Parameter Search in an Agent-based Model of Pedestrian Movement in Retail Environments

by

Thananat Jitapunkul

S.B., E.E.C.S. M.I.T., 2010

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
August 20, 2010

Certified by .....
Deb K. Roy
Associate Professor
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Parameter Search in an Agent-based Model of Pedestrian Movement in Retail Environments

by

Thananat Jitapunkul

Submitted to the
Department of Electrical Engineering and Computer Science
August 20, 2010
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Parameter search in an agent-based model of pedestrian movement in retail environments is part of a research effort by data-driven architecture in the Cognitive Machine Group at the MIT Media Lab. The approach pursued in this thesis is agent-based modelling, with an ultimate goal to use generative behaviors in agents to study effects of architectural and managerial decisions on retail environments.

In this thesis, I designed and implemented an agent training module as a part of a software system which simulates and learns patterns of human pedestrian movement in retail environments. This thesis covers two different components: (1) the implementation of a hill-climbing training module and (2) a pedestrian path comparison metric. To measure the module's performance, the system is tested against video sequences collected from the actual retail environment.

Thesis Supervisor: Deb K. Roy
Title: Associate Professor

Acknowledgments

To think that I have reached the end of my study at MIT and am earning an M.Eng. degree, I must confess that it has been an incredibly reckless adventure that I went through during the past four full years. While it was fun being a part of the MIT community and learning many things, both good and bad, along the way, it is the time to say goodbye and move on to a new start in life.

I might not have made it this far without many supporters to get me through hard times which, indeed, MIT has given me plenty! Therefore, for many of the people whom I am indebted to, please take this acknowledgement as my farewell notice.

The first and foremost people whom I would like to thank are my parents and my family, who have always been my moral and emotional support as well as my role models. Although I rebelled and became irrational at times, I can assure you that you have all my love and respects. Noo+ Bam, thank you for being such a unique sister, and I hope that you will eventually stop calling me Moo+ Uan soon. I definitely miss you all and I will find time to go home to meet you guys some time in the near future.

The next people whom I would like to thank are my thesis supervisor, Professor Deb Roy, and Rony Kubat, my thesis mentor. Without your advice and the foundation of the project that you have been working on, this thesis would not have been possible. It has been an interesting and inspiring project which I have been tackling for one full year and I hope that this project will get to its ultimate goal in a very near future! Also, thank you for not yelling at me when I procrastinated too much. I am also indebted to Deb for his project on Trisk, the perception-grounding robot, which I have also worked on in my freshman and sophomore year.

Another person whom I am greatly indebted to is my academic advisor, Professor Patrick Winston. Thank you for being my advisor and friend for the last three and a half years. I know that I have been too stubborn and did not take many of your advice that seriously, (which I confess to have "sometimes" regretted), but, at the very least, I am finally safe and sound out of MIT! Also, I would like to show my appreciation to you for being my support (and tolerating my rants) in many distressing situations, especially that particularly challenging deal with a professor during my junior year. Also, congratulations again for your daughter's graduation.

As an old saying goes, "All work and no play makes Jack a dull boy". One cannot go through MIT without any fun. I would like to thank my Thai friends at MIT: undergrads, graduated undergrads, hope-to-graduate grads and graduated grads for hosting fun stuff for me to join over the last four years. For my N'N' Thai juniors, thank you for being friends with your "old man" and making my life not so lonely with your little pranks, games and dinners. I hope that we will have a chance to play Settlers of Catan again in the future. For my P'P' Thai seniors, thank you for being good big bros and big sis for me na krub. As an eldest son in my family, it was quite a unique experience back in my freshman year to be the youngest child in the TSMIT family. To tell you all the truth, I sometimes felt that I was being spoiled. Moreover, thank you for all your advice and support that you all have given me.

My thanks also extends to other people in Cogmac group. Stefie, thank you for being my first ever UROP mentor at MIT and getting my interest in robots going. I wish I were more productive during that time, though. Congratulations on your graduation and the upcoming good news in October! Leo and Soroush, thank you for being my office mates for my first two years at Cogmac and for constantly coming down to check my progress in writing this thesis! George, thank you very much for your work in Star Graph. Now, this thesis is probably your first citation. Also, thanks to Stefie and Matt for proof-reading my thesis. It was definitely a great help! For other group members, thank you for letting me be a part of your group and chatting with me whenever I try to procrastinate. The last forgotten member in the group that I would like to thank is Trisk, the old red robot. Since it is my inspiration to go to graduate school, I wish Trisk were still up and well and I am sure I will miss it a lot.

My undergrad friends are also a big part of my life at MIT. Anh, thank you for going through many hells with me in 6.170 and other classes. It has been indeed a long time since the time we became friends in Professor K's freshman seminar. Thank you for digging me out of lab and letting me be your friend. There are many others I would like to mention but the page number is limited, so I will just give you all a collective thanks for all the fun, chats, experience and insights that we all shared at MIT, but be assured that you all have my sincerest gratitude.

Last but not least, I would like to thank MIT for all the technical and life experience you have given to me over the last four years. The fact that I have learned a lot, get inspired,

and met great people that I have thanked so far in this acknowledgement is an enough reason to appreciate my time here despite all the chaos and sleeplessness.

I did say that this acknowledgement is a "farewell" note to everyone. However, this is definitely not a permanent farewell, and I hope very much that I will be able to meet with you guys again in the future and talk about those "good old days" in "hell."

Thank you!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The aim of this thesis is learning pedestrian movement patterns for the purpose of analyzing retail environments. One might wonder why patterns in pedestrian movements are an interesting research topic, because moving around and interacting with other people are such ordinary activities. On the other hand, one cans quickly realize that understanding and reproducing their general patterns is not an easy task. There have been studies of such pedestrian movement patterns, both human and animal, in previous research efforts.

Likewise, retail environments such as convenient stores, banks, and supermarkets are frequent destinations for most people. However, managers and investors are interested in the environment within their establishment and how their customers interact with and within it. Such analysis is of interest because it can provide business and managerial insights to improve customers' experience and sales performance. Nevertheless, an impartial quantitative approach to the problem is generally not available, forcing the decisions to be based on case studies and trial-and-error.

It is the aim of this thesis to simultaneously explore both areas by providing a quantitative measurement of the retail environment's business performance based on the pedestrian movement of its customers. The experimental work in this thesis is based on my contribution as a part of "Data-driven Architecture Project" in the Cognitive Machine Group at MIT Media Lab.

1.1 Learning Human Pedestrian Movement

In order for humans to do daily activities, it is almost always required for them to walk or travel around. However, having a machine generalize pedestrian movement patterns from a given video sequence is still an unsolved problem.

Ideally, machines should be able to produce and visualize believable, generalized pedestrian paths in a given environment with specific data. If successful, this technology would serve as a useful tool for visualization and analysis of human pedestrian behavior, for example, in the gaming industry and social science research projects. Many issues, however, will have to be addressed before the research problem is solved:

1. How can the "believability" of any given set of paths be measured? In other words, how can one measure the likelihood that given paths are produced by humans under a given context? If two sets of paths are compared, what should be the criteria in determining which is more believable?

2. How should the machines be trained to recognize patterns in given set of paths and generate paths under the same patterns? What should be the balance between constraints and flexibility in the training process?

3. How can goal-directed movements (such as walking to the kitchen to cook dinner) be included in the developed models?

In short, the major issues include how to convincingly determine if the simulated patterns are believable as human paths and how such believable simulation can be produced. This thesis will attempt to answer these questions within the context of retail environment settings, specifically banking premises.

1.2 Analysis of Retail Environment

The stores' environment influences customers' impression and experience which, in turn, potentially determine the stores' business performance because of the images that the environment can form in customers' minds [20]. Therefore, in order to optimize customers' experience and business performance within a retail environment, its planners and opera-

tors might be interested in measuring and improving store performance in several aspects, for example:

1. Is the current number of cashiers optimal?
2. Do product locations encourage customers to shop around?
3. Are directions in the store clear enough? Is it hard for customers to find their desired items?
4. Is there too little space in the store? Otherwise, can the use of the store's space be further optimized?
5. How should sections within the store be rearranged?
6. If there is a plan to open another branch of the store, what architectural specifications should the new store have?

Although these problems are usually considered by the management team, it is hard to rigorously argue for one possible option over others without systematic research to support the results [2]. While previous experience is helpful in giving directions for such decisions, such evaluation cannot be impartially quantified.

The environmental optimization issues can be categorized into two aspects: architectural and managerial. While it is easy to adjust managerial problems such as problems 1, 2 and 3, designing solutions for problems 4, 5, and 6 is much harder because these problems might involve altering the architectural setting of the premise. Because reconstruction or refurbishment is too difficult and costly to attempt trial-and-error approaches, architectural features of the business premises is usually treated as fixed in the business analysis unless convincing reasons can be suggested.

1.3 Benefits of Human Pedestrian Movement Pattern Research in Retail Environment Analysis

As described in Section 1.2, managers need credible reasoning or evidence to support costly decisions such as the alteration of their premises' architecture. It is the vision of this thesis

that the study of human pedestrian movement could fill the role of an impartial advisor to such decisions. In fact, both research areas complement each other's need. While the retail environment analysis becomes a powerful quantitative tool, the stores serve as an observation ground for pedestrian movement and pattern recognition researchers.

1.4 Objective of the Study

As a step towards the final goal of developing a system which can reliably analyze a retail environment, it is necessary to study and extract patterns from pedestrian paths in real dataset. While pattern recognition techniques require quantifiable metrics to guide the training process, it is not obvious how to quantify pedestrian paths nor how to compare the quality of path data sets. As it will be discussed in Chapter 2, the criteria varies from one researcher to another and heavily depends on the assumptions of how the paths are generated.

Therefore, it is the main focus of this thesis to explore such this quantitative metric and implement them as part of a training program which optimizes the simulation model and visualizes the improved results. Ideally, the metric will fully reflect what humans consider to be "believable" and can be flexibly applied to paths of different size, direction and number.

1.5 Scope of the Study and Limitation

Due to time constraints in developing rich interactive behavior between agents, this study focuses on the agents' movements within the environment, regardless of the specific purposes behind these trajectories. In other words, only the locations of agents during the simulation will be analyzed and modelled in this study.

The training will concentrate on pedestrian movement of customers within the retail environment because they are usually the target group for behavior analysis and also the group of people who generate the majority of the paths. In addition, because the aspect of interaction is disregarded, the role of staff members as path generators will be omitted from the study.

In summary, this thesis describes how customer agents can be trained so that they can portray patterns of human pedestrian movement in a retail environment context. Chapter

2 reviews previous research on the topics and select the framework of techniques that this thesis will pursue. Chapter 3 discusses and justifies steps towards development of the pedestrian movement's simulation training system, as well as experimental details. Chapter 4 discusses results and implications of the experiments. Finally, Chapter 5 summarizes my contribution in this thesis, and gives suggestions for future research directions.

Chapter 2

Review of the Literature

2.1 Previous Work in Retail Analysis

It has been claimed in the literature that the retail environment is important in influencing the sales performance of the stores. For example, Martineau claims in [20] that the performance of a retail environment depends on what customers perceive as the store's personality, and the environment is one of the key factors in impressing the customers. Despite the vast number of published research papers in the area, it is suggested that management teams of retail stores do not put much effort into systematic research to guide their business decisions on refurbishing their stores and training their staff [2].

The retail environment consists of many factors which are potential candidates as independent variables in research studies to investigate their effects on consumers' experience and sales performance. These factors include lighting, music, shelf space, staff, architectural design etc. Researchers have previously attempted to study the effects of some or all of the factors in various settings. For example, Baker concludes in [2] that the quality of interaction between customers and staff clearly has positive correlation with customers' price acceptability. On the other hand, other "ambient" and "design" factors show less conclusive evidence of their influences on consumers. A more formal model and detailed factors are studied in her later work [3].

While there have been many areas that have been explored in the field, there are still many factors within the retail environment that still need further investigation. Examples of problems suggested by Turley & Milliman in [28] include the effect of exterior appearance and interior design of stores on sales performance, the crowding effect caused by stores'

architectural layout, and customers' activities in environments that require privacy. Therefore, the study of architectural and layout aspects, which is the focus of this thesis, still has potential for growth.

While it is shown in [28] that previous research attempts in the retail environment analysis depend on human subjects to draw conclusions about their hypotheses, these researches are limited by the huge overhead cost of human-subject experiments. Setting up meetings with subjects and preparing experimental scenarios that human subjects can interact with can be very time-consuming. Moreover, Baker admits in [3] that stores cannot afford to spend much time and manpower to accommodate lengthy, complicated field experiments. Therefore, researchers like Baker are limited to methodological techniques with low cost and easy setups like "videotape methodology," in which subjects are simply shown videotapes of simulated setups [11, 2, 1, 3]. This prevents researchers from pursuing more ambitious but complicated approaches to the research problem. Moreover, because the main focus of previous work is analyzing customers' behavior in existing establishments and in simple experimental setups, it is hard for researchers to discover radically new paradigms or business models. Lastly, observation of abnormal situations such as emergencies is rare, making it hard for researchers to make generalizations and analyze these cases critically.

2.2 Pedestrian Movement Analysis

In previous literature about analysis and simulation of pedestrian movement, there have been many ways to characterize human pedestrian paths. These paths can be generated by random walks, act as a part of flows of entities in the scene, or be influenced by individual entities' decisions.

2.2.1 Paths as Random Walk

One of the simplest models of pedestrian paths is that they are generated using randomization. Randomization has the advantage that specific understanding about the observed subjects' goals and behavior is not necessary. An example of such a model is a random walk.

While Turchin has formulated a mathematically rigorous formula for a random walk in [24], the derived differential equation is so complicated that it does not have much practical

use. Therefore, a simpler but less sophisticated alternative called a "correlated random walk" is used by many researchers instead. A correlated random walk assumes that the observed entities randomly choose their turning angle and the distance they would travel in that direction at any given time. The probability distribution of the turning angles and the traveling distance can be arbitrary. However, most of the literature assumes that the these quantities follow either a normal or uniform distribution for simplicity [8]. This paradigm has been used to analyze paths of insects and trajectories of particles in diffusion.

The success of the correlated random walk is usually measured as dispersement from the starting point of observation. In other words, it measures how much the observed entities have dispersed from the center of a crowd. This can be calculated as either mean-squared dispersement distance (MSDD) [16, 21] or mean dispersement distance (MDD) [6, 8]. While the calculation of MSDD is simple and straight-forward, it does not accurately measure the observed dispersement which is the MDD. As a result, how to calculate MDD was a topic of discussion in the literature until an acceptable solution was proposed in [8].

The advantage of this paradigm is its simplicity in describing the collective behavior of the path generators. However, the dispersement distance alone cannot account for many other basic aspects of pedestrian paths, such as speed and overall direction of the observed entities. In addition, for human pedestrian paths, the assumption that all entities make decisions about their movement randomly is invalid. Unlike insects and particles whose goals and decision-making processes are either unknown or non-existent, human actions are usually goal-directed. Therefore, it is reasonable to assume that human paths should reflect human underlying goals (if there are any) and should not look like paths whose turning angles and distance are totally randomized.

2.2.2 Paths as a Part of Flow

Characterizing paths collectively as flows of entities within the scene is one of the popular approaches in pedestrian movement analysis. The major assumption of this approach is that observed entities move according to a specific function of movement flows. This assumption is valid when the observed scene is so crowded that small aberrations in individual paths no longer affect the collective characteristics of a flow. At the same time, paths in a flow have general patterns that can be extracted. Examples include hallway traffic and emergency evacuation [27, 15, 14]. The analytical results under this approach are usually expressed

as continuous functions and differential equations that describe properties of the observed paths, such as population density and rate of change in speed etc.

The benefit of this technique is its power to generalize a set of complex paths into a compact representation. Once continuous functions are found, it is no longer necessary for the simulation of the studied scenes to focus on small variations in individual paths.

One of the major disadvantages of this approach is that it can over-generalize how paths should flow in any given location. Even though it can be argued that each area can consist of multiple flows of entities, it is usually hard to specify the number of flows necessary and the configuration of each flow to fully characterize a given set of paths. As a result, finding the right formulae and parameters to characterize these flows can be extremely challenging in complex pedestrian scenes. This is especially true for environments with large open space in which pedestrians are free to move around in any direction.

In particular, previous works which use the flow assumption usually have only a few flows in the scene, and the existence of such flows can be easily justified. For example, it can be expected that the evacuation scenes like those in [15] have only one direction of path flow in any given location. In other work such as [17] and [27] which observe and simulate colliding flows of humans, the number of these flows is limited to two or a small number.

Therefore, while the flow assumption is powerful in its generalization of collective patterns of paths, its solution is not universal. If the pedestrian path analysis only uses this assumption, the scope of suitable scenarios that can be analyzed is confined to scenes that have uncomplicated path behaviors. Therefore, some previous work uses other criteria along with the flow assumption to help refine the performance of the path analyzer. For example, it is demonstrated in [15] that by using interactions with other entities in the same scene, the flows can "self-organize" to improve the quality of path simulation when two flows of people move in an opposite direction.

2.2.3 Paths as a Result of Individual Entities' Decision

The main idea of this assumption is that all paths are goal-directed and entities' movement are directed by their individual decisions. The "decision-making" and "independent" qualities make this assumption equivalent to that of a modelling technique called "agent-based modelling.

Agent-based Modelling

Agent-based modelling is a simulation modeling technique which employs autonomous decision-making entities called "agents" [5]. This field has a wide range of applications in computer science, social science and business planning. According to Bonabeau in [5], the major types of applications of agent-based modelling can classified into four areas: flow management [27, 13, 17], market simulation [19], organization management and simulation of diffusion of information among agents [12].

According to Bonabeau in [5], there are three major advantages to agent-based modelling. First of all, the technique can generate emergent phenomenon from the "bottom-up". By allowing agents to interact with one another, there are endless combinations of behavior that the "system" can exhibit, using only limited, local decision-making in agents.

This leads to the second benefit which is the availability of a complete description of the system. Because all kinds of agent behavior are the result of the states within agents or inter-agent interaction, these behaviors can always be explained by using agents' local information. Lastly, because all decisions of agents are made locally, it is easy to make changes to the system, such as adding or removing agents from the environment.

At the same time, Bonabeau also points out issues with agent-based modelling. The first issue is that the agent-based modelling cannot be applied universally because the models work best when there is the right amount of detail and configuration in the model for agents to exhibit certain behaviors. The other issue is that some quantities, such as agents' preferences, are hard to quantify [5].

Because emergent behavior is possible, it is implied that complex behavior can be broken down into smaller, simpler problems. Therefore, agent-based modeling opens an opportunity to simplify computational problems. Many decisions are too complex for a single entity to make, or too difficult for human to preconfigure all the parameters. For instance, the human path analysis, which is this thesis' focus, is a complex non-homogeneous problem, because human paths are generated by multiple humans whose goal-directed intentions are different and hidden from observers. Agent-based modelling has the potential to simplify these problems because highly complex scenarios can be decomposed into many simple decision-making processes. Employing multiple independent agents in the simulation environment opens an opportunity to make many inexpensive, simple decisions based on local information

and parameters, rather than making a costly, intricate global decision.

Agents' behaviors can be influenced by interaction with other agents, which changes their internal states, and agents' goals are to minimize some "objective cost function," or maximizes some "objective score function" [4]. These functions can be a combination of small factors chosen based on researchers' objectives and assumptions about the implemented agents. For example, Berrou et al use emotional states like "inconvenience", "frustration" and "discomfort" in their pedestrian path modelling [4]. Turner and Penn assume Gibson's ecological theory of perception and Gibson's principle of affordance as factors that motivate their agents to make pedestrian choices and implement their agents with "vision" capacity. As a result, agents in Turner and Penn's work choose their paths in order to maximize the area in the environment that they have viewed [29].

Therefore, the major challenge in agent modelling is choosing the most suitable objective function for the application and optimizing the parameters to tweak the agents' behavior (such as Turner & Penn's work in [31]).

Application of Agent-based Modelling in Path Analysis

There has been some previous work in the agent-based modelling for the retail environment analysis problem. For example, Casti demonstrates that capabilities of agents in showing customer density in a supermarket environment in [9]. While the purpose of this work is similar to this thesis's ultimate goal, which is providing tools for business planners to design retail environments, this thesis's focus is a banking environment, which will be fully discussed in Chapter 3. In addition, there have been many papers that advocate agent-based modelling as a tool for path simulators and general architectural analysis.

Earlier related work in path analysis concentrates on developing agent behavior and decision-making process so that the traversed paths of agents match the existing data's patterns. For example, Turner and Penn's recreation of pedestrian movements within the Tate Gallery does not mention using agents to make further generalizations beyond the gallery's environment [30, 31].

On the contrary, more recent papers have begun to suggest the use of agent-based modelling for generalized performance prediction. In [7], Burkhard, Bischof and Herzog suggest that the use of Massive, which is an agent-based simulation software, for analysis of communication of entities within a building. Likewise, Narahara shows that the emergent

behavior created by agents can be used in evaluation of architecture [23, 22]. Because the purpose of both authors is to introduce the potentials of agent-based simulation in observing people's behaviors within certain buildings and then making an evaluation of how the environment affects the agents, the results of both papers are not rigorously evaluated. However, this is sufficient to suggest the potential of agent-based modelling in the analysis of a retail environment in terms of space usage and interactions between people.

