12-15-2016

BUILDING OCCUPANCY SIMULATION AND DATA ASSIMILATION USING A GRAPH BASED AGENT ORIENTED MODEL

Sanish Rai

BUILDING OCCUPANCY SIMULATION AND DATA ASSIMILATION USING A GRAPH BASED AGENT ORIENTED MODEL

by

SANISH RAI

Under the Direction of Xiaolin Hu, PhD

ABSTRACT

Building occupancy simulation and estimation simulates the dynamics of occupants and estimates the real time spatial distribution of occupants in a building. It can benefit various applications like conserving energy, smart assist, building construction, crowd management, and emergency evacuation. Building occupancy simulation and estimation needs a simulation model and a data assimilation algorithm that assimilates real-time sensor data into the simulation model. Existing build occupancy simulation models include agent-based models and graph-based models. The agent-based models suffer high computation cost for simulating a large number occupants, and graph-based models overlook the heterogeneity and detailed behaviors of individuals. Recognizing the limitations of the existing models, in this dissertation, we combine the benefits of

agent and graph based modeling and develop a new graph based agent oriented model which can efficiently simulate a large number of occupants in various building structures. To support real-time occupancy dynamics estimation, we developed a data assimilation framework based on Sequential Monte Carol Methods, and apply it to the graph-based agent oriented model to assimilate real time sensor data. Experimental results show the effectiveness of the developed model and the data assimilation framework. The major contributions of this dissertation work include, 1) it provides an efficient model for building occupancy simulation which can accommodate thousands of occupants; 2) it provides an effective data assimilation framework for real-time estimation of building occupancy.

INDEX WORDS: Building Occupancy Simulation, Agent based, Graph based, Occupancy Dynamics Estimation, Data assimilation, Sequential Monte Carlo Methods.

BUILDING OCCUPANCY SIMULATION AND DATA ASSIMILATION USING A GRAPH BASED AGENT ORIENTED MODEL

by

SANISH RAI

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2016

BUILDING OCCUPANCY SIMULATION AND DATA ASSIMILATION USING A GRAPH BASED AGENT ORIENTED MODEL

by

SANISH RAI

Committee Chair: Xiaolin Hu

Committee: Rajshekar Sunderraman

Yichuan Zhao

Ying Zhu

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2016

DEDICATION

I dedicate this dissertation to my parents, Dilip Kumar Rai and Shakuntala Rai, for encouraging my ambitions and to my wife, Manjina, for supporting it so lovingly.

ACKNOWLEDGEMENTS

I want to express my gratitude to my advisor, Dr. Xiaolin Hu, for his continuous guidance, patience, and support. I want to thank him for believing in me and providing invaluable supervision throughout my time as a graduate student. His presence carries great significance in my work. I would also like to thank Dr. Raj Sunderraman for all his support and encouragement during my time at GSU. I express special appreciation toward my dissertation committee for their valuable feedback and suggestions at every step of my work. Also, I would like to thank my colleagues from SIMS lab, Yuan Long, Minghao Wang, Fan Bai, Haidong Xue, Peisheng Wu, and Nicholas Keller for creating a great academic environment and always engaging me in intelligent and thoughtful conversations from which many of my ideas have developed.

I want to thank my friends Pallabi Gupta and Ishaka Maskey for always encouraging me. I also want to acknowledge my brother Diwash Rai, my family members and friends for their goodwill and support.

Lastly, I would like to thank my wife, Manjina Shrestha, for providing immeasurable time, support, and love. Thank you for always being there for me.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ..... **viii**

