspects are fully considered. [MBG05] made a similar attempt to detect bottlenecks in architectural models by integrating evacuation scenario into a geometric model.

Before the actual construction of the building, the expected scenario in such a building can be modeled and simulated in advance. During such simulation, possible bottlenecks and congestions are identified. The architectural plan can be optimized as often as necessary to suit a specific scenario such that the maximum throughput can be achieved. Initial attempts were made in this thesis to parallelize such a process (see chapter 4). Also, the internal planning of the building such as space allocation for different owners of the building can be optimized using the simulation. From the simulation, it is possible to define a specific visitor scenario and testing on different layouts, congestions levels can be observed and the layout can be optimized. The commercial software ShopSim developed by Savannah Simulations uses a similar method to optimize a shopping complex layout.

Finally, with the integration of the pedestrian simulation framework and a traditional traffic simulation, it is possible to simulate the movement of pedestrians (or persons in a car) across a large region. Such simulations are especially useful around a large sport complex or a theme park complex where the streets are closely connected to the complex. During a special event, extra congestions can be expected on the streets around this region. Such information can be more precisely obtained from simulations and these data can in turn be used to navigate vehicles or pedestrians, who do not intend to visit this complex, by making alternative route suggestions to avoid this region.

Appendix A

Discrete Event Simulation Library

The SIM library [BE95] was partially used to perform the discrete event simulation of the pedestrian behavior. The event scheduling strategy, as well as the data structures to store and manage queues and event list were used from the SIM library. In this appendix, an implementation of a simple M|M|1 queuing system is presented.

In [Wat93] a general purpose discrete event simulation library programmed in C called CSIM is presented. CSIM uses a 3-phase approach to schedule events and a tertiary tree structure to store and manage event list, conditional event list, and queues. Several other functionalities such as probability distributions, histogram classes and the analysis routines are also available. SIM is a C++ [BE95] adaptation of the CSIM library, developed at the Virje Universiteit, Amsterdam, in order to provide object-oriented capabilities. SIM provides such a functionality that the resulting language is similar to *simula*, which is presented in [Poo87]. The SIM library is integrated together with the HUSH graphics library [BE95] such that graphical analysis of the simulation, as well as the results can be made. The SIM library can be compiled both as ASCII version as well as the graphical version. SIM library supports both process-oriented and event-oriented approach to write simulation programs.

A typical simulation program, using the SIM library, creates and runs an application derived from the *session* class. The main function then contains the creation of a simulation object, which makes it possible to use the simulation primitives. It creates the participating events and entities, which are derived from the corresponding abstract classes and which are given the functionality in their function operators. Apart from that, the required resources and queues are also created. The resources represent passive objects used by events while queues are used by events waiting on a resource to become free. Histogram and analysis objects can be created to gather and analyze results. Before running the actual simulation, initial events and entities must be set up, which depends on the simulation.

Following is the event-oriented implementation of an elementary queuing system with Markovian arrival and service processes, and a single service unit. Initially, the events of a queuing system – arrival, departure and service – are created. Also the declaration of the simulation object, the resources, random number generators, queue, etc., are also made.

```
#include "sim.h"

simulation* sim;
generator* g;
resource* servicepoint;
queue* q;
double meanarrival, meanservice;

class arrival : public event
{
  public :
    arrival();
    virtual int operator()();
};


class departure : public event
{
  public :
    departure();
    virtual int operator()();
};


class service : public event
{
  public :
    service();
    virtual int operator()();
};
```