In addition to relying on the emergent behavior of agent-based modelling, performance of pedestrian path generation can be boosted by providing a priori information to the model. For example, Courty & Corpetti have used flow analysis as noted in subsection 2.2.2 to boost the quality of their generated paths [10].

2.3 Implications from the Literature about Analysis of Human Pedestrian Movement in Retail Environment

From the literature review discussion in Sections 2.1 and 2.2, the most important insights that guided this thesis can be summarized as followed:

1. It is mentioned in Section 2.1 that the current approach in the retail environment analysis research only allows researchers to propose minor adjustment to the business practice with quantitative justification because of the cost of human experiments.

2. Agent-based modelling is capable of creating emergent behaviors and has potential as a technique for making predictions.

3. Agents' decisions can use perceptual input like vision like in Turner's papers [30, 31, 29] or use emotional states like Berrou's work [4].

4. Agent's parameters can be tweaked and optimized so that the agent exhibits more accurate behaviors according to certain objective cost/score functions.

Therefore, in order to save cost on pedestrian path analysis in the retail environment, this thesis will pursue an agent-based modelling approach because of its potential to make insightful predictions of the retail environment without the huge overhead cost of human experiments. The agents in the simulation will be given vision capabilities like Turner's agents in [30] to alllow them to see obstacles and estimate distances.

Lastly, to avoid making too much assumption about agents' behavior, this thesis will introduce a new objective function formula, to be fully discussed in subsection 3.3.4 that relies on comparison of pedestrian paths' coordinates to a reference dataset and given path data. This decision will allow agents to be trained and configured so that their movement patterns converge to that of entities in the real-life retail environment.

Chapter 3

Methodology of the Research

3.1 Raw Data

The raw data of this thesis consists of video sequences that were collected from a branch of a bank in South Carolina on July 7, 2009 from 12PM to 1PM. Multiple cameras were installed to observe activities in the bank's hallway and lobby. Because each camera can only capture a small portion within the whole bank, most of the pedestrian paths cannot be entirely captured by only one camera. In each time step of 0.067 seconds (or 15 frames per second), each camera would capture one image within its field of view. Each image frame was later analyzed to track people in the image and record their coordinates. Chains of these coordinates are then encapsulated into paths. For customers' privacy, only visual data was recorded in the video sequences. There was also no labeling of whether the captured paths belong to staff members or customers.

To use the fragmented path data, smaller path segments recorded by multiple cameras had to be merged in order to reconstruct the original pedestrian paths. The process involves identifying path segments in different cameras as a part of the same longer path. The final result of the merging process should be paths that would have been observed in a single camera if the camera were to capture the whole scene at once.

However, because the path merging process is complicated and time-consuming, and is not a focus of this thesis, only a one-hour sequence whose paths were manually merged was used in this thesis' experiment. During this time block, there are 32 different tracks that traverse the banking environment.

Figure 3-1: **Screenshot of raw training paths within the simulation environment**: The screenshot captures paths during one-hour tracks within the banking environment, which are described in detail in Section 3.2.1.

3.2 Agent Training and Optimization

3.2.1 Specification of Simulated Environment

The simulated environment is modelled based on the bank where the video sequences were recorded. The simulated building is 9456 millimeters wide and 28271 millimeters long, which is spacious compared to the size of simulated agents (which are simulated as small rectangular-shaped entities).

The sections in the simulated building which are most relevant to the goal of agents are an ATM machine on the left side and the teller's counter on the right side of the environment. Both of these areas have their own queuing location for customer agents to simulate a real-life queuing procedure. Agents enter and exit from the building through the two designated entrances. Other components with minor importance to the agents' goals include three office rooms and the main hallway area, which provide a spacious area for agents to move in.

More snapshots can be seen in Figures B-1, B-2, B-3, B-4 and B-5. Agents are drawn

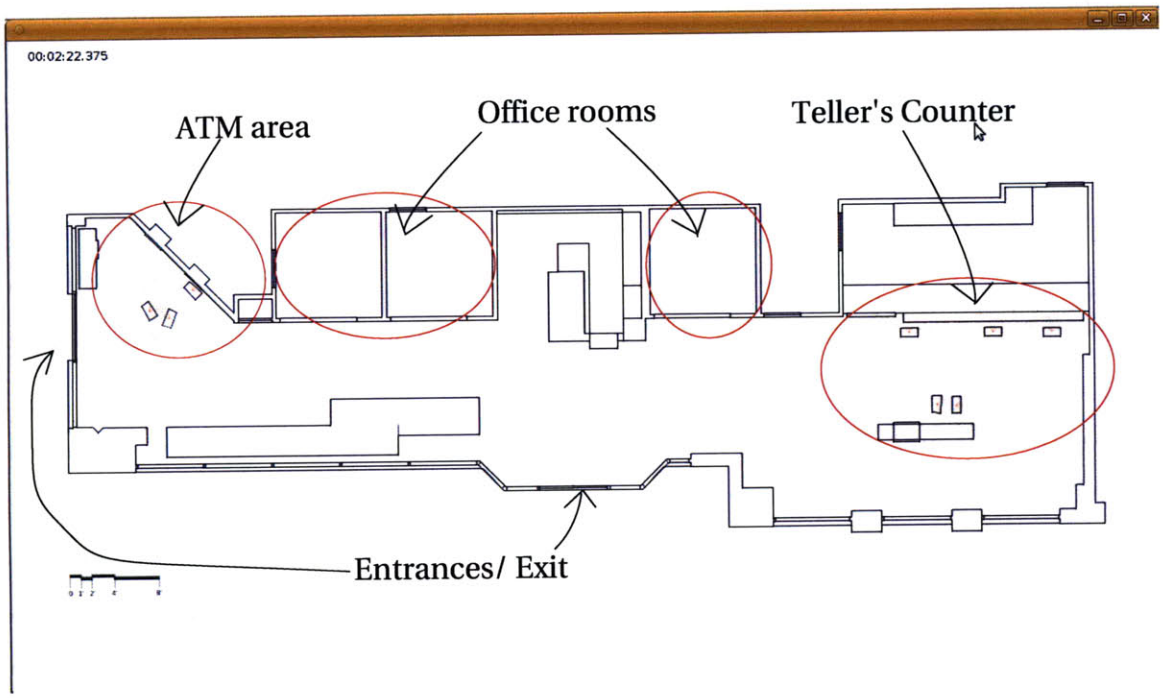


Figure 3-2: **A snapshot of simulated bank environment:** The simulated bank model consists of several major areas including ATM machines, teller's counter, office rooms and the main lobby/hallway area. Customer agents (shown in the simulation as small, moving rectangles) enter and leave the building via the two main entrances to the building.

in small rectangles. Individual agent's red lines show the agent's field of view at any given moment. Blue lines portray the paths that agents have traversed in the environment. Once agents have exited the building, these lines are removed.

3.2.2 Overview of Implemented Agent

The agent used in this thesis is a collaboration with Rony Kubat, a PhD candidate in the Cognitive Machine Group. My major responsibility in the agent's development was adapting it so that it could be trained flexibly, as well as making minor adjustments to the agent's behavior.

The agent acts as "customers" in the simulated environment, which is a bank in this case. These agents are goal-directed with two variations of goals: making transaction from the ATM and contacting tellers. Agents are spawned, under uniform distribution, from each entrance to complete their goals before leaving the building through either of the available exits. The transaction time of each of the agents is randomly sampled under a normal distribution and is independent of other agents' choices.

To traverse from one location to another in the scene, such as from an entrance to an ATM machine, agents generate paths that are directed by a map called a "Navigation Graph". Nodes in the Navigation Graph are arranged in a triangular grid except for locations where there are objects or obstacles in the scene. Edges that do not cross into the obstacles are then added to connect adjacent nodes. By using the Navigation Graph, the basic customer agents choose their paths using the shortest routes to their destination.

As customers, the agents are assigned to queue for their service if there are no ATM machines or tellers available. They will seek either the last agent in the queue or the queue-starting location if they are the first one seeking the queue. If an agent somehow loses its way during its queuing or after it exits the queue, it returns to the end of the line. After their tasks are completed, agents randomly choose an exit to leave the environment and find the shortest path to reach it.

Other than following the Navigation Graph and queuing procedure, agents try to move and turn to avoid obstacles and other agents. In order to do so, agents use a LIDAR (vision) sensor to help them perceive their surroundings.

For simplicity, agents are portrayed in the simulation as moving rectangles of size 460 $\times$ 240 square millimeters.

In short, implemented agents:

1. Act as customers and choose the shortest unobstructed path to reach their destinations.

2. Are goal-directed. Their goals are randomly chosen between seeking teller assistance and completing transactions with ATM machines.

3. Form a queue if their goals cannot be immediately completed

4. Exit from the environment after they accomplish their goals

3.2.3 Details of Agent Behavior

This section will discuss the behaviors described in Section 3.2.2 in detail, including agents' internal states and mathematical formulae which direct agents' behaviors.

The behavior of the implemented agent can be separated into two main layers: a layer for high-level decision-making and a layer for lower-level navigation. Examples of agents' actions that the higher-level layer would be responsible for include finding tellers, queuing for service, transacting and exiting the bank etc. In order to accomplish the goal set by the higher level, decisions in the lower level direct agents' navigation through speed and angular velocity. At each level, there are many possible states (as shown in Figure 3-3).

High-level Decision-making Layer

The states in the higher layer can be classified into six main categories. A summary of the relationships between these states can be found in Figure 3-4.

1. **Seeking Teller**: This is the first state of an agent after it is initialized. The goal of an agent in this state is to seek and navigate to the target's location. Therefore, this state involves a lot of low-level navigation which will be discussed in the "low-level decision making" section. Once the target is found, the agent will enter "transacting" state if no other agents are occupying the target. Otherwise, the agent will enter "seeking queue" state.

2. **Seeking Queue**: If an agent cannot transact right away, the agent will seek the end of the queue. The end of the queue is found by inquiring a Queue object for the

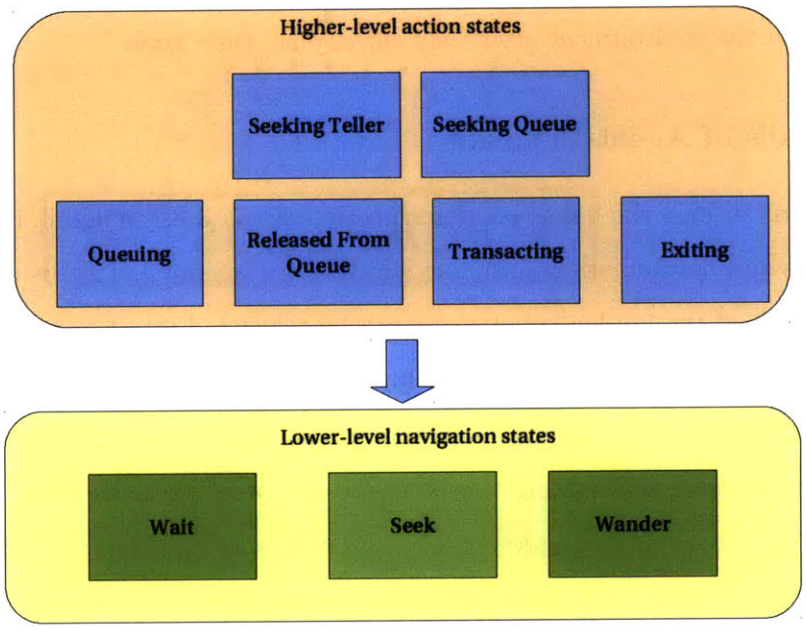


Figure 3-3: **States in higher and lower levels of agent behavior**: Once an agent makes a higher decision by entering a state in the higher level, the decision will be fulfilled by the lower-level decision which directly governs the speed and angular velocity of agents. Like the higher-level decision, there are also different states that lower-level behavior can be in. Details of decisions within the two layers can be found in Figures 3-4 and 3-5.

location of the starting point or the last agent in the line. Once the agent has moved within the threshold distance from the end of queue, it will enter the "queuing" state.

3. **Queuing**: In this state, a queuing agent will choose the last agent in the line before itself as the target that it will follow until it reaches the starting point of a queue. By "following," the agent will try to align with the target agent while maintaining a specific interpersonal distance. Once the agent is at the start of the queue and its ultimate target (like "tellers" or "ATM") is visible and unoccupied, the agent will enter the "released from queue" state.

4. **Released From Queue**: This state simply releases an agent from the queue it is in. Because this state is reached only after the agent has exited its queue and finds that its ultimate target (tellers or ATM) is unoccupied, it will be declared "lost" if the ultimate target is not available, visible or obstructed so that the agent cannot reach the target within a limited amount of time. If it is lost, the agent will reenter the "seeking queue" state. Otherwise, it will move on to the "transacting" state.

5. **Transacting**: In this state, the agent will occupy its target for a variable amount of time. The choice of time is normally distributed between a minimum and a maximum time allowed. After the time limit expires, the agent will reach the "exiting" state.

6. **Exiting**: After the agent has reached this state, an exit will be chosen as the agent's target. The choice of the exit is random regardless of the entrance in which the agent was originally spawned. The location of these exits are queried from the simulation environment. Once the agent has reached its intended exit, it will be "absorbed" and disappear from the environment.

Low-level Decision-making Layer

The aim of the low-level decision-making layer is to help the higher level reach its decision's goal. For example, an agent must find a way from the entrance to an ATM in order to satisfy the "seek tellers" state discussed earlier. Low-level decision-making uses information from sensors such as LIDAR and other information from the simulation environment to decide on the agent's speed, angular velocity and orientation.

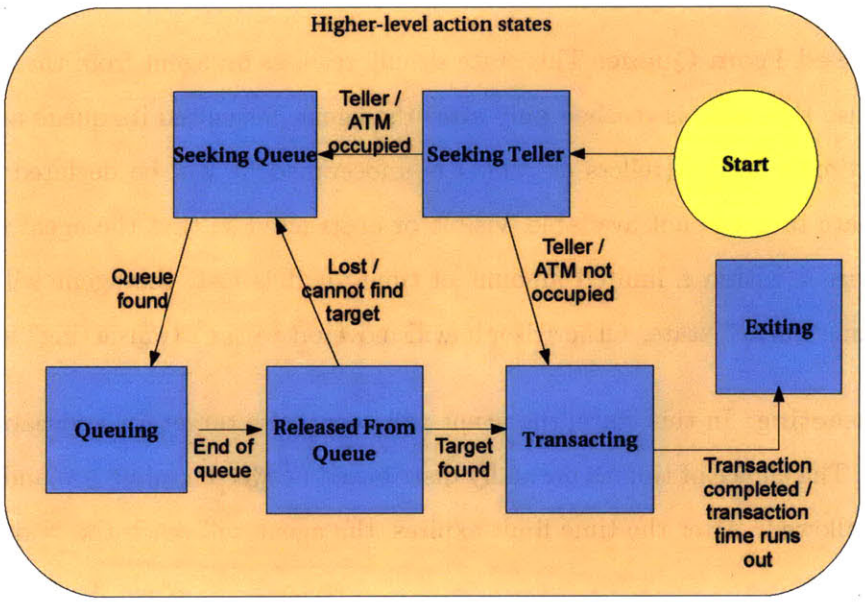


Figure 3-4: **Higher-level decision-making in agent**: The state chain model shows all possible higher-level states that an agent can be in under the context of a banking environment. The states in this level and their transitions are similar to how humans decide their courses of action.

For navigation purpose, the simulation maps the environment with nodes arranged in a triangular grid. The simulation eliminates nodes that overlap with the environment's obstacles (such as tables, counters etc.). The remaining nodes are then connected to their nearest neighbors as long as the edges do not cross into obstacles. This results in a graph called a "Navigation Graph." Nodes in Navigation Graph can be used as immediate targets that lead navigating agents to their intended destinations. Anyhow, there are three different states that the low-level decision-making can be in.

1. **Wander**: This is the first state that an agent enters. It is assumed in this state that the agent either does not have a target during the current time step or does not "see" the target. The purpose of this state is to let the agent wander around in hope that it will encounter its target. The agent's priority in this state is to move towards the direction with more open space. If the agent's target is visible, the agent's state switch to a more goal-directed "seek" state.

   More specifically, in "wander" state, the agent's torque (τ) and force (F) are governed by the following equations.

$$F = \frac{1}{\|\theta\|} \sum_{\theta} d_\theta + \frac{M - M_{min}}{M_{min}} \tag{3.1}$$

$$\tau = \frac{1}{\sum d_\theta} \sum_{\theta} d_\theta \theta + \frac{I - I_{min}}{I_{min}} \tag{3.2}$$

   where F = the force applied to agents
   τ = the torque applied to agents
   d_θ = the distance from the LIDAR center to the nearest obstacle in the direction of θ.
   θ = the angle from the agent's LIDAR sensor
   M = the agent's mass
   M_{min} = the minimal possible value for agent's mass
   I = the agent's moment of inertia
   I_{min} = the minimal possible value for agent's moment of inertia

2. **Seek**: If the target (or immediate targets like nodes in the Navigation Graph) is spotted in the LIDAR vision field, the agent becomes goal-oriented and move towards

that target. Therefore, the agent's speed and angular velocity will be set so that its orientation is directed towards the target. The agent switches to "wander" state if it loses track of the target.

The specific equations for force and torque in "seek" state at any time step are:

$$\tau = (1 + TC) \times (c \times \theta_{target}) \tag{3.3}$$

If the agent's target at the current time step is the final destination,

$$F = \max(TC \times \cos(\theta_{target}) \times d_{target} + (1 - TC) \times \frac{1}{\|\theta\|} \sum_{\theta} d_\theta, 0) \tag{3.4}$$

Otherwise,

$$F = \frac{1}{\|\theta\|} \sum_{\theta} d_\theta \times \cos(\theta_{target}) \tag{3.5}$$

where

	F	=	the force applied to agents
	τ	=	the torque applied to agents
	d_θ	=	the distance from the LIDAR center to the nearest obstacle in the direction of θ.
	d_{target}	=	the distance from agent to the current target
	θ	=	the angle from the agent's LIDAR sensor
	θ_{target}	=	the bearing angle towards the current target
	TC	=	a fuzzy boolean value which tells how close the agent is to its target
	c	=	some constant

3. **Wait**: This is a special state that is entered only when the simulation program specifically calls for the agent to "do nothing." In this state, the agent will stop moving and turning and wait for the time limit set by the method call to expire before switching back to either of the two regular states.

   The quantities of agent's force and torque are governed by the following equations.

$$F = -v \times D_v \times c_F \tag{3.6}$$

$$\tau = -\omega \times D_\omega + c_\tau \times \frac{1}{\sum d_\theta} \sum_\theta d_\theta \theta \tag{3.7}$$

where

F	=	the force applied to agents
τ	=	the torque applied to agents
v	=	the speed of agent
ω	=	the angular velocity
d_θ	=	the distance from the LIDAR center to the nearest obstacle in the direction of θ.
θ	=	the angle from the agent's LIDAR sensor
D_v	=	the speed damping
D_ω	=	the angular velocity damping
c_τ, c_F	=	some constant values

In addition to the basic behavior of each state, if the need arises for agents to avoid obstacles in any state, the agent will apply larger-than-normal torque for a rapid turn and smaller-than-normal force to slow down the agent. Under such situation, the calculated force and torque are readjusted as followed:

$$\tau_{avoid} = c_\tau \times \frac{1}{\sum d_\theta} \sum_\theta d_\theta \theta \tag{3.8}$$

$$F_{avoid} = c_F \times \frac{1}{\|\theta\|} \sum_\theta d_\theta \tag{3.9}$$

$$\tau_{final} = \tau_{avoid} \times P_{avoid} + \tau \times (1 - P_{avoid}) \tag{3.10}$$

$$F_{final} = F_{avoid} \times P_{avoid} + F \times (1 - P_{avoid}) \tag{3.11}$$

where

F_{avoid}	=	the portion of force applied to agents to avoid obstacles
τ_{avoid}	=	the portion of torque applied to agents to avoid obstacles
F_{final}	=	the adjusted force applied on agents
τ_{final}	=	the adjusted torque applied on agents
d_θ	=	the distance from the LIDAR center to the nearest obstacle in the direction of θ.
θ	=	the angle from the agent's LIDAR sensor
P_{avoid}	=	a Fuzzy-logic boolean deciding whether

			obstacle avoidance is necessary; $0 \leq P_{avoid} \leq 1$
	c_τ	=	some constant on avoidance torque such that $c_\tau > 1$
	c_F	=	some constant on avoidance force such that $0 < c_F < 1$

From the calculated force and torque on agents, the agents' speed and angular velocity during any given time step i can be calculated with the following equations:

$$v_i = v_{i-1} + t \times \frac{F_{i-1} - (v_{i-1} \times D_v)}{M} \tag{3.12}$$

$$\omega_i = \omega_{i-1} + t \times \frac{\tau_{i-1} - (\omega_{i-1} \times D_\omega)}{I} \tag{3.13}$$

where			
	v_i	=	agent's speed during time step i
	ω_i	=	agent's angular velocity during time step i
	F_i	=	force applied on agent during time step i
	τ_i	=	torque applied on agent during time step i
	D_v	=	the speed damping
	D_ω	=	the angular velocity damping
	M	=	the mass of agent
	I	=	the moment of inertia

The relationship between these low-level states based on the above discussion is illustrated in Figure 3-5.