LIST OF TABLES ..... **xii**

LIST OF FIGURES ..... **xiii**

1. Introduction **1**

1.1 Overview **1**

1.2 Background **4**

1.3 Problem Statement **7**

1.4 Agent based model **9**

1.5 Graph based model **11**

1.6 Sensors **12**

1.7 Data assimilation **14**

1.8 Sequential Monte Carlo methods **15**

1.9 Organization of the Dissertation **16**

2. Related work **18**

3. Graph-based Agent-oriented model **24**

3.1 General structure of the graph model **25**

3.2 Occupant Agents **29**

3.3 Agent's behavior **32**

3.4 Agent movement model **33**

3.5 Queue Processing 34

3.6 Discrete time-based simulation algorithm 34

3.7 Experimental results and analysis for the graph based agent oriented model 36

3.8 Graph based agent oriented model for a pedestrian tunnel simulation 41

3.9 Hybrid model 44

4. Data assimilation using graph based agent oriented model 48

4.1 Basic Theory 48

4.2 Data assimilation framework 52

4.3 Data assimilation using direct sensor data 58

4.4 Data assimilation experiments 63

4.4.1 Experiment settings 63

4.4.2 Results for smaller layout 68

4.4.3 Results for larger layout 77

4.4.4 Comparison of resampling using sensor data particles 86

5. Towards activity informed DDDS framework 90

5.1 Case Study: A Smart Office Environment 90

5.2 Activity informed DDDS framework 91

5.3 Agent based model for building occupancy 93

5.4 Behavior patterns of occupants in a building 96

5.5 Behavior Pattern Recognition using Coupled HMM 102

Conclusion 109

References 112

LIST OF TABLES

Table 3.1 Transition table for vertex 2 a) normal scenario b) evacuation to Node 4 27

Table 4.1 Experimental Design..... 67

Table 4.2 Average RMSE for nodes..... 70

Table 4.3 RMSE for using sensors in (a)half (b)1/3rd (c)1/5th of the nodes 78

Table 5.1 Initial probability for HMM..... 99

Table 5.2 Emission probability for HMM 99

Table 5.3 Average accuracy for behaviors 102

LIST OF FIGURES

Figure 1.1 Data assimilation based on PF methods 16

Figure 3.1. Framework for graph-based agent-oriented model 25

Figure 3.2. a) A building floor plan b) Graph representing the floor plan 26

Figure 3.3. Number of occupants (a) node 2 (b) node 4 37

Figure 3.4 (a) Occupants in safe node with varying link size (b) Occupants in safe node with varying agents knowing direction to safe node 38

Figure 3.5 Node 4 occupants with varying group size 39

Figure 3.6 Computational time for agents (a) with 100% agents knowing direction (b) with 50% agents knowing direction (c) agents with group size 20 40

Figure 3.7 Computational time for mixed agents 40

Figure 3.8. Pedestrians in a tunnel 42

Figure 3.9. 6000 Pedestrians (a) Input rate (b) Average flow rate including source (c) Flow rate at mid-section (d) Average flow rate without source 43

Figure 3.10. 2000 Pedestrians (a)Input rate (b) Average flow rate including source (c) Flow rate at mid-section (d)Average flow rate without source 44

Figure 3.11. Flow rate at exit end (a) 2000 Pedestrians (b) 6000 Pedestrians 44

Figure 3.12 Agents crossing paths 44

Figure 3.13 Hybrid mode conversion of agents 46

Figure 3.14 Screen shot of hybrid model simulation 47

Figure 4.1 Data assimilation framework 58

Figure 4.2 Issue caused by placement of sensors 59

Figure 4.3 Standard Resampling b). Resampling with particles from sensor data 61

Figure 4.4. Experimental structure for small layout building 65

Figure 4.5 Experimental building structure for larger layout 66

Figure 4.6 Real simulation at increasing time steps for first scenario 68

Figure 4.7 Data assimilation simulation at increasing time steps for first scenario 68

Figure 4.8 Comparing results for first scenario 69

Figure 4.9 Comparing results for second scenario..... 72

Figure 4.10 Comparing results for third scenario 73

Figure 4.11 Comparing results for fourth scenario 74

Figure 4.12 Comparing results for fifth scenario 75

Figure 4.13 Comparing results for sixth scenario 76

Figure 4.14 Comparing results for seventh scenario 77

Figure 4.15 Data assimilation results in larger layout using sensors in half of the total nodes 79

Figure 4.16 Data assimilation results in larger layout using sensors in 1/3rd of the total nodes.. 80

Figure 4.17 Data assimilation results in larger layout using sensors in 1/5th of the total nodes.. 81