Initially, as a new customer arrives, the arrival event is triggered and the customer is appended into the queue. In every arrival event, the next arrival is also generated and scheduled at the time of arrival. The inter-arrival time is exponentially distributed and the λ parameter is decided based on *meanarrival*. On the completion of arrival event, the service event is automatically activated. The service operator checks for any event available in the queue and executes it as long as a free service unit is available. In the service event, the service time is also exponentially distributed and the λ parameter is decided based on *meanservice*. A departure event is scheduled once the service is complete. In the departure event, the occupied resource is freed and the customer is removed from the queuing system.

```
arrival::arrival() : event()
{
}

// arrival event.
int arrival::operator()()
{
  arrival* arr = new arrival();

  // schedule next arrival
  sim -> schedule(arr,(g -> exponential(meanarrival)));

  // append
  q -> append(this);
  return OK;
}

departure::departure() : event()
{
}

// departure event.
int departure::operator()()
{
  // release servicepoint and leave it
  servicepoint -> release();
  sim -> terminate(this);
  return OK;
}
```

```
service::service() : event()
{
}

// service event.
int service::operator()()
{
  while ((!(q->empty()))&&(servicepoint->available()))
  {
    sim -> terminate(q -> removefront());

    // acquire servicepoint and schedule departure
    servicepoint -> acquire();
    departure* d = new departure();
    sim -> schedule(d,(g -> exponential(meanservice)));
  }
  return OK;
}
```

The events are structured in such a way that once a new arrival process is initiated and a service event bound to the arrival process is put on hold, the simulation automatically triggers the successive events as long as the terminating condition is reached. Now that the event objects are declared, the simulation application is implemented. After the declarations, the first arrival is scheduled and a service unit is put on hold. Once the simulation begins, further events are scheduled in the event list.

```
int main(int argc, char ** argv)
{
  cout << "mean time between two arrival
              events (minutes)" << endl;
  cin >> meanarrival;

  cout << "mean service time (minutes)" << endl;
  cin >> meanservice;


  sim = new simulation();
  g = new generator();
  q = new queue();
  servicepoint = new resource(1);
  // initially a customer arrives
       and the server is conditional
  arrival* arr = new arrival();
  sim -> schedule(arr,0.0);

  service* sr = new service();
  sim -> hold(sr);

  // run for 8 hours or 480 minutes
  sim -> run(480.0);

  delete q;
  delete sim;
  return 0;
}
```

The above shown example is a skeletonized example of an elementary queuing system and does not include any histogram or analysis classes. Use of such classes can help us to analyze properties such as average size of the queue or average waiting times of a customer.

Bibliography

[ABB$^+$06] B. Arbter, F. Bitzer, S. Bürklen, D. Dud kow ski, C. Müller, M. Scharf, and K. Wiegerling. Approaches for Modeling Mobility in Nexus. Technical report, Institut für Straßen- und Verkehrswesen, Universität Stuttgart, 2006.

[AHU87] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley Publishing Company, 1987.

[AL97] C. Ashcraft and J.W.H. Liu. Using domain decomposition to find graph bisectors. *BIT*, 37(3):506–534, 1997.

[Ba98] V.J. Blue and J.L. adler. Emergent Fundamental Pedestrian Flows from Cellular Automata Microsimulation. *Transportation Research Record*, 1644:29–36, 1998.

[BBO$^+$01] J. Benntner, G. Bauer, G.M. Obermair, I. Morgenstern, and J. Schneider. Optimization of the time-dependent traveling salesman problem with Monte Carlo Methods. *Physical Review E*, 64, 2001.

[BBR01] M. Bauer, C. Becker, and K. Rothermel. Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. In *Workshop on Location Modeling for Ubiquitous Computing, UbiComp*, September 2001.

[BCMS06] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. A mobile computing middleware for location- and context-aware internet data services. *ACM Trans. Inter. Tech.*, 6(4):356–380, 2006.

[BE95] D. Bolier and A. Eliëns. *SIM: a C++ library for Discrete Event Simulation.* Vrije Universiteit, Amsterdam, Netherlands, October 1995.

[Ben62] J.F. Benders. partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238 – 252, 1962.

[Ber02] M. Bernreuther. *Geometrische Modellierung mit Simplizialkomplexen: Vom CAD-Modell zur numerischen Analyse.* PhD thesis, Universität Stuttgart, Stuttgart, Aug 2002.

[Béz74] P. Bézier. Mathematical and practical possibilities of UNISURF. In R. Barnhill and R. Riesenfeld, editors, *Computer Aided Geometric Design*, pages 127–152. Academic Press, 1974.