3.2.4 Parameters Used in the Training Procedure

According to the agent's specification in Section 3.2.2, numerical parameters that are relevant to the implemented agent are:

- A bound on the distance between an agent and its target such that the agent has "reached" the target
- Distance between adjacent agents during queuing
- Threshold of the distance between agents to be classified as "near" or "too near"
- Threshold of the distance between an agent and obstacles (objects or other agents) to be regarded as "too close"

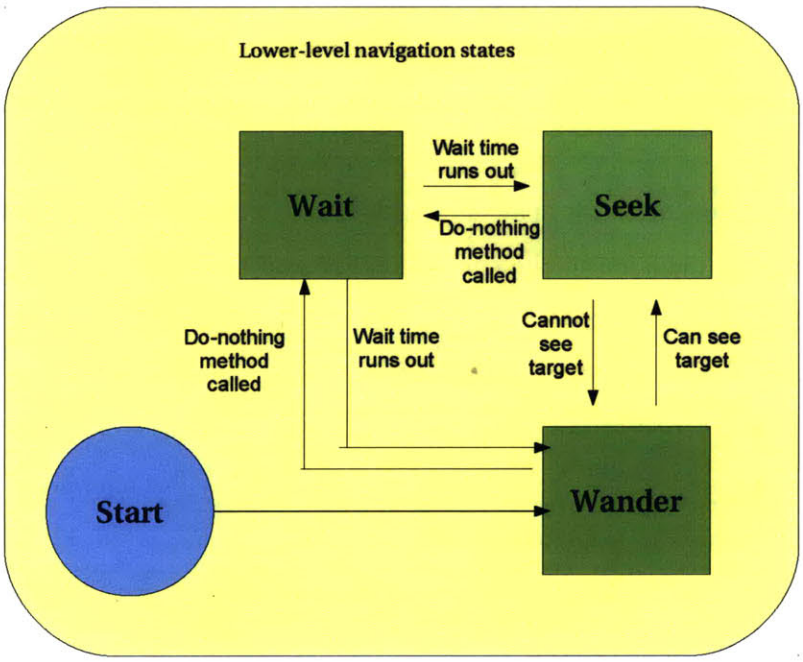


Figure 3-5: **Lower-level decision-making in agent:**

- Slow-down rate of an agent's velocity before the agent comes to a complete stop as it joins a queue
- The location of LIDAR/sonar emitter relative to each agent
- Mean and standard deviation of...
  - Mass of agents
  - Moment of inertia
  - Agent's speed damping
  - Agent's angular speed damping
  - Transaction time with ATM, tellers and other entities

Because there is no clear indication whether and how these parameters can directly influence the agent's choice of paths (except for delays and minor variations of movement), all relevant parameters are used in the optimization process. Out of the chosen 31 parameters, 6 parameters are treated as dummy parameters because their intended functionalities have no influence on pedestrian path choices in the current agent implementation. These

dummy parameters are included as a sanity check. Because these parameters cannot affect the pedestrian movement of agents in any way, it is reasonable to assume that the training process will never change their values as long as the training module is correctly implemented.

The randomization rates for spawning customers at both entrances to the simulation environment are not included in the simulation because the attribute is a property of the environment, not customers. One possible interpretation is that the spawning rate reflects the popularity or attractiveness of the bank. Although the agent-spawning rate does not normally affect agents' pedestrian paths, if the rate gets too high, it can cause overcrowding in the environment and blocking (as shown in Figure B-5). For simplicity, the spawning rate is currently ignored in the optimization process.

Full description and justification of the parameters used in this thesis can be found in Table A.2 in the appendix section.

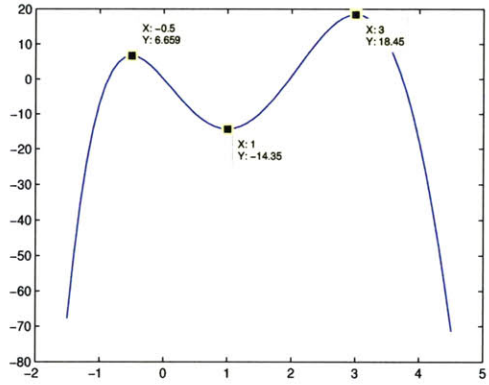
3.2.5 Hill-climbing Optimization

Hill-climbing is one common optimization technique due to its simplicity. It begins with some initial value for a parameter. At each iteration, the scores of the current setting and its neighboring values are computed. If any of the candidates score better than the current setting, the best one is chosen as the new value of the parameter, and the process is repeated. If the parameters do not change, the process is halted.

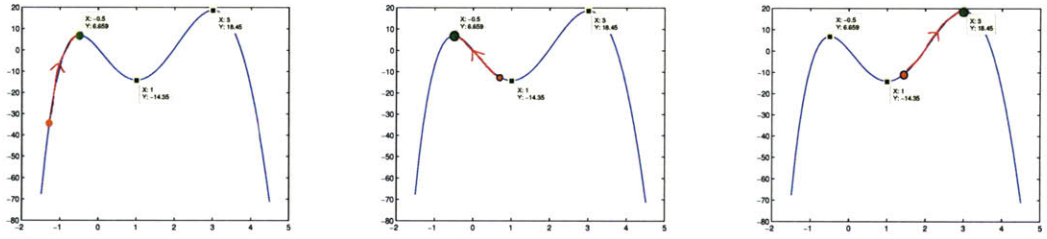
Performance Limitation Caused by Local Optimality

Consider a case where there is only one varying parameter. Figure 3-6 illustrates the procedure of hill-climbing described earlier in this section. The function in subfigure 3-6(a) has two local maxima at $x = -\frac{1}{2}$ and $x = 3$, which is also the global maximum under the shown domain. The application of hill-climbing on three different initialized values is shown in subfigures 3-6(b), 3-6(c) and 3-6(d).

Starting from any initialized values, the solution will converge to the best neighbor of the current value, before halting at the first local optimal value it reaches. Initialized at $x = -1$, the variable in subfigure 3-6(b) finds a solution by following its neighbors on its right, which produce much better score than those on the left side. The local maximum, however, is not the global maximum for the given function (which is $x = 3$). Because



(a) A simple objective function



(b) (c) (d)

Figure 3-6: **Hill-climbing technique:** Starting at any initialized values, variables under hill-climbing are adjusted towards their immediate neighbors with the best possible score from the objective function. Hill climbing will halt only when the current values of these variables are the best choices among the candidates.

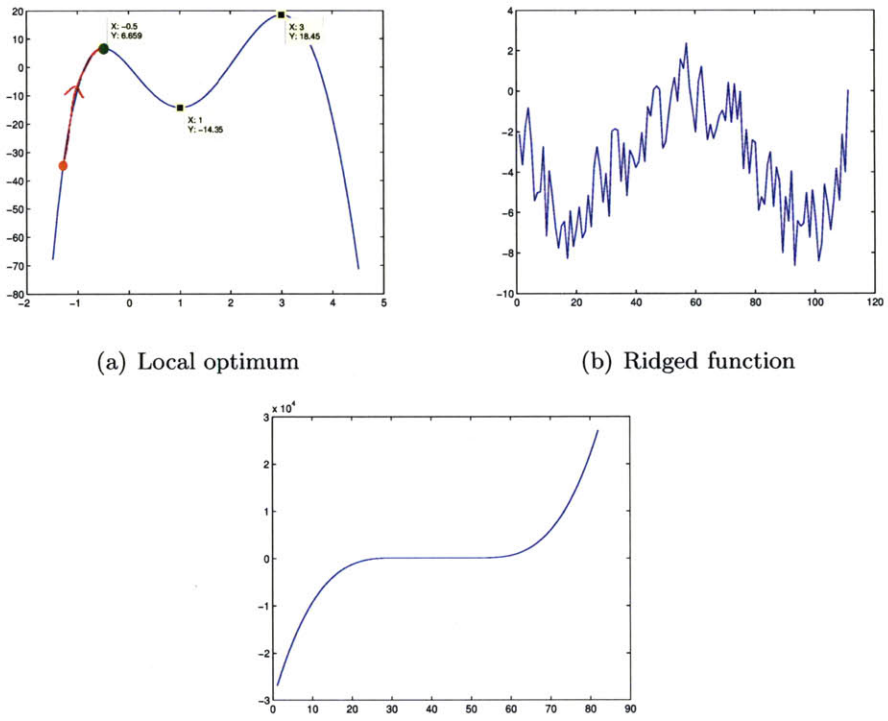
hill-climbing is a greedy algorithm, it can only guarantee that its solution will be a locally optimal value.

As a result, initialized values have great influence on the final outcome of the optimization. Although the initialized values of subfigures 3-6(c) and 3-6(d) are relatively close to each other, their optimized solutions are very different, with only the optimization in subfigure 3-6(d) reaching the real global maximum. This inability to transcend local optimality is one of the main issues that plague the performance of the hill-climbing technique.

Performance Issues Caused by Characteristics of the Objective Function

Other than the local optimality problem, hill-climbing faces potential issues as a result of irregular properties of the objective function. An example of these problems include:

1. **Functions with "ridged" shape or with high frequency**



(a) Local optimum

(b) Ridged function

(c) Flat function

Figure 3-7: **Major issues affecting hill-climbing's performance:** Inability to transcend local optimality and erratic optimization under irregular objective functions such as ridged and flat functions are the three major issues affecting hill-climbing's performance. They prevent the optimization from reaching the global optimum, or even a satisfactory local optimum.

The problem arises in this type of function because of "undersampling". In other words, if the resolution used in the search for "neighboring values" is not sufficiently fine, it is possible to smooth out the ridged function and miss the supposed local optimum, making the technique yield different optimized results depending on the choice of resolution despite having the same initialized value. There is no clear solution to the problem because if the shape of the objective function cannot be determined before the optimization process, there is no lower bound to ensure that undersampling will not happen.

2. **Functions with "flat" regions**

   If the optimizing variable reaches any region such that there is no gradient or the gradient is well below the allowed threshold, hill-climbing will assume that the local

optimum is reached. However, that might not be the case as shown in subfigure 3-7(c). In this subfigure, the region of $20 \leq x \leq 60$ is relatively flat and has no perceivable gradient. However, none of the values in the range is a local minimum. Therefore, if the initialized value is within this flat region, its solution under hill-climbing procedures will not be optimal.

Other Variations of Hill-climbing Technique

To overcome these limitations, other variations of the hill-climbing have been developed. Their main purpose is to reduce the risk of encountering bad local optimum and overcoming undesirable characteristics of the objective function. As discussed by Russell and Norvig in [25], examples of these variations include:

1. **Stochastic hill climbing** – The procedure randomly chooses candidates from the "uphill" side of neighboring values.

2. **First-choice hill climbing** – The procedure keeps randomizing candidates until it finds a value whose score is better than the current one.

3. **Random-restart hill climbing** – Have several runs of hill-climbing with different initialization during the same time. Halts when one of the runs finds a solution.

Although these inventions have advantages, their benefits are not critical in this thesis' application. One of the major reasons is that it is computationally expensive to compute the objective function used in this thesis. As will be fully elaborated in Section 3.3, the procedure to calculate the objective function requires running full simulations which can take hours to finish. Therefore, the strategy to randomize until a better solution is found (as in "first-choice hill climbing") is not feasible.

Likewise, although "stochastic hill climbing" reduces the risk of sticking to a single bad local optimum and produces satisfactory results in some areas, the performance of "stochastic hill climbing" heavily depends on the landscape of the objective function. In particular, it usually converges slower than the normal procedure if the landscape has steep improvement [25]. Since slow convergence means more calls to calculate the objective functions, the cost outweighs the benefit of Stochastic hill climbing.

The only technique among the alternatives that could have improved this thesis' optimization problem is "random-restart hill climbing". Its main idea opens up possibility

of parallelization to explore multiple initialization at a time. However, it was my decision to concentrate on proving the feasibility of the basic procedure which is discussed in this chapter before adopting parallel programming to boost the quality of its solution. Instead, this thesis will optimize its parameters using the basic procedure.

3.2.6 Adaptation of Hill-climbing Technique for Optimization of Multiple Parameters

Although the basic principle of the hill-climbing technique as described in Section 3.2.5 is straightforward, it does not discuss the case with more than one parameter in the optimization process.

Naive search through all possible combinations of values for the best neighbor in every iteration is not feasible with multiple parameters. There are 31 parameters in the optimization, and suppose that only 3 candidates are explored per parameter, in order to explore all combinations in each iteration, one needs to calculate the objective function for at least 3^{31} or approximately 618 trillion times. Therefore, a better solution is to optimize one variable at a time while holding each other variable constant.

It is known that this technique gives a locally optimal result with just one variable. However, for multiple-parameter cases, it is, in fact, not necessary to "fully" optimize each parameter at a time. The reason is that values of any adjusted parameter can be guaranteed to be optimal only if the values of all other parameters remain unchanged from the time these parameters are optimized. Although the performance metric will not worsen after a series of adjustments on other parameters, it is possible that these variables will no longer be optimal.

More importantly, because it is necessary to run simulations to calculate performance scores for each candidate parameter value, complete optimization can be extremely expensive, depending on the initial values. In the worst case, the program will not halt if there is no local optimum. (For example, the function $y = x^2$ has no maximum at all.)

In addition, by optimizing one variable at a time, the order of these variables becomes significantly important. Sorting parameters differently can lead to different local optima, and the first few variables used in the optimization are more important in determining the final outcomes than later variables. Since it is not usually practical to sort all parameters by their potential outcomes in advance, it is more prudent to avoid such overdependency

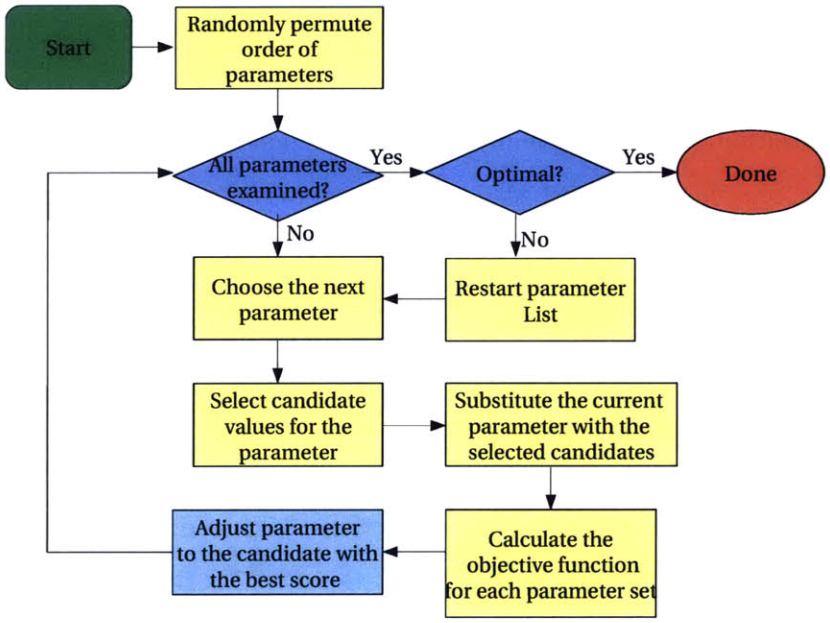


Figure 3-8: **A generalized flow chart of the basic hill-climbing with multiple parameters** – This flow chart illustrates the general procedure for hill-climbing with multiple parameters as described in Section 3.2.6. Ideally, the program will loop through a list of parameter until an optimal solution is found.

by spreading out the risk.

As a result, the approach that this thesis takes to maximize the performance score over multiple parameters is to slightly improve performance for each parameter in every iteration. In particular, each variable is increased, or decreased by a small amount, or remains the same. The value that produces the best score is chosen. Assuming that the performance metric is deterministic, the best of these scores will be at least as good as the score from the latest evaluation. The optimal value is reached when either (1) all parameters remain unchanged from the last iteration, or (2) the rise in performance score is below a certain threshold. The general idea of the described procedure is illustrated in Figure 3.2.6. For details of the actual implementation, refer to Section 3.4 and Algorithm 1 in the appendix section.

The choice of this "small amount" is a trade-off between quality of final results and computational cost. The optimization will have a better resolution when the change per iteration is small. However, the algorithm will have to iterate many more times to reach

the desired local optimum.

By doing this, all parameters have approximately equal chance to make an adjustment. Although it can be argued that the order of parameters is still relevant in this procedure of gradual optimization, by iterating many times such dependency will become less significant. This will save computational cost that might have been over-invested on any particular variable.

3.3 Measurement of Correlation Between Raw and Simulation-generated Path Data

From the discussion about hill-climbing in Sections 3.2.5 and 3.2.6, the algorithm needs an objective function to guide its optimization. Because this thesis optimization goal is to improve the quality of pedestrian paths produced by simulation agents, it is intuitive that the objective function should be calculated by using paths from the simulation as an input. The collected pedestrian tracks serve as a benchmark to determine this "quality," which is too subjective to precisely define. This section will elaborate how simulated paths can be compared to paths in the dataset and how to quantify them systematically to calculate the objective function for hill-climbing algorithm.

3.3.1 Assumptions

In order to justify the discussion in this section, the following assumptions must be accepted.

- **Assumption 1: The collected pedestrian paths are representative of how "good-quality" paths should be** – In other words, these paths are typicla of paths generated by humans. Otherwise, there is no benchmark to evaluate the quality of the simulation parameters even though the simulation results look qualitatively believable by human observers.

- **Assumption 2: Both simulated paths and dataset paths should increasingly exhibit greater similarity as more data on both sources are collected.** – Even though the exact shape and curvature of human paths can be arbitrary to some degree, there are usually some areas within the environment which humans use more often than others. Therefore, by observing for a long time, these patterns should be

detected from a set of paths. Likewise, if the simulation is properly optimized, its paths should collectively exhibit similar patterns to the data.

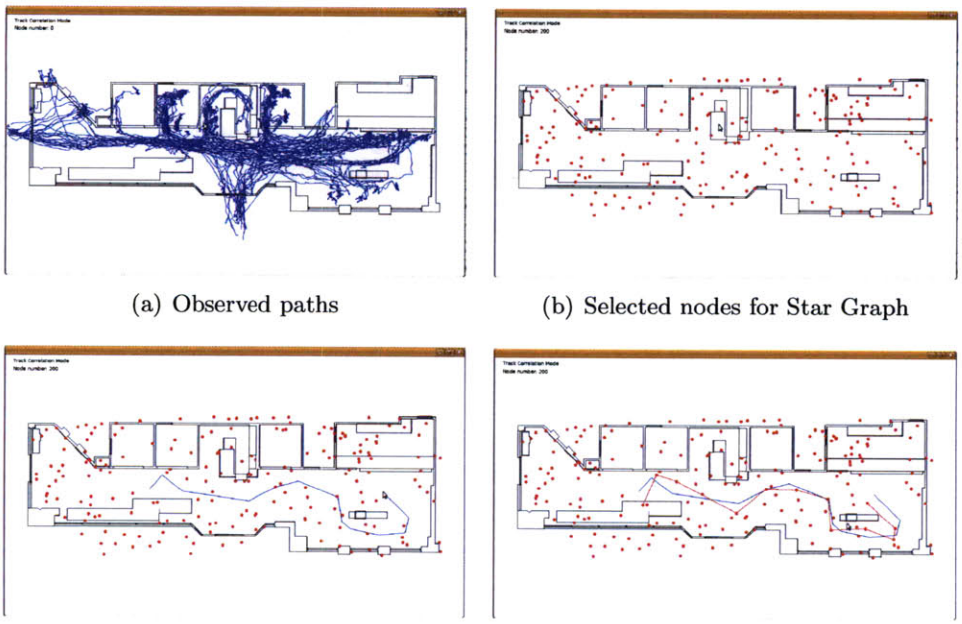
Regarding assumption 1, it is possible to argue that the available raw paths shown in Figure 3-1, which is only one hour long and has only 32 tracks, do not represent general human paths in the banking environment. However, even during only one hour, the paths in the figure has already traversed over the majority of the spaces in the environment. Moreover, the recorded time is from 12PM to 1PM which is a normal bank hour. Therefore, it is not unreasonable to assume that the observed paths are normal activities of banks during working hours. Although these paths are not a perfect set of data, assumption 1 should hold for this data and this thesis' experiments.

3.3.2 Graph Representation of Collected Paths

To perfectly reproduce a continuous path, one must specify infinitely many data points of agents' locations. Even for paths which are collected from a simulation whose time steps discretize paths into intervals of path segments, there are still many more data points than what is necessary to represent these paths. Considering memory and computational cost required for storing and analyzing these coordinates, it is best to simplify the complexity of human path representation.