Figure 4.18 Data assimilation results using sensors in half of the nodes 83

Figure 4.19 Data assimilation results using sensors in one third of the nodes 85

Figure 4.20 Data assimilation results using sensors in one fifth of the nodes 85

Figure 4.21 RMSE comparison using different number of sensors 86

Figure 4.22 Comparison of standard resampling with resampling using sensor data particles with 200 particles 88

Figure 4.23 Comparison of standard resampling with resampling using sensor data with 100 particles 89

Figure 5.1 (a) Smart Office with sensors (b)Heat map based on sensor count in a week. 90

Figure 5.2 Activity-informed DDDS framework..... 92

Figure 5.3 A simple example of a smart office environment consisting of a conference room and cafeteria connected by a hallway. 103

Figure 5.4 (a) Office layout with waypoint graph (b) A shopping mall layout with agents..... 95

Figure 5.5 Behavior pattern states transition probability for HMM 100

Figure 5.6 a) Sensor frequency data (b) Comparing the real and predicted behavior (c) Normalized probability for the behavior pattern in real time 100

Figure 5.7 (a) HMM for each location (b) A single path through an n-state HMM..... 103

Figure 5.8 (a) CHMM for 3 locations (b) Detail CHMM for location a having 3 states..... 105

1. INTRODUCTION

1.1 Overview

This dissertation work aims in creating an efficient real-time building occupancy simulation. A real-time building occupancy simulation consists of two parts: the simulation model for the building occupancy and, the framework for real-time estimation of occupancy. A building occupancy simulation models the dynamic spatial-temporal behavior and activities of occupants in buildings. Studying and knowing the occupancy dynamics of a building is useful for various applications, including more effective evacuation for buildings that have a large number of occupants, conserving energy based on occupancy presence, designing and monitoring smart environments, and real-time crowd management in places such as airports and train stations. Various simulation models have been developed to study the dynamics of building occupancy. These models study occupancy patterns, occupancy behavior and their interactions with each other as well as with the environment. Meanwhile, advances in sensor technology allow more and more buildings to be equipped with sensors, which can provide real-time information about the environment. Assimilating these sensor data into a simulation model would allow better state estimation of building occupancy and lead to more accurate simulation results. This is especially useful for supporting real-time decision-making related to building occupancy. Data assimilation is the process of integrating observation data from the real world into a simulation model to produce better estimates of system states. To enable data assimilation, both a simulation model that captures the state transition of the system and a data assimilation method that assimilates real-time data into the model are needed.

Various models have been used for building occupancy simulation, out of which agent-based and graph-based are two of the widely selected models ([1], [2], [3]). In agent-based

simulation, each agent represents an occupant, and the dynamic process of occupants is repeatedly simulated over time to generate the complex and intriguing emergent behavior [4]. A graph-based model for occupancy simulation uses the graph (node and edges) to represent the building structure and can model the occupancy dynamics using flow or queuing network [5]. The agent-based simulation has the advantage of being able to represent each occupant's behavior and decision making in detail. However, it has a high computation cost for applications with a large number of occupants. On the other hand, a graph-based model assumes occupants as a homogeneous mass and models their flow across the graph structure. It can model a large number of occupants in an efficient manner, but cannot model heterogeneity and individual behavior of the occupants. Realizing these limitations with the existing models, in this dissertation, we develop a graph-based agent-oriented model that combines the property of both graph based and agent based models to obtain their key advantages. In the model, the graph-based feature means occupants' movement is modeled as a flow from one node to another through a graph representing the building environment, and the agent oriented feature refers to the model's capacity to consider agents as individual entities with unique features and decision making capabilities. The model has the advantage that it can capture the heterogeneity of individual occupants, while in the meantime can simulate thousands of occupants in an efficient manner. This advantage is critical for supporting the data assimilation for buildings with a large number of people like a game stadium, shopping malls, terminals.