[Béz76] P. Bézier. Définition numérque des courbes et surfaces II. *Automatisme*, XII:17–21, 1976.

[BG95] J.W. Berry and M.K. Goldberg. Path optimization for graph partitioning problems. Technical Report TR 95-34, DIMACS, August 1995.

[BGOM03] R. Baños, C. Gil, J. Ortega, and F.G. Montoya. Multilevel Heuristic Algorithm for Graph Partitioning. In *3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, ESSEX (UK), April 2003. LNCS.

[BGZ04] H.-J. Bungartz, M. Griebel, and C. Zenger. *Introduction to Computer Graphics.* Charles River Media, second edition, February 2004.

[Bha01] P. Bhattacharya. Efficient Neighbor Finding Algorithms in Quadtree and Octree. Master's thesis, Indian Institute of Technology, Kanpur, 2001.

[BINN99] J. Banks, J.S. Carson II, B.L. Nelson, and D.M. Nicol. *Discrete-Event System Simulation.* Prentice-Hall, 1999.

[BM58] G. Box and M. Muller. A Note on Generation of Random Normal Deviates. *Annals Math. Stats.*, 29:610 – 611, 1958.

[Boa85] Transportation Research Board. Highway Capacity Manual. Special Report 204 TRB, 1985.

[Bra00] N. Bradley. *The XML Companion.* Addison-Wesley, 2000.

[Bru04] P. Brucker. *Scheduling Algorithms.* Springer Verlag, 2004.

[BS06] S. Brailsford and D. Stubbins. Using Discrete-Event Simulation to Model Emergency Evacuation of a Public Building. In *Proceedings of the 2006 OR society Simulation Workshop*, 2006.

[Cas05] A. Castro. Pedestrian evacuation simulation. *J. Comput. small Coll.*, 20(5):141–142, 2005. Consortium for Computing Sciences in Colleges, USA.

[CGT96] F. Cao, J.R. Gilbert, and S.-H. Teng. Partitioning meshes with lines and planes. Technical Report CSL-96-01, Parc Xerox, January 1996.

[CH66] K.L. Cooke and E. Hasley. The shortest route through a network with time-dependent intermodal transit times. *J. of Math. Analysis and Applications*, 14:493–498, 1966.

[Cha97] I. Chabini. A new algorithm for shortest paths in discrete dynamic networks. In *8th IFAC/IFIP/IFORS Symp. on transportation systems*, 1997.

[Cha98] I. Chabini. Discrete dynamic shortest path problems in transportation applications. *Transportation Research Record*, 1645, 1998.

[CP89] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Manage. Sci.*, 35(2):164–176, 1989.

[dC63] P. de Caslteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.

[Dea99] B. C. Dean. Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271, 1959.

[Dre69] S.E. Dreyfus. An appraisal of some shortest path algorithms. *Operations Research*, 17:395–412, 1969.

[Dre03] T. Drexl. Entwicklung intelligenter Pfadsuchsysteme für Architekturmodelle am Beispiel eines Kiosksystems (Info-Point) für die FMI in Garching. Diplomarbeit, Institut für Informatik, Technische Universität München, 2003.

[EFS03] F. Eickhoff, J. Fenchel, and D. Scheck. *Bewertung von Positionierungssystemen für den Innenbereich für den Einsatz innerhalb des NEXUS-Projekts.* Fachstudie, Universität Stuttgart, 2003.

[EJ90] J.R. Van Eck and T. De Jong. Adapting datastructures and algorithms for a faster transport network computations. *Proceedings of the 4th Int. Symp. on Spatial Data Handling*, 1:295 – 304, 1990.

[Eli95] A. Eliens. *HUSH – a C++ API for Tcl/Tk.* The X Resource, 1995.

[Far02a] G. Farin. A history of curves and surfaces in CAGD. In G. Farin, J. Hoschek, and M.-S. Kim, editors, *Handbook of 3D Modeling and Graphics*, pages 1–22. Elsevier, 2002.

[Far02b] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide.* Morgan Kaufmann Publishers, San Francisco, CA, 5 edition, 2002.