A convenient alternative is splitting these paths into a chain of connections between a small set of fixed nodes in the observed environment. By such discretization, a collection of pedestrian paths can instead be analyzed as a graph whose edge weight is the number of times that a path connects any pair of nodes of the graph. Therefore, the amount of required data points to characterize a path can reduce from infinitely many points to just a small number of points.

A major decision that comes with the graph representation is how to choose the best connections to characterize the continuous paths. Developed by George Shaw, a member of Cognitive Machine Group, "Star Graph" is an implementation of such graph which deals with the problem [26]. For a given path, Star Graph greedily chooses a new connection based on (1) the difference in Euclidean distance between the latest connection and its equivalent raw path segment is minimized, and (2) other smoothing parameters. An example of correlation of a single path onto a given set of graph nodes is shown in Figure 3-9.



(a) Observed paths

(b) Selected nodes for Star Graph

(c) An example of path traversing through the environment

(d) The result of correlation of the path in (c) onto selected nodes

Figure 3-9: **An example of the use of graph for path discretization:** This figure shows how paths are correlated with a graph. Given with a set of path (figure (a)), the paths can be discretize over a set of nodes (figure (b)). More specifically, for any single path in the environment (figure (c)), the path will be correlated so that the result is connections between the graph nodes that best represent the whole path (figure (d)).

It has been discussed in assumption 2 in Section 3.3.1 that a collection of paths from an environment should have certain patterns to characterize the scene. As illustrated in Figure 3-10, Star Graphs can clearly visualize the pattern of trajectories within the environment and is therefore an appropriate representation to characterize paths in this thesis' experiment. As a result, all paths analyzed in this thesis will be discretized based on the Star Graph algorithm's decision.

The choices of Star Graph nodes are critical. It is debatable that a standard rectilinear grid might be a good choice to distribute nodes in order to get consistent correlation patterns. However, this thesis will use a few sets of nodes of arbitrary number and arrangement to correlate each set of paths. The major reasons for this decision include:

1. Rectilinear grids are actually a special case of flexible node arrangement.

2. Some patterns of nodes can correlate a set of paths better than others. It is better to spread the risk of encountering bad sets of nodes by having several alternatives.

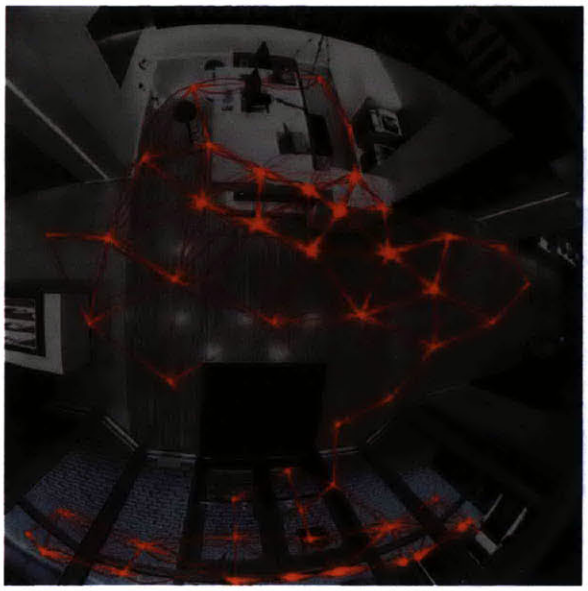


Figure 3-10: **An example of fully-correlated Star Graph:** The thickness of an edge in the graph shows the frequency that path segments are mapped as a connection between both vertices of the edge. Once properly visualized, the graph makes path dynamics within the environment more visible. (Reproduced under permission of George Shaw)

For this thesis' experiments, nodes that are used for correlation will be randomized over the simulated banking environment, regardless of whether the nodes lie on non-passage ways or solid models in the environment. All correlations will use the same set of randomized nodes for standardization unless it is specified otherwise. The number of randomized nodes can be 100, 200, 300, 400 and 500 nodes.

3.3.3 Interpretation of Pedestrian Paths with Frequency Matrices

After simplifying the paths into edges of a graph, they can be further analyzed as frequency matrices whose entries are the number of connections between any pair of nodes made by the provided set of paths, i.e. the weight of the edges in the Star Graph.

In order to disregard the number of paths in the given data set, entries in frequency matrices are normalized as the number of connection between a particular pair of nodes out of the total number of connections made in the graph.

While representing paths as a graph, a frequency matrix omits information about rotation of paths. As a result, it serves as a compact way to describe paths in each data set. The speed of agents is indirectly encoded in the diagonal elements of the frequency matrix

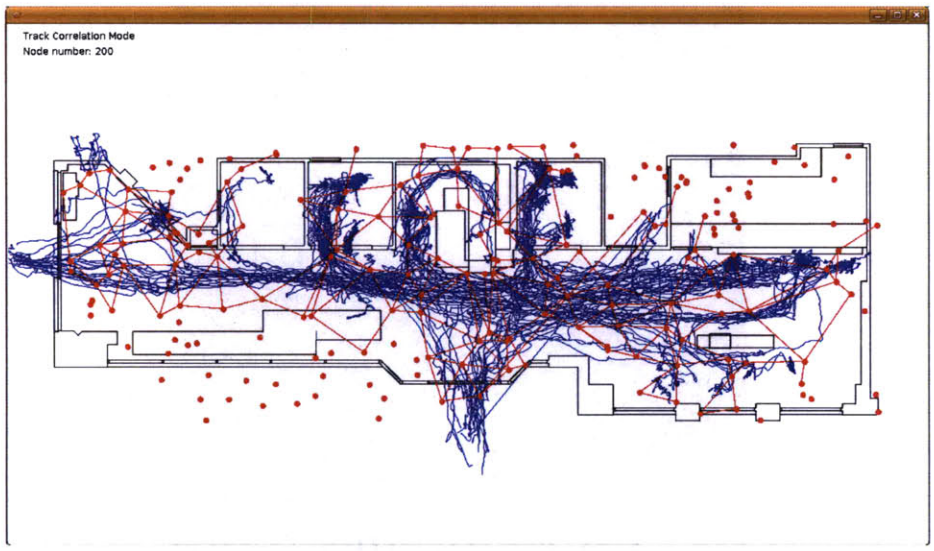


Figure 3-11: **Fully-discretized paths of the one-hour-long raw paths**: This figure shows a discretization of the The one-hour-long raw paths (from Figure 3-1 over 200 preset nodes. The choice of edges are selected by Star Graph. However, the thickness of edges does not represent the frequency of transitions between edges.

because if the agent's speed is low, there are going to be a lot of self-looping sequences. Because of this self-looping, it is necessary to ensure that the frame rate of the simulation matches the real data's frame rate. Otherwise, the speed in the simulation will be misinterpreted because of oversampling or undersampling. The frequency-tuning has been neglected in this thesis' experiments for simplicity. Although the absolute value of the experimental results will not be accurate, the relative values and relationship between measured data should still be approximately correct. Because all results, especially speed components, are measured under the same (but possibly incorrect) sampling rate. Section 3.3.4 will explain how to use this information to find similarity between two sets of pedestrian paths.

3.3.4 Comparison Metric

As mentioned in Section 3.3.3, pedestrian paths can be represented with transition frequency matrices. During the training process, it will be required that frequency matrices from the raw data and the simulated data have a quantifiable way to compare their "similarities." This section will discuss the comparison metric that will be used in this thesis' experiments.

Specification of the Metric

Good metric for comparison of frequency matrices should have the following properties:

1. **Similarity as judged by humans and comparison scores are positively correlated**

   Although the correlation between similarity and score should ideally be linear, other forms of correlation such that the gradient of the comparison score does not significantly vary from one range of similarity to another is acceptable. The only reason that the correlation should be positive is because it is easier to form an impression of being "more similar" if the score is high rather than low.

2. **The final score is normalized**

   Having normalized metric scores is critical because it allows valid and fair comparison between different simulation settings regardless of the number of nodes used in the Star Graph or the number of paths simulated throughout the session.

3. **The subscores (and final score) are bounded within finite upper and lower bounds**

   If either upper and lower bounds of the metric's score is not finite, inclusion of even one extremely positive or negative subscores can significantly alter the final score. Therefore, having a small, limited range of possible scores can help alleviate the problem.

Modification of Kullback-Liebler Divergence into Comparison Metric

The metric that this thesis uses to compare two frequency matrices is inspired by the Kullback-Liebler divergence formula as shown in equation (3.14). Kullback & Liebler proposes a divergence formula to measure how different (or divergent) any two probability distributions are, based on the difference in information [18]. This divergence formula in the discrete domain can be stated as:

$$D_{KL_{orig}} = \sum_i P_i \ln(\frac{P_i}{Q_i}) \tag{3.14}$$

whereas $\ln(\frac{P_i}{Q_i})$ is defined by Kullback & Liebler as the information in entry i to decide if it is generated by probabilistic distribution P or Q. However, instead of summing all

elements to find divergence, the formula is applied to each pair of corresponding entries between two frequency matrices to find preliminary distance between the entries, resulting into only one term in the simplified formula.

$$D_{KL} = \frac{P}{P+Q} \ln(\frac{P}{Q}) \tag{3.15}$$

Then, for the sake of symmetry, a symmetric term is added to the original equation, resulting into:

$$D_{KL} = \frac{P}{P+Q} \ln(\frac{P}{Q}) + \frac{Q}{P+Q} \ln(\frac{Q}{P}) \tag{3.16}$$

Equivalently, it can be written as:

$$D_{KL} = (\frac{P}{P+Q} - \frac{Q}{P+Q}) \ln(\frac{P}{Q}) \tag{3.17}$$

Suppose that $Q \leq P$, then $D_{KL} \geq 0$ because P and Q in the context of this thesis have been normalized to have values in the range of [0,1] as stated in Section 3.3.3, making them strictly non-negative. In other words,

$$\begin{aligned} D_{KL} &= (\frac{P}{P+Q} - \frac{Q}{P+Q}) \ln(\frac{P}{Q}) \\ &\geq 0 \end{aligned} \tag{3.18}$$

As a result,

$$0 \leq \exp^{-D_{KL}} \leq 1 \tag{3.19}$$

$$0 \leq \exp^{-\frac{P}{P+Q} \ln(\frac{P}{Q}) - \frac{Q}{P+Q} \ln(\frac{Q}{P})} \leq 1 \tag{3.20}$$

Because the Kullback-Liebler divergence in equation (3.16) is symmetric, the above argument is also valid for when $P \leq Q$. In practice, a constant of insignificant value should be added to both P and Q to prevent either or both of them to have values of zero and cause D_{KL} to explode to ∞. With this constraint, it can then be stated that:

$$0 < \exp^{-\frac{P}{P+Q} \ln(\frac{P}{Q}) - \frac{Q}{P+Q} \ln(\frac{Q}{P})} \leq 1 \tag{3.21}$$

As a result, it can be concluded that an exponential of Kullback-Liebler divergence

matches the specification of comparison metric because the resulting comparison score is bounded and strictly positive. Hence, more similar entries earn better scores than less similar ones.

The final metric similarity score, S, is the arithmetic mean of the divergence of corresponding pairs of entries in frequency matrices. For graphs with m nodes, the distance between the two frequency matrices is:

$$S = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} \exp^{-\frac{P_{i,j}}{P_{i,j}+Q_{i,j}}\ln(\frac{P_{i,j}}{Q_{i,j}})-\frac{Q_{i,j}}{P_{i,j}+Q_{i,j}}\ln(\frac{Q_{i,j}}{P_{i,j}})}}{m^2} \tag{3.22}$$

Dealing with Sparsity of Frequency Matrix

It is theoretically valid for a node in a graph to be connected with any other nodes within the same graph. However, because nodes in this thesis' application are spread over a two-dimensional plane and edges are connected only when the connections best represent a segment of a given path, it is highly unlikely that there are connections between nodes that are far apart from each other. Therefore, frequency matrices are naturally sparse.

Sparsity becomes an issue in the metric calculation because there are too many entries whose values are zero in both data and simulation frequency matrices. If these entries are included in the calculation, then the result will be inaccurate. If they are treated like normal entries, they would receive full scores. However, because the matrix is sparse, the final score will always be close to perfect, regardless of the similarity of non-zero entries. If they otherwise receive scores of zero but are not excluded from the normalization process, then the final score would be skewed towards zero. As a result, it is a design decision to completely omit any entries that are zero in both data and simulation frequency matrices in order to avoid significant bias towards either end of extreme scores.

Considering the discussion about sparsity, the formula should be further modified. If the metric formula uses m graph nodes, the formula becomes:

$$S = \frac{\sum_{i=1;(i,j)\notin Z}^{m}\sum_{j=1;(i,j)\notin Z}^{m} \exp^{-\frac{P_{i,j}}{P_{i,j}+Q_{i,j}}\ln(\frac{P_{i,j}}{Q_{i,j}})-\frac{Q_{i,j}}{P_{i,j}+Q_{i,j}}\ln(\frac{Q_{i,j}}{P_{i,j}})}}{m^2 - |Z|} \tag{3.23}$$

where $Z = \{(i,j)|(P_{i,j} = 0) \wedge (Q_{i,j} = 0)\}$

3.4 Software Implementation and Design Decisions

From the discussion in Section 3.2 and 3.3, both optimization framework and its objective function have been elaborated. This section will discuss about how both components are integrated into the software system used in this thesis' experiments and about other design decisions that have been made for practical purposes. All modules in this program are implemented in Java language.

3.4.1 Major Modules in the Software System

This software is a result of collaboration with Rony Kubat and George Shaw. The whole software system spans from raw data processing to agent simulation. Generally, the software implementation for the optimization process can be divided into four major modules, described in the following sections.

Agent Behavior Module

This module defines the behavior and decision-making criteria of each type of agents. All parameters used in the optimization are parameters from this module. As a result, agents are implemented so that their numerical parameters are adjustable. Variables in agents that have potentials to become adjustable parameters are implemented as a type of Java object which can configure their values during the initialization.

After the simulation has begun, except for the values of variable parameters, all details in this module are treated as given for the rest of the optimization process.

Simulator Module

The simulator puts the agents into the simulation environment. During each time step, the simulator regulates and calls for agents' activities such that their behavior concurs with the agent-behavior module's specification. After the track is simulated, this module records and optionally visualizes tracks which agents traverses over during the simulation. The simulation is atomized into "time steps" which are currently not calibrated to correspond to any specific real-world time. Nevertheless, within the simulation, the time step duration has consistent proportion to other units.

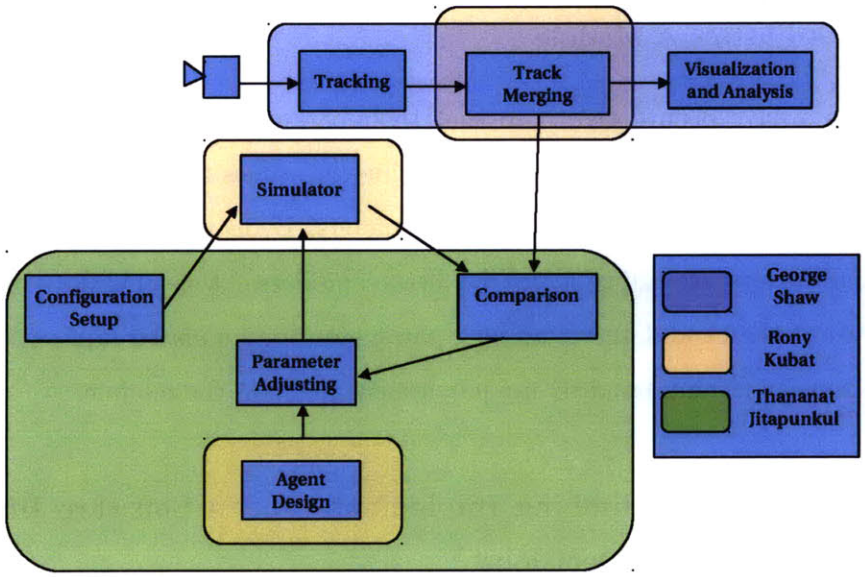


Figure 3-12: **A diagram of modules in the implemented software system as a part of Data-driven Architecture project:** The software system is a collaboration with Rony Kubat and George Shaw. My major contribution is contained in the area of agent optimization. The training procedure starts by providing the software with initialized values for parameters defined in agent implementation. Then, the program will start to iterate through the process of "simulation", "comparison of result with data" and "parameter adjustment" until the training program can generate satisfactorily optimized set of parameters.

Data-Result Comparison Module

As discussed in Section 3.3.4, the comparison metric for this module compares the paths collected from the simulation in the simulator module with the paths from the real data set by using a formula in equation (3.23). The output of this module is a similarity score ranging from 0 to 1, where the score of 1 means that the data and simulated paths are perfectly the same to each other.

Parameter Adjustment Module

Because agents have been designed so that their parameters can be easily adjusted, the only responsibility of this module is to determine the values that should be set to agents' parameters and call agents to change their parameters. By using the hill-climbing technique in which parameters are adjusted to the best candidates, this module is usually called only after all of the candidate values of a parameter are simulated and evaluated. Later simulation runs will be updated to any adjustment made in the module.

3.4.2 Details of Hill-climbing Implementation in Software Implementation and Design Decisions

Both fundamental concepts of hill-climbing and general organization of this software system have been presented in Section 3.2.6 and 3.4.1 (and illustrated in Figures 3.2.6 and 3-12 respectively). This section will discuss the implementation details, practical issues and design decisions of the system.

Figure 3-13 shows a flow chart illustrating the implementation details.

Determining the Rate of Change

The choice of parameter adjustment rates per iteration is critical to the performance of hill-climbing algorithm. Smaller rate means higher resolution in the search for local optimum. However, the differences of objective function values of neighbors might get extremely small, especially for cases with relatively flat objective functions. In addition, this leads to slow convergence of the algorithm. On the other hand, it becomes easier for larger rate to skip the local optimum and inadvertently undersample the objective function.

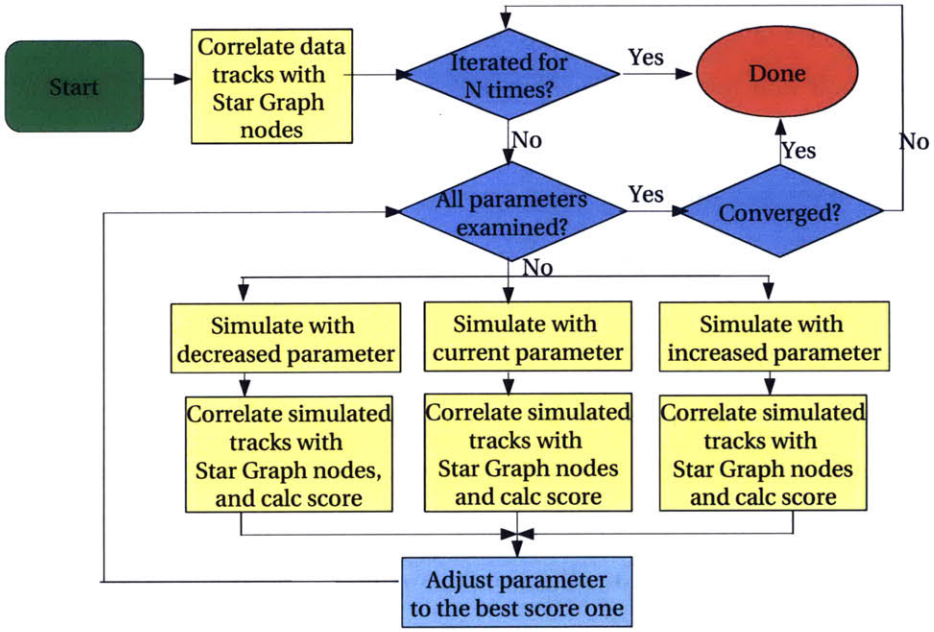


Figure 3-13: **Flow chart of the implemented agent training program:** The flow chart shows the procedure of the training program in optimizing simulation agents. Bounded to a finite number of iterations, each loop follows stages of candidate selection, simulation, evaluation based on real data tracks and parameter adjustment. Once an optimized solution is found or the runtime takes too long, the program terminates. Note that parameter values' candidates are chosen greedily from the current value's neighbors. The algorithmic version of this flow chart can be found in algorithm 1 in appendix C

The rate of change that have been used in the experiments of this thesis is 1% (both for the increasing and decreasing changes).

Decisions on Halting the Training Program

There are two types of possible outcome for any run of parameter adjustment program: the program finds the optimal solution or it pursues the solution for an infinite amount of time. In the cases where solutions are found, there are two criteria which determine whether the current set of parameters is optimal or not.

The first one is whether the parameters have changed over the past iteration. If not, then it can be assumed that no further adjustment will be made because no other candidate values can outperform the original values and readjust the parameters. The second criteria is whether the score has improved but at slower rate than a predefined threshold. It is beneficial for the objective function with relatively flat landscape because the criteria saves the time that would have been spent to search for a new set of parameters whose score is

marginally better than the current one. If either of the two criteria is satisfied, then the program will halt and return the current set of parameters as solution.