As the second part of the dissertation, we present a data assimilation framework which estimates the system state from observation data to achieve more accurate simulation. Data assimilation is an analysis technique that combines the observations of the actual system with the model to produce an estimate of system states. Typically, the state of the real system is

unobservable and thus, the simulation often starts from a state that is different from that of the real system leading to inaccurate results. Thus, there is a prominent need to estimate the current state of the real system dynamically. The sensor data provides an observation of the system, and data assimilation assimilates these observations to infer the current system state. Data assimilation has been widely used since long ago in areas of geosciences, mainly weather forecasting, and hydrology, and recently in other applications like forest-fire, smart environment, and traffic simulation. For the system, which is stationary, linear and Gaussian, various conventional state inference algorithms such as Kalman filter and its variants exists. However, in the graph based agent oriented model, the model is specified by behaviors and lacks analytic structures like in equation based numerical models. This makes it difficult to apply conventional estimation techniques and thus requires nonlinear, non-Gaussian, multimodal estimation techniques. In our work, we select Sequential Monte Carlo methods which are non-parametric filters for data assimilation of building occupancy simulation.

Sequential Monte Carlo (SMC) methods, also known as Particle Filters are a set of sample-based methods that recursively estimate the state of a dynamic system from observation data using Bayesian inference and stochastic sampling techniques [6]. SMC methods represent the probability density function as a set of samples each of which is known as a particle with an associated weight. Different methods like perfect sampling, sequential importance sampling, and resampling, acceptance-rejection sampling is used to generate samples for the particles. Particle filters have an advantage of being able to represent arbitrary probability densities without requiring all information about the structure of the system model, making it an effective method for supporting dynamic simulation with sophisticated simulation models. At the same time, particle filters are iterative methods that can recursively adjust their estimations of system states when new

observation data becomes available. This feature best suits the system where new sensor data arrive sequentially and the simulation system is continuously updated.

In this work, we performed some other works related to real-time building occupancy simulation and presented the findings as well. We developed an agent based model which we use to simulate a smart office environment [7]. The agent models the basic properties of an occupant: movement to destination and collision avoidance behavior. We use a way point graph structure for the environment which is a preplanned route calculation method to generate a path for agents. We simulate a basic smart office scenarios and collect data for some period. We then use Hidden Markov Models to use the historical data and predict the office behavior pattern in real-time. We also present a case study of data collected from real test bed consisting of binary sensors in an office layout. Since the graph based agent oriented model consists of both agent model and graph model, it allows a convenient conversion of agent based component to graph based and vice versa. Using this property, we have created a hybrid model which allows occupants to model in either agent based or graph based on any node section.

1.2 Background

Building occupancy simulation finds its application in a variety of fields leading to increasing emphasis on developing methods to build occupancy dynamics model. A model of occupancy dynamics in a building gives information about the building with its occupants and can predict the occupancy dynamics as a function of time [8]. Building occupancy simulation allows us to analyze the behavior of occupants in various applications as occupant movement simulation, evacuation simulation, or stadium evacuation simulation. Several models are used for building occupancy simulation, out of which agent-based and graph-based are two of the widely selected models. In agent-based simulation, each agent represents an occupant, and the dynamic process of

occupants is repeatedly simulated over time to generate the complex and intriguing emergent behavior. A graph-based model for occupancy estimation uses the graph (node and edges) to represent the building structure and can model the occupancy dynamics using flow or queuing network. The agent based model works at the lower level with a focus on agent's properties and interaction, whereas graph based works at a higher level with a focus on the spatial cognition and architecture.

An occupancy simulation model might require some initial conditions like building structure, total initial occupants, their properties, and historical behavior. The model needs to estimate the total occupants, their location and behavior for a realistic simulation of occupancy evolution over time. A building occupancy simulation models the behavior of the occupants in the environment so the model might depend on the type of the building. For example, for an office building, the occupants will normally come to the office, attend meetings, go to the cafeteria, and leave office whereas, in a school, the students will go to their classes, have breaks, and go home. Modeling these kinds of behaviors helps understand their behavior and is useful for applications like conserving energy and designing smart buildings. In another application, evacuation simulation is used to model the occupant's behavior in case of emergencies (fire, explosion, toxic gas threat). Stadium evacuation simulation represents evacuation for a larger mass of people and is critical since it concerns the safety of thousands of lives. The evacuation process is dependent not only on the building structure (passage width, exit placements, obstacles) but also on the crowd's behavior (speed, exit information, time to react). As such, it becomes important to have a model, which can simulate the process correctly and analyze the performance of evacuations in a specific building for various types of occupants.