[FB74] R. Finkel and J.J. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[FP02] S.F. Frisken and R.N. Perry. Simple and Efficient Traversal Methods for Quadtrees and Octrees. *Journal of Graphics Tools*, 7(3):1 – 11, 2002.

[Fra00] A. Frank. *Organisationsprinzipien zur Integration von geometrischer Modellierung, numerischer Simulation und Visualisierung.* Dissertation, TU München, München, 2000.

[Fru71a] J.J. Fruin. Designing for Pedestrians: A Level of Service Concept. *Highway Research Record*, 355:1–15, 1971.

[Fru71b] J.J. Fruin. *Pedestrian Planning and Design.* Metropolitan Association of Urban Designers and Environmental Planners, Inc. New York, 1971.

[FSMHJ05] M. Friedrich, G. Schleupen, M. Moltenbrey, and H.-J.Bungartz. A parallel implementation of a schedule-based transit assignment algorithm for large networks. In F. Hülsemann, M. Kowarschik, and Ulrich Rüde, editors, *Proceedings of the 18th Symposium Simulationstechnique (ASIM 2005)*, Fortschritte in der Simulationstechnik - Frontiers in Simulation, Erlangen, September 2005. SCS European Publishing House.

[Fuj90] R.M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[Fuj00] R.M. Fujimoto. *Parallel and Distributed Simulation Systems.* John Wiley and Sons, 2000.

[Gan06] X. Gan. *A GUI-Based Pedestrian Profile Management Interface.* Universität Stuttgart, 2006.

[GG75] M.R. Garey and R.L. Graham. Bounds on multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4:187–200, 1975.

[Gip87] P.G. Gipps. Simulation of pedestrian traffic in buildings,. Technical Report 35, Institut für Verkehrswesen, Universität Karlsruhe, Karlsruhe, 1987.

[GM85] P.G. Gipps and B. Marksjo. A Micro-Simulation Model for Pedestrian Flows. *Mathematics and Computers in Simulation*, 27:95–105, 1985.

[GSBL00] A. Garrido, M.A. Salido, F. Barber, and M.A. López. Heuristic methods for solving job-shop scheduling problems, 2000.

[GSG04] S. Giesecke, M. Stier, and S. Grumbein. *Pfadsuche in Architekturmodellen und Stereoprojection.* Softwarepraktikum, Universität Stuttgart, 2004.

[HB01a] J. Hightower and G. Boriello. Location sensing techniques. Technical report, University of Washington, Seattle, WA, 2001.

[HB01b] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, 2001.

[HB01c] S.P. Hoogendoorn and P.H.L. Bovy. Generic gas-kinetic traffic systems modeling with applications to vehicular traffic flow. *Transportation Research B – Methodological*, 35:317–336, 2001.

[HBD02] S.P. Hoogendroon, P.H.L Bovy, and W. Daamen. Microscopic Pedestrian Wayfinding and Dynamics Modelling. In *In Pedestrian and evacuation dynamics*, 2002.

[Hel90] D. Helbing. Physikalische Modellierung des dynamischen Verhaltens von Fußgängern. Master's thesis, Georg-August University, Göttingen, 1990.

[Hel92a] D. HELBING. A fluid-dynamic model for the movement of pedestrians. *Complex Systems*, 6(6):391–415, 1992.

[Hel92b] D. HELBING. *Stochastische Methoden, nichtlineare Dynamik und quantative Modelle sozialer Prozesse*. PhD thesis, Universität Stuttgart, 1992.

[Hen71] L.F. HENDERSON. The statistics of crowd fluids. *Nature*, 229(5):381 – 383, 1971.

[Hen74] L.F. HENDERSON. On fluid mechanics of human crowd motion. *Transportation Research*, 8:509 – 515, 1974.

[HFMV02] D. HELBING, I. FARKAS, P. MOLNAR, AND T. VICSEK. Simulation of pedestrian crowds in normal and evacuation situations. In *In Pedestrian and evacuation dynamics*, pages 21–58, 2002.