In the worst case, there is no optimal solution for the objective function. Then the program will never halt under the first two criteria. To prevent this outcome, the training program sets the maximum number of iterations that each run of metric calculation is allowed to iterate. Although this decision might prevent the hill-climbing algorithm from reaching the local optimum before halting, it is generally better than waiting for an indefinitely long time before the program halts and returns an optimal value. This is especially true because the landscape of the objective function is not known beforehand.

If users suspect that the current parameters are not optimal and better results are desired, the final parameters from the previous run can be used as the initialized values. In this thesis' experiment, the maximum number of iteration is arbitrarily set to 20 times. Suppose that the rate of change of parameter values is set to 1% per iteration, then during the simulation run, the parameters can make at most $(1.01^{20} - 1) \approx 22.02\%$ of adjustment from their initialized values.

Nondeterministic Objective Function

This issue is realized later on in the project that even with the same set of parameters, the simulation cannot always generate the same (or equivalent) transition frequency matrices. As a result, the measured similarity score becomes a probabilistic random variable. Even though the standard deviation is only 0.01 out of the total score of 1, the issue can become critical in cases where:

1. **The objective function's landscape is flat** – The variance of the metric becomes a significant source of noise in this case. This might lead the training program to eventually choose a non-optimal value as solution.

2. **The threshold for a training program's termination is lower than or equal to the standard deviation** – It has been stated earlier that the training program is designed to halt if the current iteration's score shows too small improvement in objective function metric from the previous iteration. Metric scores with non-negligible standard deviation make the it impossible for the training program to justify its halting decision.

One possible solution is to measure the simulation performance metric several times and take the average score for any set of parameters. However, huge computational cost of simulation makes the approach infeasible, making the problem unsolved.

3.4.3 Overall Asymptotic Runtime of the Program

From the proposed model of the optimization system, the runtime of the program can be estimated as equation 3.24. There are two terms in this asymptotic runtime equation. The first term is the runtime for simulating the environment and calculating performance score for every adjustable parameters. The second term is derived the runtime in the path-node correlation process to generate the frequency matrix.

$$R(m, n, s, t) = O((m + s^2)n + st) \tag{3.24}$$

where $R(m, n, s, t)$ is a function for the runtime of the program

m is the number of simulation steps

n is the number of adjustable parameters

s is the number of Star Graph correlation nodes

t is the number of tracks that are collected during the simulation

3.5 Experimental Setup and Evaluation Criteria

To make a credible argument for this thesis, it is necessary to answer the following questions through a verifiable experiment.

1. Does the proposed comparison metric have the desired properties as discussed in Section 3.3.4?

2. Is it true, as the assumption suggests in Section 3.3.1, that the amount of paths in simulated track data make the data set more similar to the real track data set?

3. What is the effect of the training program on the correlation between data and simulated track data set? Is the result satisfactory?

This Section will describe and elaborate how experiments for each of the issues are set up, as well as the criteria to evaluate their success. The parameter initialization list can be found in Table A.3 in the appendix Section.

3.5.1 Experiment 1: Comparison Metric Verification

The key of this thesis is the use of comparison metric to evaluate the quality of paths generated from simulation and to train pedestrian behavior of agents in the retail environment simulation system. In order to validate all other experiments, it is necessary to also validate this comparison metric.

According to the second assumption as stated in Section 3.3.1, "Both simulated paths and dataset paths should increasingly exhibit greater similarity as more data on both sources are collected." Therefore, if the comparison metric is valid, then as the number of simulation steps is increased, the comparison scores of these simulations should also strictly increase. However, because of non-deterministic nature of the path generation, the similarity scores should be analyzed in terms of mean average and standard deviations.

Setup and Procedure

The experiment is run with the standard parameter values for the 31 optimizing parameters (including dummy variables.) Since the experiment does not proceed to optimize these values, they remain constant. The only independent variable in this experiment is the number of runtime steps. The experimental procedure should proceed as followed:

1. Prepare a one-hour-long path data set as a reference data
2. Randomize three sets of 200 nodes over the environment bound.
3. Use Star Graph to correlate the reference path data and collect a frequency matrix.
4. Simulate a bank scene for 1000 time steps. Collect the tracks of paths that simulated agents traverse over the environment.
5. Use Star Graph to correlate the simulated path data to get a frequency matrix.
6. Use the comparison metric formula as described in Section 3.3.4 to calculate a similarity score from raw-data frequency matrix and simulated-data frequency matrix.

7. Repeat steps 4 to 6 for 100 times. Record the average score and the standard deviation of these runs.

8. Repeat steps 4 to 7 with 2000, 3000, ..., and 10,000 time steps.

9. Compare the mean and standard deviation of similarity scores for each length of simulation time steps.

Evaluation Criteria

The comparison metric is considered to be valid if the mean of similarity scores for each number of simulation time steps strictly increases along with the simulation length. In addition, the standard deviation must be within an acceptable range (i.e. 1% of the simulation score).

3.5.2 Experiment 2: Measurement of the Number of Star Graph nodes' Effects on Optimization Program's Performance

Because the comparison metric depends on Star Graph correlation to generate the frequency matrix, the number of Star Graph nodes should have significant effects on how comparison scores reflect the performance of simulation. On one extreme that there is only one node, the score should always be perfect because every path segments will be mapped to the node, generating 1×1 similarity matrix. On the other extreme where there are infinitely many nodes, the score should converge to zero because every path segments can be mapped to their own nodes. Having the number of graph nodes on both extremes are therefore useless to generate credible performance score. However, it is not clear how the middle-ranged number of node will perform in terms of scores.

Moreover, it should be assumed that the number of paths should slightly affect the runtime of each optimization iteration because there are more entries in the performance matrix to calculate into scores. However, given that the majority of time spent on the optimization process is spent on path-generating simulation, its effect should not be very significant.

This experiment aims to explore the effects of the number of nodes in middle ranges on the optimization performance, both in terms of scores and its runtime.

Setup & Procedure

The "middle-ranged" number of graph nodes in this experiment is defined to be a few hundred nodes, given the size of the simulation environment. The setup will use the standard initialized parameter values for the 31 optimizing parameters. To fully illustrate the effect of these nodes, the optimization process will be run for 5000 time steps, with an offset of 500 time steps.

The experimental procedure should proceed as followed:

1. Randomize locations for 100 nodes over the simulation environment
2. Run optimization process and collect the runtime and the performance score of the result under 5000 time steps.
3. Repeat steps 1 and 2 with 200, 300,...,500 nodes
4. Using the optimized parameters as initialized values, rerun 10 simulations and calculate the average score and standard deviation of the generated paths. Compare the values with the initial score and the post-training score.

Evaluation Criteria

This experiment will be evaluated based on how the performance improves or degrades with the increased number of nodes and whether the increased runtime is noticeable or not. Moreover, the simulation reruns will serve as a sanity check to validate the optimization's success.

3.5.3 Experiment 3: Measurement of the Number of Simulation Time Steps' Effects on Optimization Program's Performance

As the simulation runs longer, the hidden patterns of the path becomes easier to observe until the transition frequency between the correlating graph nodes become stable, i.e. all patterns are revealed. It can then be implied that paths from longer simulations better represents the potential performance of the current parameter configurations. As a result, this experiment seeks to explore the effect of simulation length on the optimization program's ability to improve the performance scores.

Setup & Procedure

1. With the standard parameter initialization, run optimization with the one-hour-long raw path data and paths from simulation with 1000, 2000, ..., 10000 time steps, using 200 Star Graph nodes. After the optimization, collect the optimized parameters.

2. For each set of the optimized parameters, run a simulation for 10000 time steps to generate paths.

3. Measure the quality of each set of paths with the performance metric.

4. For each of the calculated scores, compare the score with:

   (a) the average and standard deviation of the base, pre-optimized score of each time duration (acquired in experiment 1).

   (b) the score of paths generated by running optimized parameters of each time-step case for 10000 time steps.

5. For part 4a, rerun 10 simulations, each of which uses the optimized parameters as initialized values, and calculate the average score and standard deviation of the generated paths. Compare the values with the initial score and the post-training score.

Evaluation Criteria

There are two parts of evaluation in this experiment. The first part is about whether the optimization for each different simulation steps has improved the performance score from the base level as measured in experiment 1. (The base line level is the average of performance scores of simulations with the same number of steps.) If the performance has improved, the optimized scores should be significantly improved from their base lines. The reruns' average score will serve as a sanity check of the optimization process.

The second part is about how well the optimized parameters from shorter simulation can generalize to the later part of the simulation. The benchmark of this aspect of experiment is the performance score of the paths that are generated by a simulation initialized by 10000-simulation-time-step optimized parameters, running for 10000 time step. If parameters are optimized under much fewer time steps (such as 3000 time steps), when these parameters are run for 10000 time steps, the parameters should not generalize to the path pattern of

10000 time steps, and the resulting sets of paths should earn inferior or equal performance compared to the benchmark.

3.5.4 Experiment 4: Measurement of Optimization Program's Performance Under Randomized Suboptimal Parameters

It will later be discussed in the results of experiment 3 in Section 4.1.3 that the optimized parameters do not show significant improvement from before the optimization. That means that the landscape of the objective function around the standard initialized parameters is already at a local maximum. There are two possible explanations. The first explanation is that parameters of the implemented agent models do not have significant effects on how the agents choose their paths, implying that the objective function is flat. The other possible explanation is that the chosen values of the parameters are at some local optimal point of the objective function landscape.

This experiment's purpose is to prove or disprove the second explanation to provide clues for future implementation.

Setup & Procedure

1. Randomize values for all 31 parameters (but still within the proper context of each parameter)

2. Run optimization software to adjust the values using 5000 simulation time steps and 200 Star Graph nodes.

3. Compare the new performance score with the base line scores from experiment 1's result.

4. Repeat steps 1 to 3 under the new randomized parameter values for a few times (such as three).

5. For each case, rerun 10 simulations, each of which uses the optimized parameters as initialized values, and calculate the average score and standard deviation of the generated paths. Compare the values with the initial score and the post-training score.

Evaluation Criteria

The local-optimum explanation is proved if the objective function scores of any of set of randomized parameters can be improved after the optimization process. It is disproved otherwise, and can be concluded that the given parameters are too insignificant to improve the performance score. Any improvement shown in the experimental result will be justified by the average scores of the simulation reruns.

Chapter 4

Data Collection and Analysis

4.1 Experimental Results

4.1.1 Experiment 1: Comparison Metric Verification

The results of experiment 1 can be shown in Figure 4-1. The trend of the performance score starts to rise rapidly in the optimization using only a few thousand time steps of simulation and then shows signs of leveling as the number of simulation steps rises. The result is expected because while the optimization program adjusts parameter values, it simulates tracks to receive feedbacks about the choices of new values for these parameters. If the tracks are simulated using only a few time steps, it is highly unlikely that their paths can represent the rich patterns in the raw data.

The real significance of this graph is the standard deviation of each measurement which grows linearly as a function of simulation time steps. This is a clear sign that there is a large effects of non-determinism on the performance score calculation. However, the performance metric as introduced in Section 3.3.4 is deterministic and the raw data paths are always fixed. Therefore, there can be two major sources of non-determinism: the generated simulation paths or the Star Graph correlation. According to [26], the only major source of non-determinism in the full Star Graph is the LBG Vector Quantization algorithm. However, because nodes that are used for correlation are preset, this thesis does not use the non-deterministic part of Star Graph to choose nodes. Although it is still possible that there are bugs in Star Graph implementation that causes the non-deterministic behavior, it is equally likely that the generated simulation paths are non-deterministic. In

other words, even though parameters are identically initialized, agents can generate paths that are slightly different. The causes of non-determinism in the generated path might have been bugs in agent and simulation implementation.

Inconsistency between the sampling rate in the real and simulated data (mentioned in subsection 3.3.3) is most likely unrelated to the non-deterministic problem because the frequency rates on both cases are fixed. However, the fact that the difference in frame rates invalidates the diagonal entries of the frequency matrices makes it very likely that the performance score is incorrect. However, for the rest of the analysis, this fact will be ignored because as long as the trend of the performance score is the major concern. There should not be too much effects because the error caused by the frame-rate inconsistency applies approximately the same for all score calculations, preserving approximate relationship between these scores.

Because of the insight about rising standard deviations, it shows that the scores of the generated paths can be increasingly volatile as the number of simulation time steps used by the optimization process increases. As a result, for later experiments, the simulation time steps over 10000 steps will not be explored until the non-determinism problem can be satisfactorily solved.

4.1.2 Experiment 2: Measurement of the Number of Star Graph nodes' Effects on Optimization Program's Performance

The result in Figure 4-2 indicates that the performance scores are higher in the case where fewer nodes are used. All cases show improvement in the train processes. However, the average score from ten reruns using the optimized parameters does not show any improvement from the initial value. In fact, the average is worse than the initial score on all cases.

This can be explained based on the result of experiment 1. As demonstrated in experiment 1, the performance score is not deterministic, making the score subjective to a small amount of variance. It is highly likely that the "improvement" in the score is a result of the score swinging to the extreme value and the hill-climbing procedure records the maximum score that shows up. As a result, the performance score obtained right after the training process is exaggerated because the value is the "maximum" score possible rather than the "average" score.

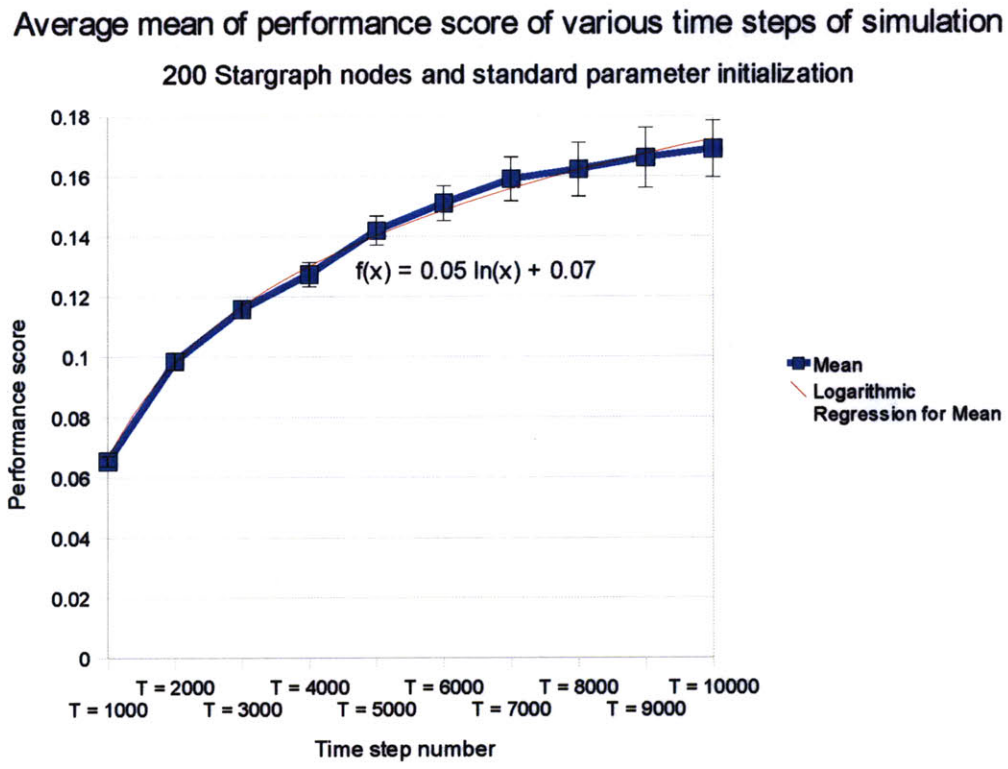


Figure 4-1: **Average mean of performance scores of various time steps of simulation:** The graph shows the rising trend of the performance scores as the number of simulation steps used in the optimization process increases. While the scores increases in a logarithmic function (as shown in the regression graph line), the standard deviation of these scores increase linearly. All scores are mean averages of 100 measurement.

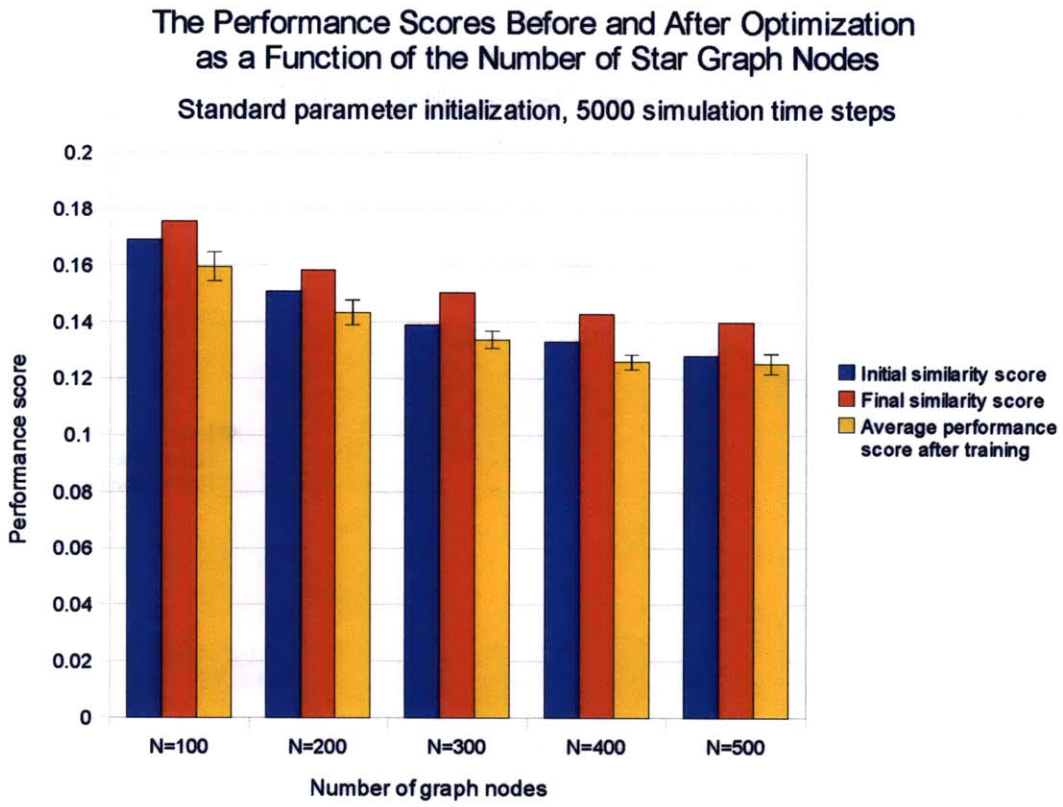


Figure 4-2: **Performance scores before and after optimization as a function of the number of Star Graph nodes:** The graph shows that the simulation with lower number of nodes, the performance score tends to be higher. While the score result from the hill-climbing process shows a significant improvement, it is shown by the average post-optimizing scores that such improvement is likely to be exaggerated. Therefore, there seems to be no clear trend of how the number of Star Graph nodes that are spawned to the scene have correlation with the score improvement from the optimization. The setting for this graph uses standard parameter initialization and simulates over 5000 time-step feedback.

4.1.3 Experiment 3: Measurement of the Number of Simulation Time Steps' Effects on Optimization Program's Performance

As stated in the experimental description in subsection 3.5.3, this experiment intends to observe the effect of the time steps used in the feedback simulation to train parameters on two aspects of performance: the asymptotic runtime of the system and optimized score improvement.

Asymptotic runtime as a function of time steps

As demonstrated in the runtime analysis in subsection 3.4.3 of Chapter 3, the runtime should vary linearly with the number of time steps that the optimization process uses to run its feedback simulation. This effect is demonstrated in Figure 4-3. Although the runtime violently rises and falls between 6000 time steps and 10000 time steps, the overall relationship is still linear.

Improvement of performance scores after optimization as a function of the number of time steps used in the optimization process

The optimized parameters are trained based on simulation feedback with a certain number of time steps (i.e. 1000,2000,3000 steps etc.) There are two aspects of the performance that should be considered. The first is whether the trained parameters can improve performance scores when they are simulated with the same number of time steps as the one they are trained in the optimization process. The second one is whether the trained parameters can still perform well in longer simulations. For example, if a certain set of parameters are optimized using 5000 time-step simulation feedback, can it perform well when one use the same set of parameters to run a simulation for 10000 time steps and still get a significant improvement in performance scores?