Selecting an appropriate simulation model depends on the requirements of the analysis study and a single model may not be sufficient for all since each model is developed to represent a certain case problem. In occupancy modeling, depending on crowd size, their environment, and their motive, their dynamics might be different. As such an appropriate model is required, which can be used to analyze occupancy dynamics of a crowd of small to large size under different conditions. The existing agent-based models represent each occupant as a rational entity with own property and decision making capacity, and model the behavior emerging from their interaction. However, for an agent-based system, as the number of occupants increase, their interaction and complexity increase the computation cost, so at some point the simulation becomes infeasible. On the other hand, the existing graph-based models assume the occupants as a homogeneous mass and model their flow across the graph structure. The graph-based model can accommodate the increase in occupants, but since it does not consider the heterogeneity of the agent, the model may not represent the correct occupancy dynamics.

To overcome these limitations, we propose a graph-based agent-oriented simulation model, that combines the property of both graph-based and agent-based models to obtain both scalability and heterogeneity. We apply the model in occupancy modeling to demonstrate the advantage of our model for obtaining scalability while maintaining the individual agent property. In our model, graph-based represents the building structure in nodes/vertexes and represents the occupant's movement as flow from one node to another through the vertex. The model considers the lower entities as agents and provides them with individual features and decision making capabilities. To control the computational cost as the number of agents increase, we model only the required behavior of the agents which mainly impact the simulation. In our example, we show that the model efficiently simulates the behaviors of occupants with different characteristics in high-

density buildings under various circumstances and gives occupancy information such as a total number of occupants, their location, the rate of flow through an egress, occupants' speed and their behavior. Our research also focuses on using the real-time sensor data from the buildings for dynamic data driven assimilation. Most of the building simulation models are offline, as such they cannot synchronize with the real system. Nowadays sensors are becoming cheaper and viable to install in more and more buildings which make it possible to obtain real-time occupancy dynamics of the building. However, the sensors are not able to cover all the areas spatially and temporally and the sensor data are prone to noise error. To overcome these limitations, we developed a data assimilation framework using our new model and assimilated the sensor data for real-time occupancy dynamics estimation.

1.3 Problem Statement

Building occupancy estimation research is a popular area of interest, and a lot of models have been built to simulate the occupancy in various kinds of buildings. A lot of agent based and graph based models exist which simulate the normal occupancy, evacuation, and various other occupancy behaviors. Agent based model are efficient to model few number of occupants and graph based on a large number of people. As the number of agents increases in agent based model, computational cost increases and it might become difficult to simulate after a few hundred/ thousand number of agents. In graph based model the emergent behavior due to the interaction between agents cannot be simulated due to which the model might not appropriately model the occupant's behaviors. A building environment such as an office, school, labs do not have a large number of people in there, and so an agent based model is sufficient to model the occupancy dynamics there. However, an environment like shopping malls, concert areas, game stadium, train subways, airport terminal have many people. Having more number of people means that there will

be a high casualty in case of events like fire, earthquake, other threatening events. Thus, there is a significant need for these kinds of the environment to be simulated with accuracy for use in applications like building construction, evacuation planning, controlling crowd, etc. Graph based models although are computationally less expensive, not accurate enough to simulate such environment as they cannot appropriately model the lower level interactions of the agents. As such, the simulation may not properly represent the actual behaviors occurring in the building. Realizing these limitations of the existing models, we created a model which combines the property of both graph based and agent based models to obtain their key advantage: graph based scalability and agent based heterogeneity. In our model, graph based model represents the building structure in nodes and vertexes and represents the occupant's movement as flow from one node to another through the vertex. The model considers the lower entities as agents and provides them with individual features and decision making capabilities. To control the computational cost as the number of agents increase, we model only the required behavior of the agents which effects in the simulation. Our model can efficiently simulate building occupancy consisting of agents in some few hundreds to few thousands.