[HFV00] D. HELBING, I.J. FARKAS, AND T. VICSEK. Simulating Dynamic Features of Escape Panic. *Nature*, 407:487 – 490, 2000.

[HKL+99] F. HOHL, U. KUBACH, A. LEONHARDI, K. ROTHERMEL, AND M. SCHWEHM. Next Century Challenges: Nexus – An Open Global Infrastructure for Spatial-Aware Applications. In *Proc. of the fifth Annual Intl. Conf. on Mobile Computing and Networking (MobiCom'99)*, 1999.

[HL72] L.F. HENDERSON AND D.J. LYONS. Sexual differences in Human Crowds Motion. *Nature*, 240:353 – 355, 1972.

[HM95] D. HELBING AND P. MOLNAR. Social Force Model for Pedestrian Dynamics. *Physical Review E*, 51:4282, 1995.

[Hoo06] J. N. HOOKER. An integrated method for planning and scheduling to minimize tardiness. *Constraints*, 11(2-3):139–157, 2006.

[HS00] M. HARRY AND R. SCHROEDER. *Six Sigma*. Random House, 2000.

[HTRS03] A. HANISCH, J. TOLUJEW, K. RICHTER, AND T. SCHULZE. Online Simulation of Pedestrian Flow in Public Buildings. In *Proc. of the 2003 Winter Simulation Conference*, 2003.

[Hud00] J. HUDSAL. Fastest path problems in dynamic transportation networks. Technical report, University of Leicester, UK, 2000.

[HW04] C.M. Henein and T. White. Agent-based modelling of forces in crowds. In P. Davidsson, L. Gasser, B. Logan, and K. Takadama, editors, *Multi-Agent and Multi-Agent-Based Simulation*, volume 3415 of *Lecture Notes in Computer Science*, page 173. Springer, 2004.

[Jef85] D.R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[Jen96] F. Jensen. *An Introduction to Bayesian Networks.* Springer-Verlag, 1996.

[JH94] S.L. Jaynes and J.O. Hoffman. Discrete Event Simulation for Quick Service Restaurant Traffic Analysis. In *Proc. of the Winter Simulation Conference*, 1994.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and Jr. M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220, 1983.

[KHW01] J. Kerridge, J. Hine, and M. Wigan. Agent-based modelling of pedestrian movements. *Environment and Planning B: Planning and Design*, 28(3):327 – 341, 2001.

[KL70] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphics. *The Bell Sys. Tech.*, 49(2):291–307, 1970.

[KPN96] R. Knoblauch, M. Pietrucha, and M. Nitzburg. Field Studies of Pedestrian Walking Speed and Start-up Time. *Transportation Research Record*, 1538, 1996.

[L'E90] P. L'Ecuyer. Random numbers for simulation. *Commun. ACM*, 33(10):85–97, 1990.

[Lit61] J.D.C. Little. A Proof for the Queueing Formula $L = \lambda w$. *Operations Research*, 16:651 – 665, 1961.

[LK00] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill Inc., third edition, 2000.

[Lov94] G.G. Lovas. Modeling and Simulation of Pedestrian Traffic Flow. *Transportation Research*, 28B:429–443, 1994.

[LTP$^+$90] Y.J. Lu, Y.Y. Tang, P Pirard, Y.H. Hsu, and H.D. Cheng. Measurement of Pedestrian Flow Data Using Image Analysis Technique. *Transportation Research Record*, 1281:87–96, 1990.

[Mar] A.A. Markov. Extension of the limit theorems of probability theory to a sum of variables connected in a chain.

[May89] A.D. May. *Traffic Flow Fundamentals.* Prentice Hall, New Jersey, 1989.

[MB04] R.-P. Mundani and H.-J. Bungartz. An octree-based framework for process integration in structural engineering. In *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, volume II, pages 197–202. International Institute of Informatics and Systemics, 2004.

[MB06] M. Moltenbrey and H.-J. Bungartz. Design and Implementation of a Fine Grain Microscopic Traffic Simulator with Integrated Timetable-based Public Transportation. In *Proceedings of the 19th Symposium Simulationstechnique (ASIM 2006), Hannover*, 2006.