The first aspect of the performance can be observed in Figure 4-4. It can be shown that the performance scores of simulations that are simulated for fewer time steps are much less than those of longer simulations. The trend of the curve is increasing until there is a sign of leveling out as the time steps approaches 10000. The performance scores right after the optimization show significant improvement. However, like the result in experiment 2, the scores calculated from the simulation reruns, which use the optimized parameters

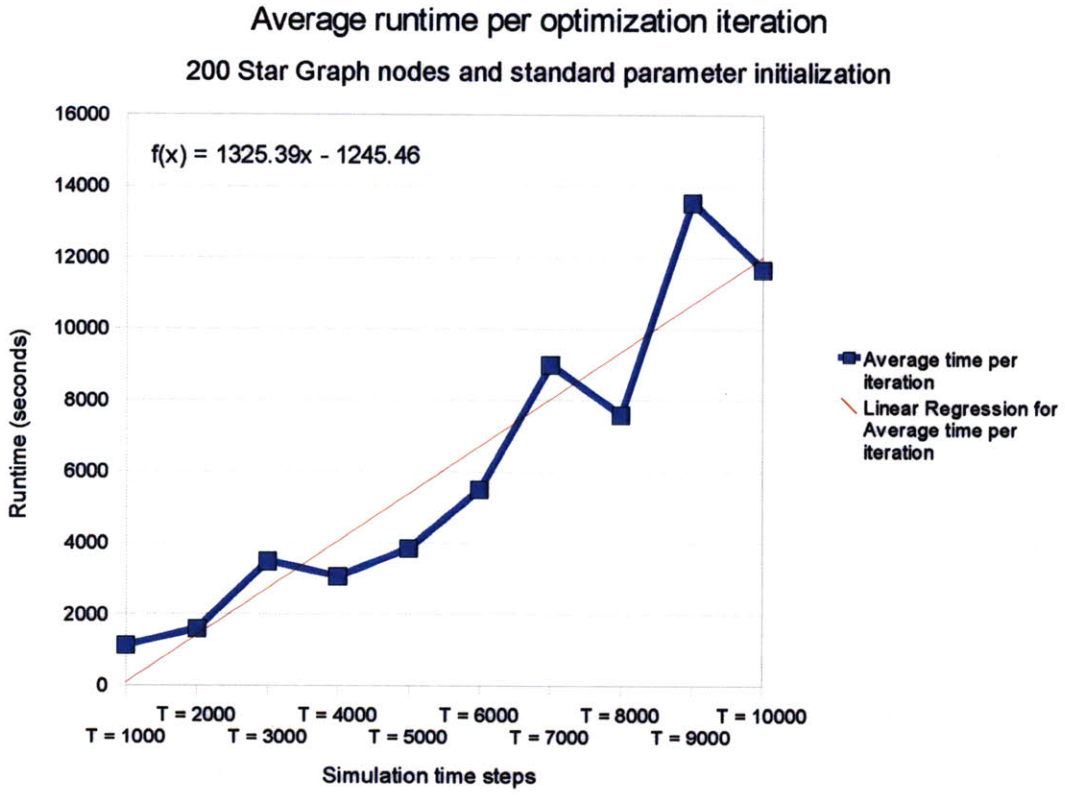


Figure 4-3: **Runtime per Optimizing Iteration as a Function of the Number of Simulation Steps Used in Optimization:** This graph shows a linear relationship between the number of simulation time steps used in the optimization and the resulting runtime per optimization iteration. This is in accordance with the runtime analysis in subsection 3.4.3.

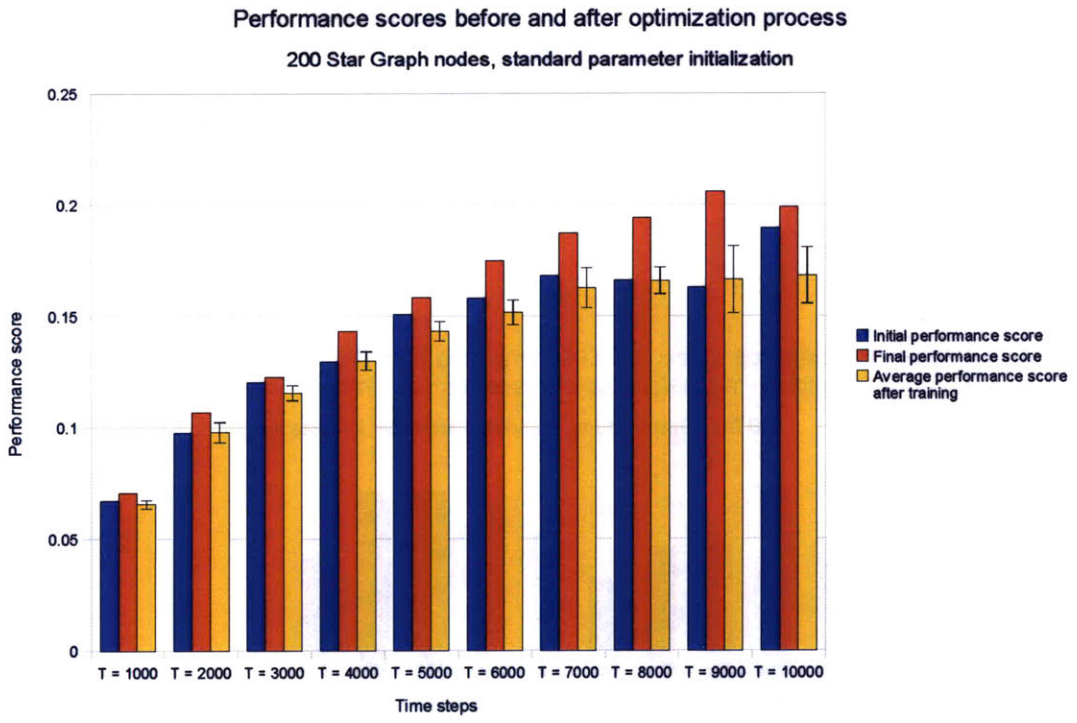


Figure 4-4: **Performance scores before and after optimization process as a function of time steps:** This graph shows how the performance score is improved after optimization, compared to the initial scores. Both initial and final scores are calculated using simulation with the same number of steps as that used in the optimization process. In other words, if the optimized parameters' optimization simulates 3000 steps as its feedback, then the simulation to calculate the final scores also uses 3000 steps.

as initialized values, do not show any signs of improvement from the original parameters. Despite the fact that the simulation length in this experiment is the same as that of the feedback simulation in the optimization process, the scores from these reruns are worse-off than the initial scores.

As a result, it is not unexpected to observe in Figure 4-5 that the performance scores in the generalized case also do not improve from the original scores which use the standard parameters as shown in experiment 1 (see Figure 4-1). The mean value of the scores is barely different from the original score.

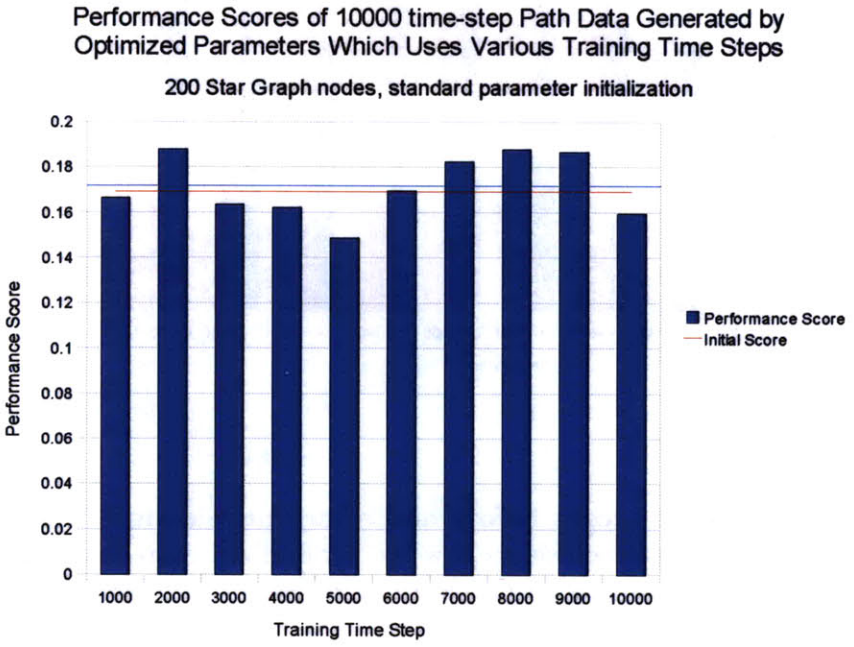


Figure 4-5: **Performance scores of 10000 time-step path data generated by optimized parameters which use various training time steps:** Using the optimized parameters which used different time steps for optimization, the graph shows how these parameters would perform when they are run for an equally long time (i.e. 10000 time steps in this case.) Because each parameter set is run once, the final score fluctuates around the mean line (shown as the deep blue line). It should be noted that the difference between the initial score (orange line) and the mean of the final score is negligible.

4.1.4 Experiment 4: Measurement of Optimization Program's Performance Under Randomized Suboptimal Parameters

As explained in the experimental description in subsection 3.5.4, this experiment aims to explore whether the standard initializing parameters are coincidentally located at the local optimum, or whether the objective function with the provided parameters is actually flat. Using three randomized initializing parameters as in Table A.3, the experimental result is shown in Figure 4-6. Despite a significant score right out of training (whose validity is questioned in experiment 2), it is clear from the average scores from reruns that, regardless of the values of initialized parameters, the optimized parameters do not perceivably improve the performance scores compared to the original parameters. In all cases, agents fare much worse under the optimized parameters than under the original parameters. If there had been any improvement at all, it is only a small change and has been overshadowed by the noise of the non-deterministic performance scores. From the experimental result, it is still inconclusive to determine whether the optimization's objective function under the chosen set of parameters is flat.

4.2 Discussions

From the above experiments, it can be concluded that the optimizations in the experiments cannot be properly evaluated due to two major reasons. The first one is the non-determinism of the performance score calculation as demonstrated in the result of experiment 1 in subsection 4.1.1. As a result, even if any of the experiments were to show signs of minor improvements, the variance of the scores make it infeasible to verify the improvement. This is especially true as the duration of the feedback simulation in the optimization process becomes longer because the the variance of the scores grows as a linear function of this duration. In addition, it is shown in Table A.5 that in T=8000 case, dummy variables like "queueTeller_transactTimeOtherCore" and "queueTeller_transactTimeOtherVar" (refer to Table A.2) are adjusted during the optimization. Because dummy variables serve as a sanity check for validity of the optimization process, it shows that there are problems in the optimization process. It is likely that variance of the performance score is the cause behind this incident.

The second reason is that the objective function of the system is suspected to be flat,

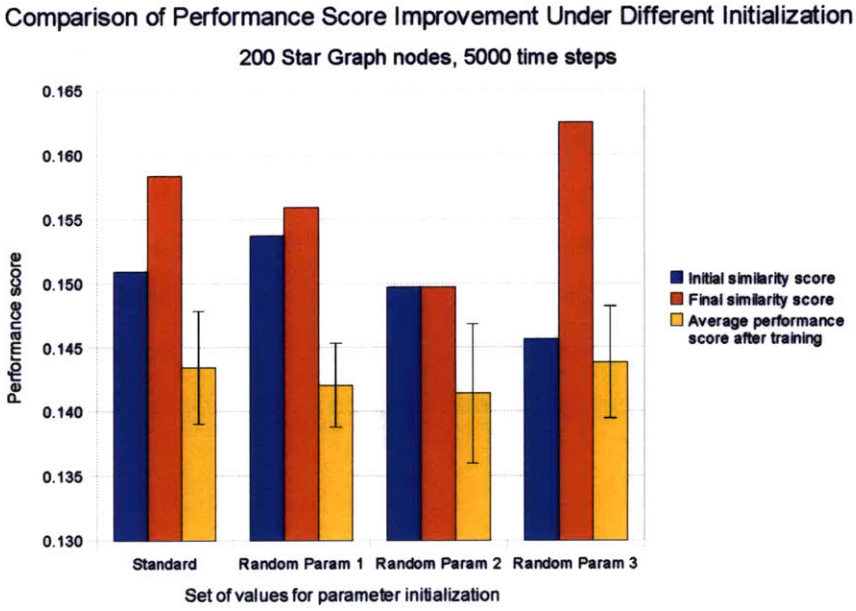


Figure 4-6: **Comparison of performance scores under different initializing values:** This figure compares performance scores when the parameters are initialized differently. The alternative sets of parameter values are randomized with uniform probability within ±80% of the standard parameter values. The optimization is trained with 5000-step simulation feedback, and the performance scores are calculated using the paths after 5000 time steps of simulation and paths from the one-hour raw data. After the training is complete, simulations using the optimized parameters are rerun for a sanity check on the significant improvement of post-training scores. It turns out that the average scores for these reruns are much less than the post-training scores.

making hill-climbing technique used in the optimization useless. Evidences in the experiments show that the function is supposed to be flat regardless of the number of Star Graph nodes (from experiment 2), the length of feedback simulation used in the optimization process (from experiment 3). However, the effect of the choice of randomized parameters (from experiment 4) is inconclusive. This suspicion is supported by the number of iterations used to optimize parameters as stated in the raw data Tables A.4, A.5, A.6, A.7, and A.8. While it is usual that each optimization in the experiments runs for more than one iteration, the number of iterations is bound to fewer than four times. This is a rather unexpected result, unless the initialized values are already very near to its local optimum. However, since this is also true for other randomized initializing values, it is highly likely that the objective function itself is flat and that the multiple iterations are triggered by the noise of the performance score calculation.

The most probable culprit for the flat objective function issue is poor choices of training parameters. Although the chosen parameters have influences on agents' behavior, they do not directly influence the agents' path choices. As discussed in subsection 3.2.2, the most important criterion for agents' path choice is the shortest unobstructed paths. Small deviation from the paths can then be influenced by other factors such as close interpersonal distance and open-space preferences. Therefore, the adjustment of these parameters does not have enough effect on the paths to significantly change the values of the objective function.

Another less urgent issue shown in the experiments is that the performance score has signs of being bound to 0.2 which is a relatively low score. The major reasons include insufficient complexity in agent implementation and insufficient amount of data. According to Figures B-1, B-2, B-3, B-4 and B-5 in the appendix, the queuing agents begin to clog up the environment, obstructing many possible paths that can be otherwise traversed through the environment. Therefore, it is possible that entries for some transitions in the frequency matrix is lower than what they could have been. This symptom is a sign of insufficient configurations in the environment.

A more likely reason behind the low-score problem is insufficient amount of data. It has always been assumed that the raw data represents human movement within a specific environment if the video sequences are sufficiently long. In addition, it has been justified in subsection 3.3.1 that the one hour's worth of raw video data should be acceptable. However,

it is possible that the length of the video is still too short to make such assumption.

In fact, as it has been pointed out in subsection 3.3.3, the fact that the frame rates of the simulated environment and the raw data are not properly tuned up can invalidate all the above calculated scores, possibly making the scores lower than their real values. However, as long as the aim of these experiments is to observe the relationship between scores in different cases, the relationship should still be preserved.

Chapter 5

Summary

5.1 Contributions

In this thesis, I have presented and implemented a new agent-training module for pedestrian movement modelling in retail environment analysis, specifically in a banking environment. The module's design is based on the agent-based modelling technique which has been discussed in subsection 2.2.3. However, in order to simplify the interaction factors from the path generation, the objective function that guides the agent parameter optimization process is based solely on similarity between the patterns of the data and generated paths. Therefore, my major contribution is the introduction, justification and implementation of this objective function (which is also referred to in the thesis as the performance score), and its integration into the agent-modelling procedure.

As described in subsection 3.3, it is assumed that a collection of paths over a long duration contain all or most of the activity patterns that can happen in the environment, and in order to complete any activities in the same environment, the actors will have to traverse similar path transitions. Therefore, the similarity between tracked paths in the raw data and the simulated result can be calculated by applying the formula in equation (3.23) to the transition frequency matrices of both path sets. In order to simplify the definition of transition, all paths are discretized and correlated with the same set of nodes which are randomly distributed over the space in the environment. The major advantages of the formula are its simple calculation, bounded results and positive correlation with the similarity, making it easier and cheap to evaluate any combination of paths. Further justifications of formula (3.23) as a metric in the optimization objective function have been

discussed in detail in subsection 3.3.4.

In order to demonstrate the effectiveness of the suggested metric as an objective function in the modelling of pedestrian-path agents (i.e. customer agents in this case), the metric is integrated with a simplified version of hill-climbing optimization, which has been discussed in Section 3.2.5. The details of this implementation can be found in subsection 3.4.2.

Although the theoretical contribution shows promises in providing efficient guides towards more believable path-traversing agents, the implementation does not fulfill the expectation according to experimental results in chapter 4. It is concluded in subsection 4.2 that by using the proposed procedure, there is little evidence to show that agents' parameters can be optimized so that the agents can express pedestrian behavior patterns that match the provided raw data. The major issues found in the implementation include the non-determinism of performance scores, poor choice of optimizing parameters, and inflexible nature of the basic implemented agent in choosing its pedestrian paths.

Despite the modest result in the experimental part, I hope that my contribution on mathematical and theoretical justification can inspire future researchers in the field to explore the presented techniques and to solve the experimental technical problems.

5.2 Scope for Future Research

Possible extensions from the work in this thesis can be categorized into three major problems:

1. **Integrating rich interactive behavior in agents:** Because the current analysis of the retail environment in this thesis is solely focused on the pedestrian paths, the implemented agents are not programmed to accommodate a lot of inter-agent interaction. While this decision helps control the complexity of the optimization problem, it also limits agent-based modeling's potential to produce emergent behavior. Therefore, it is suggested that more interaction should be integrated into behavior of agents in the future.

2. **Eliminating noise in performance score calculation:** The non-determinism in path generation makes it hard to evaluate the parameter sets because of the noise in performance scores. Therefore, although the basic idea of objective function score calculation from the frequency matrix of discretized paths is an effective metric in theory

(as justified in subsection 3.3.4), further research should be pursued to eliminate the noise in performance score. One way is to increase the samples of paths used in the score calculation and find the average of the scores from these paths. However, the path-generating simulation process is time-consuming. Therefore, it is a solution that this thesis chooses to ignore because of time limitation.

3. **Making better choices of optimizing parameters:** It is suggested in the experiment that the objective function over the selected parameters is flat. It is possible that the current set of parameters in the optimization process is not very influential in altering agents' choice of paths. A possible remedy is choosing parameters which are more explicitly related to the choices of pedestrian path parameters, such as parameters that control the Navigating Graph.

Appendix A

Tables

Table A.1: Statistics of Performance Scores Using Standard Initialized Parameters and 200 Star Graph Nodes Under Various Time Steps

	T = 1000	T = 2000	T = 3000
Mean	0.065637426706098	0.0987263832774932	0.115931961150999
S.D.	0.00170925874464745	0.0028392939346693	0.00299971084549036
Max	0.0704458415960401	0.105320225040242	0.121631478777979
Min	0.0611009287810288	0.0882505572363714	0.106130106945505

	T = 4000	T = 5000	T = 6000
Mean	0.127478695026242	0.142245401598197	0.15122840296342
S.D.	0.00405316672479552	0.00484382720250297	0.00579232782647909
Max	0.13937536598925	0.157789704126659	0.169963680768869
Min	0.11785941031681	0.128510338872295	0.138728048325397

	T = 7000	T = 8000	T = 9000	T = 10000
Mean	0.159249154109357	0.162488734560373	0.166366571191929	0.169204992673754
S.D.	0.00730641442213909	0.00893136279624332	0.00991700145532484	0.00941689463064831
Max	0.179944156478344	0.200469692691589	0.195637644401238	0.196772161584025
Min	0.145647604040419	0.146195958421031	0.146087269616482	0.149039612561075

Table A.2: Parameter Description and Status

	Parameters	Intended Functionality	Expected Effect on Pedestrian Movement if Increased	Status
1	navMulTarget_CLOSE_THRESHOLD	threshold for object being "close" to agent (in obstacle avoidance)	Agents respond to avoid obstacles at further distance	Active
2	navMulTarget_DEF_ANGLE_DAMP	base value of agent's angular rotation damping	Slow down changes in overall agents' angular velocity	Active
3	navMulTarget_DEF_AT_TARGET_THRES	threshold distance for agent to "arrive at" its target	Agents stop further away from their intended targets	Active
4	navMulTarget_DEF_INTERPERSON_DIST	base value of interpersonal distance between agent	Dummy variable	Dummy
5	navMulTarget_DEF_MASS	base value of agent's mass	Slow down changes in overall agents' speed	Active
6	navMulTarget_DEF_MOMENT	base value of agent's moment of inertia	Slow down changes in overall angular velocity	Active
7	navMulTarget_DEF_SPEED_DAMP	base value of agent's speed damping	Slow down changes in speed over the whole population	Active
8	navMulTarget_SIGMA_ANGLE_DAMP	factor of variation of agent's angular rotation damping	Slows down the changes in agents' angular velocity	Active
9	navMulTarget_SIGMA_MASS	factor of variation of agent's mass	Slows down the changes in agents' speed	Active
10	navMulTarget_SIGMA_MOMENT	factor of variation of agent's moment of inertia	Slows down the changes in agents' angular velocity	Active
11	navMulTarget_SIGMA_SPEED_DAMP	factor of variation of agent's speed damping	Slows down the changes in agents' speed	Active
12	person_agentRelFollowLocX	"eye" location relative the agent in X direction	No direct effect & proper values allow good occlusion checking	Active
13	person_agentRelFollowLocY	"eye" location relative the agent in Y direction	No direct effect & proper values allow good occlusion checking	Active
14	person_agentRelFollowLocZ	"eye" location relative the agent in Z direction	No direct effect & proper values allow good occlusion checking	Active
15	person_isovistLocRelAvatarX	location of isovist sensor relative to agent's location in X direction	Dummy variable	Dummy
16	person_isovistLocRelAvatarY	location of isovist sensor relative to agent's location in Y direction	Dummy variable	Dummy
17	person_isovistLocRelAvatarZ	location of isovist sensor relative to agent's location in Z direction	Dummy variable	Dummy
18	person_sonarLocRelActorX	the center of LIDAR unit relative to the agent in X direction	No direct effect & proper values allow good paths into open-space.	Active
19	person_sonarLocRelActorY	the center of LIDAR unit relative to the agent in Y direction	No direct effect & proper values allow good paths into open-space.	Active
20	person_sonarLocRelActorZ	the center of LIDAR unit relative to the agent in Z direction	No direct effect & proper values allow good paths into open-space.	Active
21	queueTeller_nearThres	threshold between agent and its target to consider "near"	Agents stop further from the queue end when seeking for queue	Active
22	queueTeller_queueInterpersonDist	distance between agents in a queue	The arrangement in queue line will become looser	Active
23	queueTeller_timeToTarget	time allowed before agent is "lost" after exiting a queue to find target	More agents wander off intended paths to their target	Active
24	queueTeller_transactTimeATMCore	base time of agent's transaction at the ATM	Longer overall time that agents spend at ATM	Active
25	queueTeller_transactTimeATMVar	variable time of agent's transaction at the ATM	Larger variation of time that agents spend at ATM	Active
26	queueTeller_transactTimeOtherCore	base time of agent's transaction with other types of bank staff	Dummy variable	Dummy
27	queueTeller_transactTimeOtherVar	variable time of agent's transaction with other types of bank staff	Dummy variable	Dummy
28	queueTeller_transactTimeTellerCore	base time of agent's transaction with tellers	Longer overall time that agents have to spend with tellers	Active
29	queueTeller_transactTimeTellerVar	variable time of agent's transaction with tellers	Larger variation of time that agents spend with tellers	Active
30	queueTeller_velFactor	slow-down factor for agent before reaching a queue	Agents slow down faster as the variable gets higher	Active
31	queueTeller_veryNearThres	threshold between agent and its target to consider "very near"	Agents stop further away from their intended targets	Active