In the current state of building occupancy simulation, it is not enough only to model only the occupancy dynamics, but it is a significant requirement to provide a real-time estimation of the occupancy. Real-time occupancy simulation can provide dynamic estimates of the occupancy which can be used by rescuers to during emergency events to search, rescue and egress management. It allows monitoring of energy resource per real demand, track occupants and facilitate according to their needs in real-time. Various kinds of sensors (video, sound, IR, pressure) can be installed in different places of the building which can give estimates of the occupancy. However, the data from the sensors are not reliable and cannot be used for estimation

directly. The sensor data are full of noise, incomplete and incomplete so a framework is required which can utilize an appropriate algorithm to utilize sensor data for real-time building occupancy simulation. We develop a data assimilation framework using Sequential Monte Carlo methods which use our graph based agent oriented model to combine the sensor data. The framework efficiently assimilates the sensor data and estimates the occupancy in real time thus providing a real-time building occupancy simulation model.

1.4 Agent based model

An agent-based model (ABM) is a model for simulating the actions and interactions of single or multiple agents to analyze their emergent and individual behavior in the overall system. A single agent is defined as a discrete entity with its goals and behaviors with a capability to adapt and modify its actions depending on the environment. Agent behaviors are defined as a simple set of rules based on which agents perform their action, interact with other agents and the environment. These interactions over the course of time and space give rise to various system behaviors, patterns, and structures which give a better understanding of a rather complex system. An agent based model consists of agents, their relationships, and methods of interactions, and the environment. In an agent based model, an agent must be defined as an autonomous entity. They should be able to make their own decisions depending on the situations. Agents may be heterogeneous, with different property and goals, but the relationships and interactions between the various agents must be defined clearly. The environment is the space in which agents behave and interact to evolve the system over time. As such, agent based model can be expressed in a definition as “a computational method that enables a researcher to create, analyze and experiment with models composed of agents that interact with an environment” [9]. ABMs finds its application in a broad range of areas and disciplines like biology, business, technology, economics and social sciences and others.

ABMs is widely used for modeling occupants and their behavior in a building structure. An agent in the model can represent a real-world occupant entity with appropriate property and behavior which makes the model reasonably accurate to represent occupant movement dynamics in a given environment. An agent can be modeled to behave like a person by assigning certain speed, size, goal and decision making property. Heterogeneous agents can be created as different agents with different properties representing a mixed population of various gender, size, age, speed, motive, interactions, and other properties. These agents interact in each environment; in this case in a certain building like an office or school; they move around with various speed, go to a destination, follow the rules like avoidance, grouping, herding, interact with each other and the environment like evacuation, fleeing, etc. These kinds of interactions form the overall dynamics of the occupancy modeling. Emergent behavior is generated from the aggregated individual behavior which represents the behavior of the whole system. From this, we can study the nature of the overall system as well as study the resulting impact on the individual agent.

ABMS are suitable to model occupancy dynamics since we can represent occupants easily as agents with human like behaviors and decision making capabilities. The building where the occupants exist is set as the environment and the interaction between agents and environment can be defined as a set of rules. The agents perform the behavior of the real occupants under situations that the system is meant to execute. For example, simulation can be performed where occupants are given destination as goals. Now the agents will move towards their destination following some rules which govern their direction and movement. They might interact with each other and exert behaviors like cohesion or avoidance. The environment might be a building structure with various rooms as their locations and destinations. When a large number of agents move to a common destination emergent behaviors like crowd formation might be observed. But at the same time, we

can observe the behavior of an individual agent, how they interact with other agents and the environment during crowd formation. The advantage of using ABMs for occupancy dynamics is that each agent can efficiently represent a real-world occupant and the impact on the individual due to various situations can be studied using resultant behavior of the agent representing the individual. As such, a lot of occupancy models use ABMs to represent their system however, when the number of agents increase, the system becomes incredibly complex and consume high resource. Thus, it becomes difficult to model the system efficiently with large agents and complex behaviors. Also because of their high degree of complexity, ABMs are unsuitable for real-time simulation.