[MBG05] R.-P. Mundani, H.-J. Bungartz, and S. Giesecke. Integrating evacuation planning into an octree-based cscw framework for structural engineering. In *Proceedings of the 22nd Conference on Information Technology in Construction (cibW78)*, pages 435–440. Institute for Construction Informatics, 2005.

[MBR+03] R.-P. Mundani, H.-J. Bungartz, E. Rank, E. Romberg, and A. Niggl. Efficient Algorithms for Octree-Based Geometric Modeling. In B.H.V. Topping, editor, *Proc. of the Ninth Int. Conf. on Civil and Structural Engineering Computing*, Civil-Comp Press, 2003.

[Mei04] D. Meidlinger. Erstellung eines Simulationswerkzeuges zur Ermittlung von Reisezeiten bei Nutzung von Zielführungssystemen unter besonderer Beachtung der Verkehrsmeldungsqualitäten. Master's thesis, Universität Stuttgart, November 2004.

[Mis86] J. Misra. Distributed Discrete-Event Simulation. *Computing Surveys*, 18(1):39 – 65, 1986.

[Mor66] G.M. Morton. A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing. Technical report, IBM, Ottawa, Canada, 1966.

[Mou01] A.V. Moudon. Targeting Pedestrian Infrastructure Improvements. Technical report, Department of Urban Design and Planning, University of Washington, 2001.

[MRea02] A. Madureira, C. Ramos, and et al. A new framework for dynamic deterministic job-shop scheduling problems using genetic algorithms, 2002.

[MT63] J.F. Muth and G.L. Thompson. *Industrial Scheduling.* Printice-Hall, 1963.

[Mun06] R.-P. Mundani. *Hierarchische Geometriemodelle zur Einbettung verteilter Simulationsaufgaben.* Dissertation, Universität Stuttgart, 2006.

[NB05] S. Narasimhan and H.-J. Bungartz. Congestion-Aware Optimization of Pedestrian Paths. In Hülsemann, Frank and Kowarschik, Markus and Rüde, Ulrich, editor, *Proceedings of the 18th Symposium Simulationstechnique ASIM 2005*, pages 242 – 247, Erlangen, Germany, September 2005. Erlangen: SCS Publishing House.

[NB06a] S. Narasimhan and H.-J. Bungartz. A Framework for A Graph- and Queuing System-Based Pedestrian Simulation. In H. R. Arabnia, editor, *Proceedings of the 2006 International Conference on Modeling, Simulation and Visualization Methods (MSV'06)*, pages 87 – 93, Las Vegas, USA, 2006. CSREA Press.

[NB06b] S. Narasimhan and H.-J. Bungartz. Methods on Optimal Pedestrian Traffic Scheduling and Routing. In *Proc. of the 25th Workshop of the UK Planning and Scheduling SIG*, 2006.

[Nel95] R. Nelson. *Probabiliy, Stochastic Processes, and Queueing Theory.* Springer-Verlag, 1995.

[NF94] D.M. Nicol and R.M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, 53(1):249–285, 1994.

[NGS$^+$01] D. Nicklas, M. Grossmann, T. Schwarz, S. Volz, and B. Mitschang. A Model-Based, Open Architecture for Mobile, Spatially Aware Applications. In S.C. Jensen, M. Schneider, B. Seeger, and J.V. Tsotras, editors, *Proceedings of the 7th International Symposium on Spatial and Temporal Databases: SSTD 2001*, pages 117–135, Redondo Beach, CA, USA, 2001.

[NMB06] S. Narasimhan, R.-P. Mundani, and H.-J. Bungartz. An Octree- and A Graph-Based Approach to Support Location Aware Navigation Services. In H. R. Arabnia, editor, *Proceedings of the*

2006 International Conference on Pervasive Computing and Systems (PSV'06), pages 24 – 30, Las Vegas, USA, 2006. CSREA Press.