Table A.3: Variations of Parameters Initialization Values Used in the Experiments

	Standard Parameters	Random Parameters 1	Random Parameters 2	Random Parameters 3
navMulTarget_CLOSE_THRESHOLD	500	342.669701891	527.547835427	440.871139793
navMulTarget_DEF_ANGLE_DAMP	0.8	1.16184659517	0.392206859494	0.848121450044
navMulTarget_DEF_AT_TARGET_THRES	250	121.901578294	208.922458937	323.725251828
navMulTarget_DEF_INTERPERSON_DIST	500	511.91555112	572.149322521	328.817689932
navMulTarget_DEF_MASS	5	7.77740179306	4.26137711343	3.12713354039
navMulTarget_DEF_MOMENT	0.1	0.0958313264063	0.140176064684	0.0621819945328
navMulTarget_DEF_SPEED_DAMP	3.2	2.15282779671	3.24067047072	3.64556617317
navMulTarget_SIGMA_ANGLE_DAMP	0	0.148832236128	0.176212343321	0.0627019675977
navMulTarget_SIGMA_MASS	0.1	0.393374534053	0.0342197582666	0.192223438694
navMulTarget_SIGMA_MOMENT	0	0.35475585072	0.216529889885	0.181945859417
navMulTarget_SIGMA_SPEED_DAMP	0.1	0.339913927809	0.0718234935795	0.239552732787
person_agentRelFollowLocX	0	0.358907568411	0.0920284351062	0.132699647139
person_agentRelFollowLocY	0	0.221142517169	0.355144163747	0.0295351194864
person_agentRelFollowLocZ	820	534.179574078	564.355327892	1019.99764318
person_isovistLocRelAvatarX	0	0.352204755889	0.146439821016	0.362322723689
person_isovistLocRelAvatarY	0	0.0136723579827	0.20701685156	0.349244038516
person_isovistLocRelAvatarZ	-100	-71.4215412709	-139.181635535	-129.723091824
person_sonarLocRelActorX	0	0.372549251551	0.113600484665	0.33786441999
person_sonarLocRelActorY	0	0.347319402585	0.246362486824	0.10064768008
person_sonarLocRelActorZ	-100	-71.988144684	-59.9742749862	-79.3447776416
queueTeller_nearThres	2000	1585.48684893	1344.94317219	1546.44135038
queueTeller_queueInterpersonDist	500	624.666696713	382.008636303	353.985767466
queueTeller_timeToTarget	30	38.5835298687	36.9902387764	20.2973157431
queueTeller_transactTimeATMCore	50	63.9890158958	63.3675348153	63.4945226765
queueTeller_transactTimeATMVar	20	21.6822960577	15.7832233271	24.9288581472
queueTeller_transactTimeOtherCore	10	12.6407349092	5.60056396592	7.70354707791
queueTeller_transactTimeOtherVar	20	12.0167712379	17.5581429096	25.683839606
queueTeller_transactTimeTellerCore	120	72.1383262992	142.247377109	155.838323068
queueTeller_transactTimeTellerVar	20	17.1220358955	27.1536901243	26.9110239287
queueTeller_VelFactor	0.4	0.200403336626	0.245108548249	0.418294208033
queueTeller_veryNearThres	100	121.32370312	74.310594461	89.9691478968

Table A.4: Optimization Result with 1000 to 5000 Time Steps Under Standard Initial Parameters and 200 Star Graph Nodes

Parameters	**Original**	**T = 1000**	**T = 2000**	**T = 3000**	**T = 4000**	**T = 5000**
navMulTarget_CLOSE_THRESHOLD	500	500	500	500	500	500
navMulTarget_DEF_ANGLE_DAMP	0.8	0.8	0.8	0.8	0.8	0.8
navMulTarget_DEF_AT_TARGET_THRES	250	250	250	250	250	250
navMulTarget_DEF_INTERPERSON_DIST	500	500	500	500	500	495
navMulTarget_DEF_MASS	5	5	5	5	5	5
navMulTarget_DEF_MOMENT	0.1	0.1	0.1	0.1	0.1	0.1
navMulTarget_DEF_SPEED_DAMP	3.2	3.2	3.2	3.2	3.2	3.2
navMulTarget_SIGMA_ANGLE_DAMP	0	0	0	0	0	0
navMulTarget_SIGMA_MASS	0.1	0.1	0.099	0.1	0.1	0.1
navMulTarget_SIGMA_MOMENT	0	0	0	0	0	0
navMulTarget_SIGMA_SPEED_DAMP	0.1	0.1	0.1	0.1	0.1	0.1
person_agentRelFollowLocX	0	0	0	0	0	0
person_agentRelFollowLocY	0	0	0	0	0	0
person_agentRelFollowLocZ	820	820	820	820	820	820
person_isovistLocRelAvatarX	0	0	0	0	0	0
person_isovistLocRelAvatarY	0	0	0	0	0	0
person_isovistLocRelAvatarZ	-100	-100	-100	-100	-100	-100
person_sonarLocRelActorX	0	0	0	0	0	0
person_sonarLocRelActorY	0	0	0	0	0	0
person_sonarLocRelActorZ	-100	-100	-100	-100	-100	-100
queueTeller_nearThres	2000	2000	2000	2000	2000	2000
queueTeller_queueInterpersonDist	500	500	500	500	500	500
queueTeller_timeToTarget	30	30	30	30	30	30
queueTeller_transactTimeATMCore	50	50	50	50	50	50
queueTeller_transactTimeATMVar	20	20	20	20	20	20
queueTeller_transactTimeOtherCore	10	10	10	10	10	10
queueTeller_transactTimeOtherVar	20	20	20	20	20	20
queueTeller_transactTimeTellerCore	120	120	120	120	120	120
queueTeller_transactTimeTellerVar	20	20	20	20	20	20
queueTeller_velFactor	0.4	0.4	0.4	0.404	0.4	0.4
queueTeller_veryNearThres	100	100	100	100	100	100
Number of iterations	-	2	4	3	3	4
Time used to train	-	2253.545561313	6348.892802853	10477.020265356	9189.674513252	15413.730268519
Average time per iteration	-	1126.7727806565	1587.22320071325	3492.340088452	3063.22483775067	3853.43256712975
Initial performance score	-	0.0674719741581682	0.0975707875259275	0.120418310415735	0.129883488206964	0.150926008333448
Final performance score	-	0.0708800594586844	0.106850562124662	0.12291235522072	0.143310358355787	0.158386816247695
Number of tracks	-	11	16	23	34	42

Table A.5: Optimization Result with 6000 to 10000 Time Steps Under Standard Initial Parameters and 200 Star Graph Nodes

Parameters	**Original**	**T = 6000**	**T = 7000**	**T = 8000**	**T = 9000**	**T = 10000**
navMulTarget_CLOSE_THRESHOLD	500	500	500	500	500	500
navMulTarget_DEF_ANGLE_DAMP	0.8	0.8	0.808	0.8	0.8	0.8
navMulTarget_DEF_AT_TARGET_THRES	250	250	250	250	250	250
navMulTarget_DEF_INTERPERSON_DIST	500	500	500	500	505	500
navMulTarget_DEF_MASS	5	5	5	5	5	5
navMulTarget_DEF_MOMENT	0.1	0.1	0.1	0.1	0.1	0.1
navMulTarget_DEF_SPEED_DAMP	3.2	3.2	3.2	3.2	3.2	3.2
navMulTarget_SIGMA_ANGLE_DAMP	0	0	0	0	0	0
navMulTarget_SIGMA_MASS	0.1	0.1	0.1	0.1	0.101	0.1
navMulTarget_SIGMA_MOMENT	0	0	0	0	0	0
navMulTarget_SIGMA_SPEED_DAMP	0.1	0.1	0.1	0.1	0.1	0.1
person_agentRelFollowLocX	0	0	0	0	0	0
person_agentRelFollowLocY	0	0	0	0	0	0
person_agentRelFollowLocZ	820	820	820	820	820	828.2
person_isovistLocRelAvatarX	0	0	0	0	0	0
person_isovistLocRelAvatarY	0	0	0	0	0	0
person_isovistLocRelAvatarZ	-100	-100	-100	-100	-100	-100
person_sonarLocRelActorX	0	0	0	0	0	0
person_sonarLocRelActorY	0	0	0	0	0	0
person_sonarLocRelActorZ	-100	-100	-101	-100	-100	-100
queueTeller_nearThres	2000	2000	2000	2020	2000	2000
queueTeller_queueInterpersonDist	500	500	500	500	500	500
queueTeller_timeToTarget	30	30	30	30	30	30
queueTeller_transactTimeATMCore	50	50	50	50.5	50	49.5
queueTeller_transactTimeATMVar	20	20	20	20	20	20
queueTeller_transactTimeOtherCore	10	10	10	10.1	10	10
queueTeller_transactTimeOtherVar	20	20	20	20.2	20	20
queueTeller_transactTimeTellerCore	120	120	120	120	120	120
queueTeller_transactTimeTellerVar	20	19.8	20	20	20	20
queueTeller_velFactor	0.4	0.4	0.4	0.404	0.4	0.4
queueTeller_veryNearThres	100	100	100	100	101	100
Number of iterations	-	3	5	3	2	2
Time used to train	-	16494.255876159	44960.797866691	22784.845250313	27104.920496398	23361.860545549
Average time per iteration	-	5498.085292053	8992.1595733382	7594.948416771	13552.460248199	11680.9302727745
Initial performance score	-	0.157983975940682	0.168247028794193	0.166222498758133	0.162966152567443	0.189541708918554
Final performance score	-	0.174735511786074	0.187412219592389	0.194371749083499	0.206161579546764	0.199140833532435
Number of tracks	-	51	58	66	71	81

Table A.6: The Optimized Parameters of the Randomized Initialization Set 1 over 5000 Feedback Time Steps

Parameters	Original (Rand1)	Optimized with T=5000
navMulTarget_CLOSE_THRESHOLD	342.669701891	342.669701891
navMulTarget_DEF_ANGLE_DAMP	1.16184659517	1.16184659517
navMulTarget_DEF_AT_TARGET_THRES	121.901578294	121.901578294
navMulTarget_DEF_INTERPERSON_DIST	511.91555112	511.91555112
navMulTarget_DEF_MASS	7.77740179306	7.77740179306
navMulTarget_DEF_MOMENT	0.0958313264063	0.0958313264063
navMulTarget_DEF_SPEED_DAMP	2.15282779671	2.15282779671
navMulTarget_SIGMA_ANGLE_DAMP	0.148832236128	0.148832236128
navMulTarget_SIGMA_MASS	0.393374534053	0.393374534053
navMulTarget_SIGMA_MOMENT	0.35475585072	0.35475585072
navMulTarget_SIGMA_SPEED_DAMP	0.339913927809	0.339913927809
person_agentRelFollowLocX	0.358907568411	0.358907568411
person_agentRelFollowLocY	0.221142517169	0.221142517169
person_agentRelFollowLocZ	534.179574078	534.179574078
person_isovistLocRelAvatarX	0.352204755889	0.352204755889
person_isovistLocRelAvatarY	0.0136723579827	0.0136723579827
person_isovistLocRelAvatarZ	-71.4215412709	-71.4215412709
person_sonarLocRelActorX	0.372549251551	0.372549251551
person_sonarLocRelActorY	0.347319402585	0.347319402585
person_sonarLocRelActorZ	-71.988144684	-71.988144684
queueTeller_nearThres	1585.48684893	1585.48684893
queueTeller_queueInterpersonDist	624.666696713	624.666696713
queueTeller_timeToTarget	38.5835298687	38.5835298687
queueTeller_transactTimeATMCore	63.9890158958	63.349125736842
queueTeller_transactTimeATMVar	21.6822960577	21.6822960577
queueTeller_transactTimeOtherCore	12.6407349092	12.6407349092
queueTeller_transactTimeOtherVar	12.0167712379	12.0167712379
queueTeller_transactTimeTellerCore	72.1383262992	72.1383262992
queueTeller_transactTimeTellerVar	17.1220358955	17.1220358955
queueTeller_VelFactor	0.200403336626	0.200403336626
queueTeller_veryNearThres	121.32370312	121.32370312
Number of iterations	-	2
Time used to train	-	7568.649534791
Average time per iteration	-	3784.3247673955
Initial similarity score	-	0.153744179865683
Final similarity score	-	0.155948927418452
Number of tracks	-	42

Table A.7: The Optimized Parameters of the Randomized Initialization Set 2 over 5000 Feedback Time Steps

Parameters	Original (Rand2)	Optimized with T=5000
navMulTarget_CLOSE_THRESHOLD	527.547835427	527.547835427
navMulTarget_DEF_ANGLE_DAMP	0.392206859494	0.392206859494
navMulTarget_DEF_AT_TARGET_THRES	208.922458937	208.922458937
navMulTarget_DEF_INTERPERSON_DIST	572.149322521	572.149322521
navMulTarget_DEF_MASS	4.26137711343	4.26137711343
navMulTarget_DEF_MOMENT	0.140176064684	0.140176064684
navMulTarget_DEF_SPEED_DAMP	3.24067047072	3.24067047072
navMulTarget_SIGMA_ANGLE_DAMP	0.176212343321	0.176212343321
navMulTarget_SIGMA_MASS	0.0342197582666	0.0342197582666
navMulTarget_SIGMA_MOMENT	0.216529889885	0.216529889885
navMulTarget_SIGMA_SPEED_DAMP	0.0718234935795	0.0718234935795
person_agentRelFollowLocX	0.0920284351062	0.0920284351062
person_agentRelFollowLocY	0.355144163747	0.355144163747
person_agentRelFollowLocZ	564.355327892	564.355327892
person_isovistLocRelAvatarX	0.146439821016	0.146439821016
person_isovistLocRelAvatarY	0.20701685156	0.20701685156
person_isovistLocRelAvatarZ	-139.181635535	-139.181635535
person_sonarLocRelActorX	0.113600484665	0.113600484665
person_sonarLocRelActorY	0.246362486824	0.24882611169224
person_sonarLocRelActorZ	-59.9742749862	-59.9742749862
queueTeller_nearThres	1344.94317219	1344.94317219
queueTeller_queueInterpersonDist	382.008636303	382.008636303
queueTeller_timeToTarget	36.9902387764	36.9902387764
queueTeller_transactTimeATMCore	63.3675348153	63.3675348153
queueTeller_transactTimeATMVar	15.7832233271	15.7832233271
queueTeller_transactTimeOtherCore	5.60056396592	5.60056396592
queueTeller_transactTimeOtherVar	17.5581429096	17.5581429096
queueTeller_transactTimeTellerCore	142.247377109	142.247377109
queueTeller_transactTimeTellerVar	27.1536901243	27.1536901243
queueTeller_VelFactor	0.245108548249	0.245108548249
queueTeller_veryNearThres	74.310594461	74.310594461
Number of iterations	-	1
Time used to train	-	3684.186147125
Average time per iteration	-	3684.186147125
Initial similarity score	-	0.149738522577116
Final similarity score	-	0.149738522577116
Number of tracks	-	42

Table A.8: The Optimized Parameters of the Randomized Initialization Set 3 over 5000 Feedback Time Steps

Parameters	Original (Rand3)	Optimized with T=5000
navMulTarget_CLOSE_THRESHOLD	440.871139793	440.871139793
navMulTarget_DEF_ANGLE_DAMP	0.848121450044	0.85660266454444
navMulTarget_DEF_AT_TARGET_THRES	323.725251828	323.725251828
navMulTarget_DEF_INTERPERSON_DIST	328.817689932	328.817689932
navMulTarget_DEF_MASS	3.12713354039	3.12713354039
navMulTarget_DEF_MOMENT	0.0621819945328	0.0621819945328
navMulTarget_DEF_SPEED_DAMP	3.64556617317	3.64556617317
navMulTarget_SIGMA_ANGLE_DAMP	0.0627019675977	0.0627019675977
navMulTarget_SIGMA_MASS	0.192223438694	0.192223438694
navMulTarget_SIGMA_MOMENT	0.181945859417	0.181945859417
navMulTarget_SIGMA_SPEED_DAMP	0.239552732787	0.239552732787
person_agentRelFollowLocX	0.132699647139	0.132699647139
person_agentRelFollowLocY	0.0295351194864	0.0295351194864
person_agentRelFollowLocZ	1019.99764318	1019.99764318
person_isovistLocRelAvatarX	0.362322723689	0.362322723689
person_isovistLocRelAvatarY	0.349244038516	0.35273647890116
person_isovistLocRelAvatarZ	-129.723091824	-129.723091824
person_sonarLocRelActorX	0.33786441999	0.33786441999
person_sonarLocRelActorY	0.10064768008	0.10064768008
person_sonarLocRelActorZ	-79.3447776416	-79.3447776416
queueTeller_nearThres	1546.44135038	1546.44135038
queueTeller_queueInterpersonDist	353.985767466	353.985767466
queueTeller_timeToTarget	20.2973157431	20.2973157431
queueTeller_transactTimeATMCore	63.4945226765	64.129467903265
queueTeller_transactTimeATMVar	24.9288581472	24.9288581472
queueTeller_transactTimeOtherCore	7.70354707791	7.70354707791
queueTeller_transactTimeOtherVar	25.683839606	25.683839606
queueTeller_transactTimeTellerCore	155.838323068	155.838323068
queueTeller_transactTimeTellerVar	26.9110239287	26.9110239287
queueTeller_VelFactor	0.418294208033	0.418294208033
queueTeller_veryNearThres	89.9691478968	89.9691478968
Number of iterations	-	2
Time used to train	-	7987.24616522
Average time per iteration	-	3993.62308261
Initial similarity score	-	0.145682295911969
Final similarity score	-	0.162538609301026
Number of tracks	-	42

Appendix B

Figures

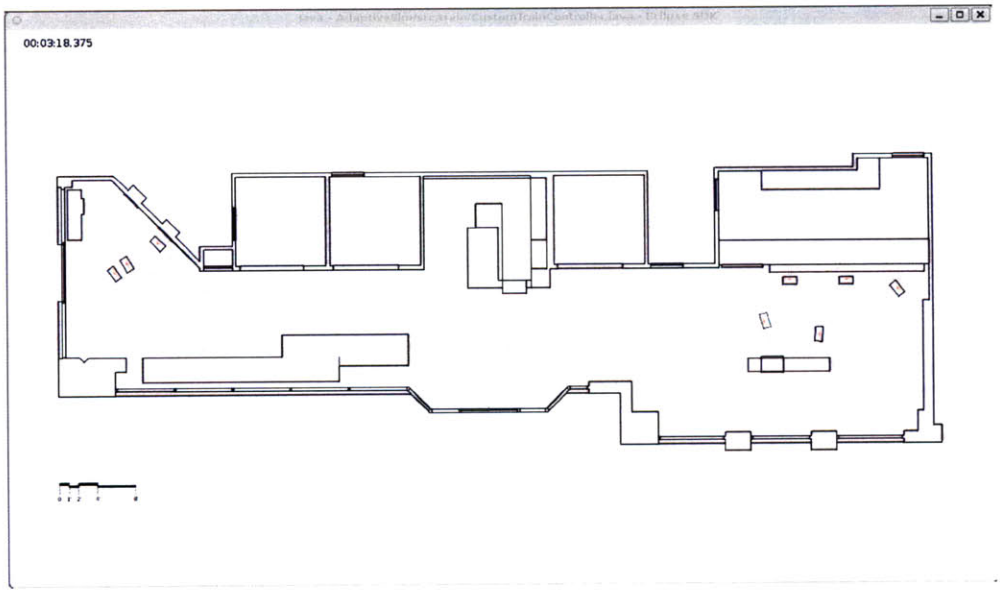


Figure B-1: Screenshot of the simulation using standard parameter initialization at time step 1500