1.5 Graph based model

Graph based models (GBM) are used for representing structure information and provide a higher-level view of the model compared with agent based models. Graph based model consists of vertex/node and edge/links representing the structure and some equation to model the system flow. In occupancy modeling, the environment structure where the occupants perform is represented using vertex-edge; the vertices of the graph represent the zones of the structure, and the edges represent the connectivity between the vertices. The vertices may have properties like size, capacity, type and the edge connecting two or more vertices may have properties like length, width, capacity, type (corridor, stairs). Occupancy is represented by the number of occupants in each zone, and their dynamics are modeled as occupancy flows between the zones governed by flow equations or queue modeling. Various equations have been developed which efficiently model the occupancy dynamics in various situations [1]. The GBMs are highly faster than the ABMS and consume less resource. So, in situations consisting of a large number of agents when it becomes expensive and slow to model, GBMs can model such situation efficiently. For a general

system, the computational complexity of simulation grows proportionally with the number of agents, their properties, and the structure of the environment. In case of simulations like a football stadium or airport terminals, where there are thousands of occupants, it is very expensive to model the system using ABM since there will be thousands of agents to represent the occupants. But the GBMs can model thousands of occupants in a large environment with ease.

The GBMs do not represent the individual occupants using the agents, so it is much faster than the ABMs. Since the concept of individual agent is not implemented, the emergent behavior between the interactions between individual agents cannot be observed in GBMs. Because of this missing feature, the GBMs might not be able to accurately represent the evolution of the system based on the individual agent behaviors in the environment. Also, it is not possible to observe the resulting impact on individual occupant during various scenarios of the system execution. But fortunately for occupancy modeling, when there are a large number of occupants the overall dynamics of the system is not affected by the behavior or interaction of a single individual occupant. The overall movement pattern is more like the flow of particles over the environment and is observed at a higher level rather than lower agent level. For example, when many people move towards the same direction, if the passage is narrow, a crowd immediately forms. Whatever be the individual occupant speed, it does not matter when a congestion is created, everyone will slow down and move with constant speed. As such we can ignore the lower level interactions of the occupants and yet get an accurate model of the system.

1.6 Sensors

Buildings today are equipped with different kinds of sensors which can be used to obtain various information like the number of people, CO2 count, presence detection, the direction of the crowd, etc. Sensors like video cameras, sound detectors, passive infrared motion detectors, access-

control devices, weight and pressure measurements and many others are available easily and in cheap cost today. Sensors are present in electronic devices like cell phones, tablets, computers and people also carry active RFID tags, NFC tags, GPS which can easily locate and track various activities. However, the data collected from the sensor cannot be directly used as the output of the estimation as they are full of noise and are incomplete. They have large uncertainty and cannot be used as a direct measurement of the environment [10]. Taking an example of a simple infrared (IR) sensor which detects when a person moves through their radius, it might false detect an animal, or even a random noise. Also, it might give multiple counts for a single person or vice versa. Sometimes, the sensor might fail to trigger and not detect at all thus giving false count. This kind of detection errors is considered as noise in the data. Since sensors record data in certain time intervals, during the time interval, information is not present. To place sensors such that they cover all the area is not possible due to cost and other accessibility issues like confidentiality, privacy. As such sensor data are incomplete regarding time and space.

In this research, we do not focus on correcting the sensor data. Instead, we assimilate the available sensor data in our model to predict the current state of the system. We assume that the building area we model has sensors installed in some areas only. Currently, sensors have the probability of detecting occupant counts by an average of up to 80% [11]. In our work, we assume that the sensors are placed such that they can predict the number of people in a room with 80% accuracy. We utilize the sensor data to correctly estimate the system state like real-time occupancy count in each area. We develop a non-linear stochastic state-space model for occupancy dynamics and use Sequential Monte Carlo methods (particle filter) to assimilate the sensor data. The sensor data are incomplete: they do not cover all the areas and time. However, the output of the model

will estimate the occupancy count over all the areas through each time step. As such, the model solves the limitations of the sensor data and provides a real-time building occupancy estimation.

1.7 Data assimilation

Data assimilation is the process of incorporating the observations of the real physical system into the simulation model of that system. Data assimilation assimilates data from the environment to improve state estimation of the system under study [6]. A smart environment is integrated with various sensors that can provide real-time information about the inhabitants present in it. This integration makes it possible for the simulation model to dynamically obtain the real-time sensor data and serve as an online tool to support real-time decision-making by incorporating the data. The online tool utilizes both sensor data information and the simulation model to "predict" the dynamics of the system in real time and thus has a different purpose from traditional offline simulation-based studies (e.g., using agent-based simulation to carry out what-if analysis).