[NRMB05] A. Niggl, E. Rank, R.-P. Mundani, and H.-J Bungartz. Organizing a p-Version Finite Element Computation by an Octree-Based Hierarchy. In *Proceedings of the International Conference on Adaptive Modeling and Simulation.* Lehrstuhl für Bauinformatik, Technische Universität München, International Center for Numerical Methods in Engineering, 2005.

[NRMHJ05] A. Niggl, E. Rank, R.-P. Mundani, and H.-J.Bungartz. Organizing a p-version finite element computation by an octree-based hierarchy. In *Proceedings of the International Conference on Adaptive Modeling and Simulation.* Lehrstuhl für Bauinformatik, Technische Universität München, International Center for Numerical Methods in Engineering, 2005.

[NS92] K. Nagel and M. Schreckenberg. A Cellular Automaton Model for Freeway Traffic. *J. Phys. I*, 2:2221–2229, 1992.

[O'F97] C.A. O'Flaherty, editor. *Transport Planning and Traffic Engineering.* Arnold, London, 1997.

[OM81] S. Okazaki and S. Matsushita. A Study of Pedestrian Movement in Architectural Space Part 5: A Proubing walk and guide walk by a guideboard. *J. of Architecture, Planning, Environment Engineering*, 302:87–93, 1981.

[OM93] S. Okazaki and S. Matsushita. A study of simulation model for pedestrian movement with evacuation and queuing. In *Proceedings of the Intl. Conf. on Engineering for Crowd Safety*, 1993.

[oTE94] Institute of Transportation Engineers. *Manual of Transportation Engineering Studies.* Printice Hall, 1994.

[Pap84] A. Papoulis. *Poisson Process and Shot Noise*, chapter 16, pages 554 – 576. Probability, Random Variables, and Stochastic Processes. McGraw Hill, second edition, 1984.

[Pat06] V. Patmanathan. Area Localization using WLAN. Master's thesis, KTH, Stockholm, 2006.

[Pet62] C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Universität Bonn, 1962.

[Poi37] S.D. Poisson. Recherches sur la Probabilite des Jugements en Matiere Criminelle et en Metiere Civile. *Precedees des Regles Generales du Calcul des Probabilities*, 1837.

[Poo87] R.J. Pooley. *An Introduction to Programming in Simula.* Blackwell Scientific Publications, 1987.

[PPR$^+$03] P. Pattanamekar, D. Park, L.R. Rilett, J. Lee, and C. Lee. Dynamic and stochastic shortest path in transportation networks with two components of travel time uncertainty. *Transportation Research*, 11:331–354, 2003.

[Qui03] M.J. Quinn. *Parallel Programming in C with MPI and OpenMP.* McGraw-Hill, 2003.

[Rei94] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications.* Lecture Notes in Computer Science. Springer, 1994.

[Req77] A.A.G. Requicha. Mathematical Models of Rigid Solid. Production Automation Project Memo 28, University of Rochester, Rochester, NY, 1977.

[Rob] N. Robins. *Smooth Normal Generation with Preservation of Edges.*

[Row00] J.K. Rowling. *Harry Potter and the Prisoner of Azkaban.* Bloomsbury Publishing, 2000.

[Sag94] H. Sagan. *Space-Filling Curves.* Springer Verlag, 1994.

[Sam84] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.

[Sat01] M. Satyanarayanan. Pervasive Computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, August 2001.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, December 1994.

[SB01] T.J. Schriber and D.T. Brunner. Inside Discrete-Event Simulation Software: How it works and why it matters. In *Proc. of the 2001 Winter Simulation conference*, 2001.

[SCM06] G. Stecco, J.-F. Cordeau, and E. Moretti. Solving a Production Scheduling Problem as a Time-Dependent Traveling Salesman Problem, 2006.

[Sed98] R. Sedgewick. *Algorithms in C++*. Addison-Wesley Publishing Company, 1998.

[SKK00] K. Schloegel, G. Karypis, and V. Kumar. *Graph Partitioning for High Performance Scientific Simulations.* CRPC Parallel computing Handbook, Morgan Kaufmann, 2000.