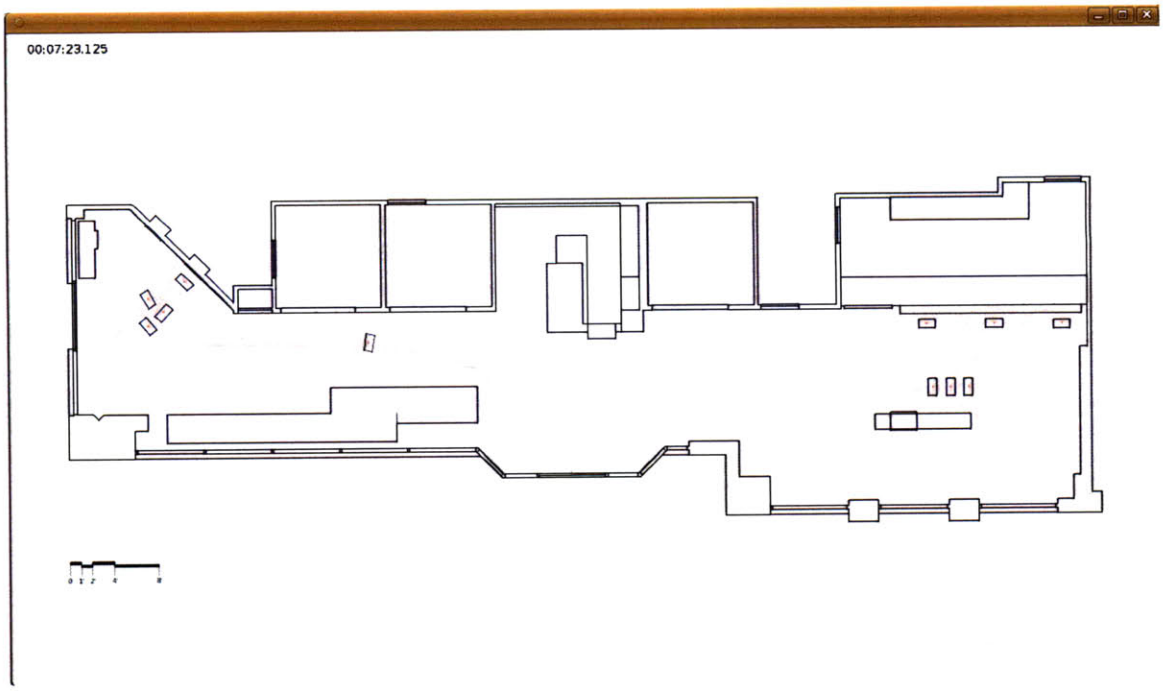


Figure B-2: Screenshot of the simulation using standard parameter initialization at time step 3500

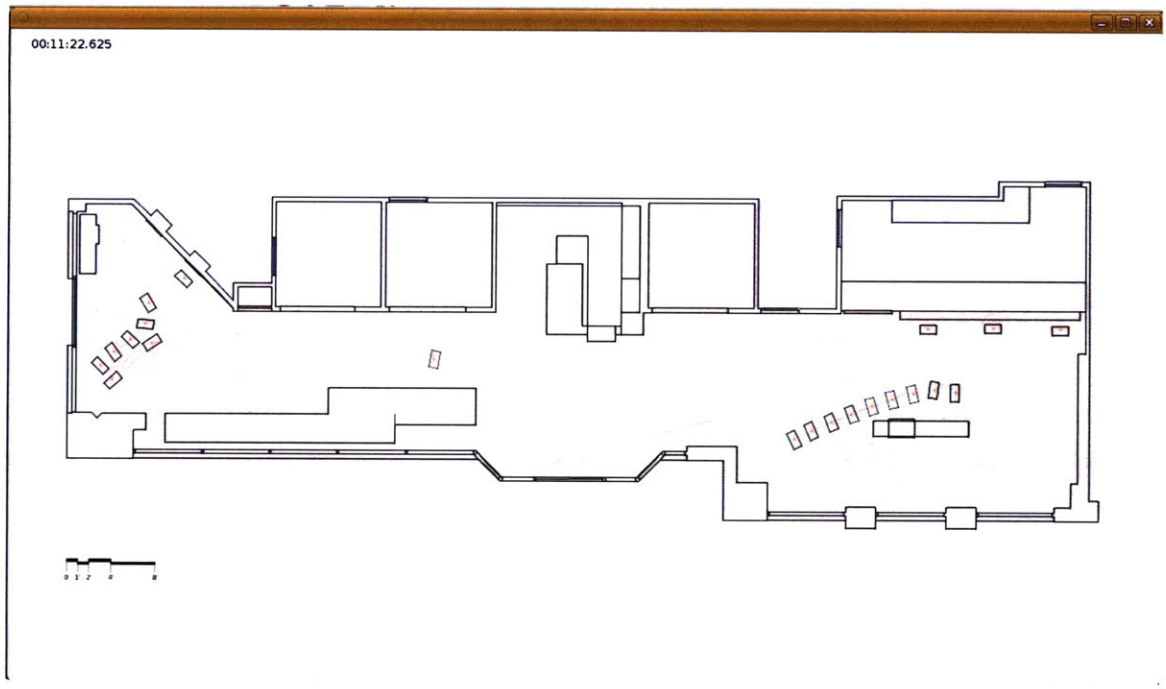


Figure B-3: Screenshot of the simulation using standard parameter initialization at time step 5500

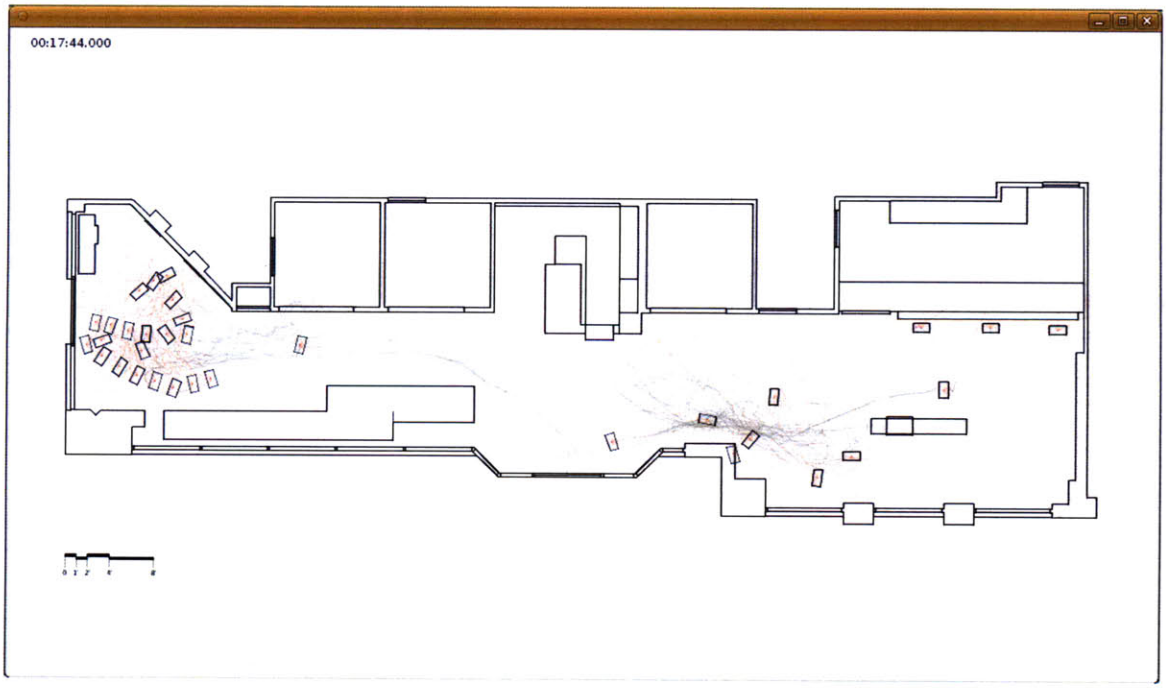


Figure B-4: Screenshot of the simulation using standard parameter initialization at time step 8500

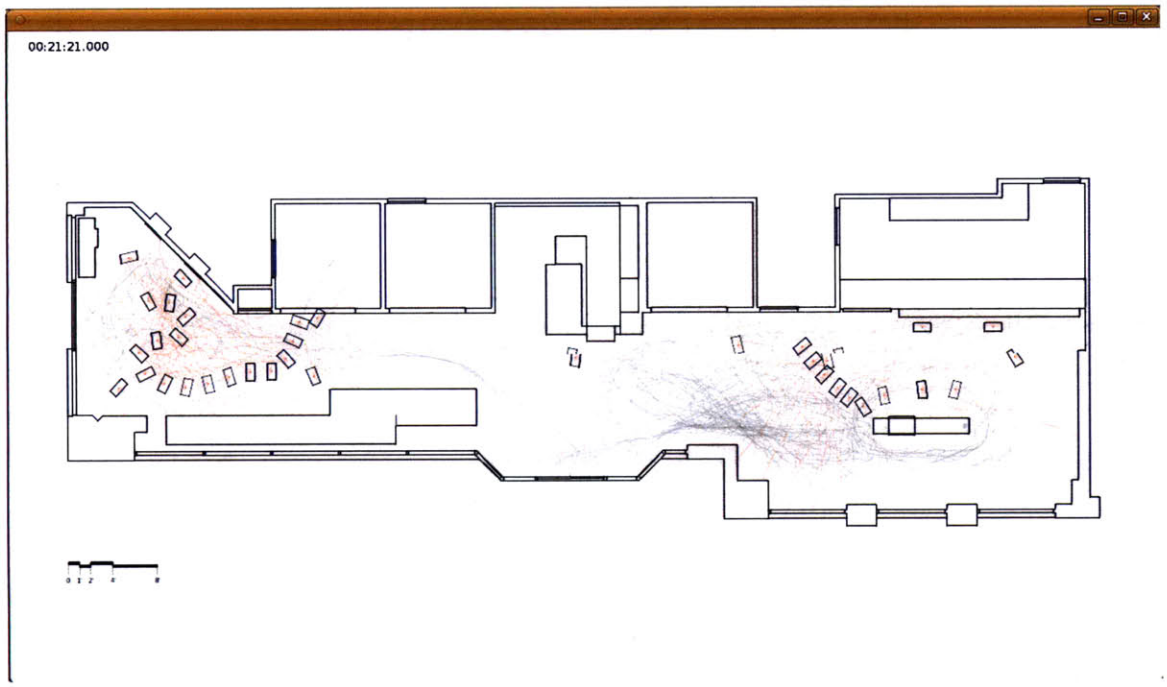


Figure B-5: Screenshot of the simulation using standard parameter initialization at time step 10500

Appendix C

Algorithms

Algorithm 1 Parameter Optimization Algorithm

Require: $(maxIter \geq 1) \wedge (threshold \geq 0) \wedge (\delta > 0) \wedge (nodeNum > 0)$

```
1: maxIter ← The maximum number of optimization iterations
2: paramSet ← A set of simulating parameters
3: threshold ← A convergence threshold
4: δ ← Amount of adjustment on parameters allowed in each iteration
5: nodeNum ← The number of correlating graph nodes
6: P ← permute(paramSet)
7: prevP ← P
8: for n = 1 to nodeSetNum do
9:    nodes[n] ← randomNode(nodeNum)
10: end for
11: bestScore ← calcScore(rawTracks, nodes, P)
12: prevScore ← bestScore
13: for i = 1 to maxIter do
14:    for j = 1 to paramNum do
15:       curr ← P[j]
16:       cand[0] ← curr
17:       cand[1] ← (1 + δ)× curr
18:       cand[2] ← (1 − δ)× curr
19:       best ← curr
20:       for k = 0 to 2 do
21:          Q ← P
22:          Q[j] ← cand[k]
23:          score ← calcScore(rawTracks, nodes, Q)
24:          if score > bestScore then
25:             best ← cand[k]
26:             bestScore ← score
27:          end if
28:       end for
29:       P[j] ← best
30:    end for
31:    if converged(P, prevP, prevScore, bestScore, threshold) then
32:       return P
33:    end if
34:    prevScore ← bestScore
35:    prevP ← P
36: end for
```

Algorithm 2 Performance Score Calculating Algorithm

```
1: dataTracks ← Raw data tracks
2: paramSet ← A set of simulation parameters
3: nodeSets ← A set of node lists for track correlation
4: tracks ← simulate(paramSet)
5: scoreSum ← 0
6: for i=0 to size(nodeSets)-1 do
7:    subscore ← 0
8:    nodes ← nodeSets[i]
9:    dataFreq ← freqMatrix(dataTracks, nodes)
10:   simFreq ← freqMatrix(tracks, nodes)
11:   for j=0 to size(nodes)-1 do
12:      for k=0 to size(nodes)-1 do
13:         dataEntry ← dataFreq[j][k]
14:         simEntry ← simFreq[j][k]
15:         subscore ← subscore + divergence(dataEntry,simEntry)
16:      end for
17:   end for
18:   subscore ← subscore / (size(nodes))^2
19:   score ← score + subscore
20: end for
21: score ← scoreSum / size(nodeSets)
22: return score
```

Algorithm 3 Kullback-Liebler Divergence

Require: $(0 \leq entry1 \leq 1) \wedge (0 \leq entry2 \leq 1)$

```
1: entry1 ← The first value
2: entry2 ← The second value
3: ε ← very small constant
4: entry1 ← entry1 + ε
5: entry2 ← entry2 + ε
6: firstTerm ← entry1/(entry1 + entry2)
7: firstTerm ← firstTerm × ln(entry1/entry2)
8: secondTerm ← entry2/(entry1 + entry2
9: secondTerm ← secondTerm × ln(entry2/entry1)
10: basicDiv ← firstTerm + secondTerm
11: divergence ← exp^(−basicDiv)
```

Algorithm 4 Frequency Matrix Algorithm

```
1: tracks ← A list of tracks
2: nodes ← A list of graph nodes for track correlation
3: for i = 0 to (size(nodes)-1) do
4:    for j = 0 to (size(nodes)-1) do
5:       freq[i][j] ← 0
6:    end for
7: end for
8: segCount ← 0
9: for t = 0 to (size(tracks)-1) do
10:   corrSegs ← correlateGraph(tracks[t], nodes)
11:   for s = 0 to (size(corrSegs)-1) do
12:      beginNode ← begin(corrSegs[q])
13:      endNode ← begin(corrSegs[q])
14:      freq[beginNode][endNode] ← freq[beginNode][endNode] + 1
15:      segCount ← segCount + 1
16:   end for
17: end for
18: for i = 0 to (size(nodes)-1) do
19:   for j = 0 to (size(nodes)-1) do
20:      freq[i][j] ← (freq[i][j] / segCount)
21:   end for
22: end for
23: return freq
```

Algorithm 5 Convergence Checking Algorithm

Require: $(threshold > 0)$

```
1: currScore ← The score from the current iteration
2: prevScore ← The score from the previous iteration
3: param ← The set of parameters used in the current iteration
4: prevParam ← The set of parameters used in the previous iteration
5: threshold ← The convergence threshold
6: improveRatio ← (currScore - prevScore) / prevScore
7: converge ← (param == prevParam) ∨ (improveRatio < threshold)
8: return converge
```

Bibliography

[1] Julie Baker, Dhruv Grewal, and A. Parasuraman. The influence of store environment on quality inferences and store image. *Journal of the Academy of Marketing Science*, 22(4):328–339, Fall 1994.

[2] Julie Baker, Michael Levy, and Dhruv Grewal. An experimental approach to making retail store environment decisions. *Journal of Retailing*, 68(4):445–460, Winter 1992.

[3] Julie Baker, A. Parasuraman, Dhruv Grewal, and Glenn B. Voss. The influence of multiple store environment cues on perceived merchandise value and patronage intention. *Journal of Marketing*, 66(2):120–141, April 2002.

[4] J. Berrou, J. Beechan, P. Quaglia, and A. Gerodimos. Calibration and validation of the Legion simulation model using empirical data. In *Proc. Pedestrian and Evacuation Dynamics*, pages 167–181, Vienna, 2005.

[5] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *PNAS*, 2002.

[6] P. Bovet and S. Benhamou. Spatial analysis of animals movements using a correlated random walk model. *Journal of Theoretical Biology*, 131:419433, 1988.

[7] Remo Burkhard, Stefan Bischof, and Andres Herzog. The potential of crowd simulations for communication purposes in architecture. In *Proc. 12th International Conference Information Visualization, 2008*, pages 403–408, 2008.

[8] John A. Byers. Correlated random walk equations of animal dispersal resolved by simulation. *Ecology*, 82(6):1680–1690, 2001.

[9] John Casti. Bizsim: the world of businessin a box. *Artificial Life and Robotics*, 4:125–129, 2000. 10.1007/BF02481332.

[10] Nicolas Courty and Thomas Corpetti. Data-driven animation of crowds. In *Computer Vision/Computer Graphics Collaboration Techniques*, volume 4418/2007 of *Lecture Notes in Computer Science*, pages 377–388. Springer, Berlin, 2007.

[11] Dhruv Grewal and Julie Baker. Do retail store environmental factors affect consumers' price acceptability? An empirical examination. *International Journal of Research in Marketing*, 11(2):107–115, 1994.

[12] M. Gunther, C. Stummer, L. M. Wakolbinger, and M Wildpaner. An agent-based simulation approach for the new product diffusion of a novel biomass fuel. *Journal of the operational Research Society*, 2010.

[13] Dirk Helbing, Illes Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407, September 2000.

[14] Dirk Helbing, Illes J. Farkas, Peter Molnar, and Tamas Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. In M. Schreckenberg and S. D. Sharma, editors, *Pedestrian and Evacuation Dynamics*, pages 21–58. Springler, Berlin, 2002.

[15] Dirk Helbing, Peter Molnar, Illes J. Farkas, and Kai Bolay. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 28:361–383, 2001.

[16] P. M. Kareiva and N. Shigesada. Analyzing insect movement as a correlated random walk. *Occologia*, 56:234–238, 1983.

[17] Jon Kerridge, Julian Hine, and Marcus Wigan. Agent-based modelling of pedestrian movements: The questions that need to be asked and answered. *Environment and Planning B: Planning and Design 2001*, 28:327–341, 2001.

[18] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[19] Tei Laine and Jerome Busemeyer. Comparing agent-based learning models of land-use decision making. *Proceedings of Conference of North American Associstion for Computational Social and Organizational Science, Pittsburgh, PA, June 27-29*, 2004.

[20] Pierre Martineau. The personality of the retail store. *Harvard Business Review*, 36:47–55, January/February 1958.

[21] C. E. McCulloch and M. L. Cain. Analyzing discrete movement data as a correlated random walk. *Ecology*, 70(2):383–388, April 1989.

[22] T. Narahara. Enactment software: Spatial designs using agent-based model. In *Proc. Agent 2007: Conference on Complex Interaction and Social Emergence*. Argonne National Laboratory and Northwestern University, 2008.

[23] Taro Narahara. The space re-actor: Walking a synthetic man through architecture. Master's thesis, Massachusetts Institute of Technology, Department of Architecture, June 2007.

[24] Clifford S. Patlak. Random walk with persistence and external bias. *Bulletin of Mathematical Biophysics*, 15:311–338, 1953.

[25] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[26] George Shaw and Deb Roy. Star graphs: Discretization, visualization, and analysis of tracking data. Paper within group, 2009.

[27] Adrien Treuille, Seth Cooper, and Zoran Popovic. Continuum crowds. In *Proc. International Conference on Computer Graphics and Interactive Techniques*, pages 1160–1168, New York, NY, USA, 2006. ACM SIGGRAPH, ACM.

[28] L. W. Turley and Ronald E. Milliman. Atmospheric effects on shopping behavior: A review of the experimental evidence. *Journal of Business Research*, 49(2):193–211, 2000.

[29] Alasdair Turner. Partners in the street ballet: An embodied process of person-space coupling in the built environment. In *Proc. Translating Embodied Mind Approaches for the Next Decade.* The Barnard Interdisciplinary Workshop on Embodiment, July 2010.

[30] Alasdair Turner and Alan Penn. Encoding natural movement as an agent-based system: an investigation into human pedestrian behavior in the built environment. *Environment and Planning B: Planning and Design*, 29:473–490, 2002.

[31] Alasdair Turner and Alan Penn. Evolving direct perception models of human behavior in building systems. In N. Waldau, P. Gattermann, H. Knoflacher, and M. Schreckenberg, editors, *Pedestrian and Evacuation Dynamics 2005*, pages 411–422. Springer, Berlin, 2007.