[Spa03] J. C. Spall. *Introduction to Stochastic Search and Optimization.* Wiley-VCH, 2003.

[SRS$^+$95] M.J. Sullivan, C.A. Richards, C.E. Smith, O. Masoud, and N.P. Papanikolopoulos. Pedestrian tracking from a stationary camera using active deformable models. In *prof. of the Intelligent Vehicles'95*, 1995.

[SS99] J. Sethuraman and M.S. Squillante. Optimal stochastic scheduling in multiclass parallel queues. In *SIGMETRICS '99: Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 93–102, New York, NY, USA, 1999. ACM Press.

[ST93] M. Spreitzer and M. Theimer. Providing Location Information in a Ubiquitous Computing Environment. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, December 1993.

[Str01] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley Publishing Company, 2001.

[TBdS06] M.C. Toyama, A.L.C. Bazzan, and R. da Silva. An agent-based simulation of pedestrian dynamics: from lane formation to auditorium evacuation. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 108–110, New York, NY, USA, 2006. ACM Press.

[Tek02] K. Teknomo. *Microscopic Pedestrian Flow Characteristics: Development of an Image Processing Data Collection and Simulation Model.* PhD thesis, Tohoku University, Japan, 2002.

[Thi01] E. Thiele. Simulation des Verkehrflusses in einem Strassennetz durch Strömungssimulation in zweidimensionalen Berechnungsgebieten. Master's thesis, Universität Stuttgart, 2001.

[TSKT95] M. Tsuchikawa, A. Sato, H. Koike, and A Tomono. A moving-object extraction method robust against illumination level changes for a pedestrian counting system. In *Proc. of the Intl. Symp. on Computer Vision*, 1995.

[TTI01] K. Teknomo, Y. Takayema, and H. Inamura. Microscopic Pedestrian Simulation Model to Evaluate "Lane-Like Segregation" of Pedestrian Crossing. In *Proc. of the Infrastructure Planning Conference*, 2001.

[Tur36] A.M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *2*, pages 230–265, 1936.

[Tys99] J. Tyszer. *Object-Oriented Computer Simulation of Discrete-Evemt Systems.* Kluwer Academic Publishers, 1999.

[UM06] Z. Usmani and R. Menezes. Increasing Sales in Supermarkets via Real-Time information about Customer's Activities – The Swarm Moves Simulation. In *Proceedings of the Intl. Conf. in Modeling, Simulation and Visualization (MSV'06)*, 2006.

[Wal77] A.J. Walker. An Efficient Method for Generating Discrete Random Variables with General Distrbutions. *ACM Trans. Math. software*, 3:253–256, 1977.

[Wat93] K. Watkins. *Discrete Event Simulation in C.* McGraw Hill, 1993.

[WBHW03] J.-P. Watson, J.C. Beck, A.E. Howe, and L.D. Whitley. Problem difficulty for tabu search in job-shop scheduling. *Artif. Intell.*, 143(2):189–217, 2003.

[Wer96] M. Wermuth. *Modelvorstellung zur Prognose.* Stadtverkehrsplannung – Grundlagen – Methoden – Ziele. Springer Verlag, Berlin, 1996.

[WK06] A. Wimmler and R. Kofler. Entwurf eines mikroskopischen Verkehrssimulators mit integriertem fahrplanfeinem öffentlichen Verkehr. Software Project, 2006.

[Wu06] S. Wu. *Observer: A visualization tool for the pedestrian simulation framework.* Universität Stuttgart, 2006.

[YB06] R. Yoder and P. Bloniarz. A Practical Algorithm for Computing Neighbors in Quadtrees, Octrees and Hyperoctrees. In *Proceedings of the Intl. Conf. on Modeling, Simulation and Visualization Methods (MSV'06)*, Las Vegas, NV, 2006.

[YM94] S. Yasutomi and H. Mori. A method for discriminating of pedestrian based on rhythm. In *Proc. of the IEEE/RSJ/GI Intl. Conf. on Intelligent Robots and Systems*, 1994.