In the real world, the system's states, which change over time, cannot be directly observed and is unknown to the simulation model. Thus the simulation may start from a state different from the state of the real system, leading to inaccurate simulation results. Hence, there is a need to dynamically estimate the "current" state of the real system and then feed the estimated states to the simulation model. This is achieved through data assimilation that utilizes real-time sensor data for inference of the "current" system state. In [12] we presented the use of data assimilation in a smart office environment to inference peoples' occupancy information from sensor data. We used Particle Filters (PF) as the data assimilation algorithm to assimilate real-time sensor data into a simulation model and achieved improved results for simulating movements of single and multiple agents in a smart office. Data assimilation combines the observations (i.e., sensor data) of the current state of a system with the results from a prediction model (i.e., the simulation model) to

produce an analysis. The results of data assimilation thus depend not only on the observation data, but also on the simulation model that "predicts" the evolution of system state. In this work, we present a framework which uses our simulation model and helps in real time analysis of occupancy.

1.8 Sequential Monte Carlo methods

We use Sequential Monte Carlo (SMC) methods also known as particle filters (PF) for data assimilation. It is a set of sample-based methods that recursively estimate the state of the dynamic system from observation data using Bayesian inference and stochastic sampling techniques [13]. A key advantage of particle filters is their ability to represent arbitrary probability densities and with little or no assumption about the structure of the system model. This makes it an effective method for supporting dynamic simulation with sophisticated simulation models. Meanwhile, PF is recursive methods that can recursively adjust their estimations of system states when new observation data becomes available. This feature is suited where new sensor data arrives sequentially, and the simulation system needs to be continuously updated.

To carry out the data assimilation based on PF, we need to formulate the problem using a non-linear state-space model as shown by the system transition function in (1.1).

$$\begin{aligned} S_{t+1} &= SM(S_t, t) + v_t \ , \\ Z_t &= OM(S_t, t) + w_t \end{aligned} \qquad (1.1)$$

In the equation, S_t and S_{t+1} are the system state variables at time step t and $t+1$ respectively. Z_t is the observation variable representing the observations or measurements (sensor data). *SM* is the system transition model and defines the evolution of the system state. In our work, this system transition model is the simulation model. *OM* is the observation model and defines the computation of observation variable from the current system state. The v_t and w_t are random variables which refer to the noises of the system state and the observation data respectively. Based on the non-

linear state-space model of (1.1), PF can be applied to estimate the new states at each time step by assimilating real-time sensor data. In the implementation of PF, the system state is represented by particles, where each particle is a state candidate.

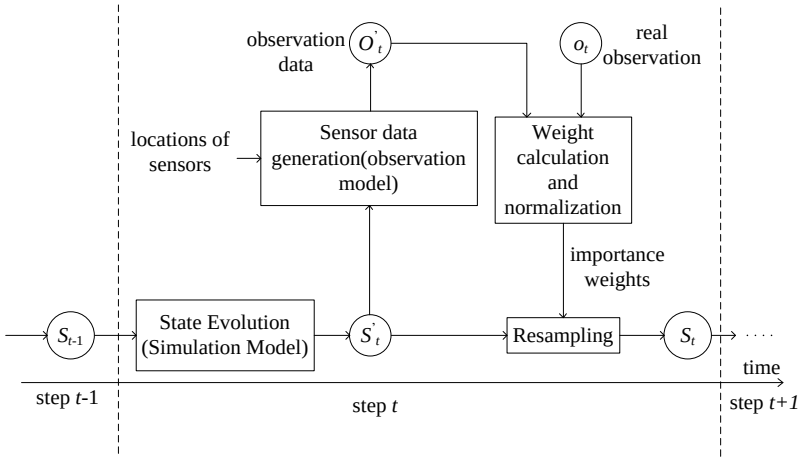


Figure 1.1 Data assimilation based on PF methods

Figure 1.1 shows the structure of PF and the procedure for data assimilation. In the figure, at time step *t*, all particles' states S_{t-1} from time step *t*-1 are input into the simulation model and evolve to a new set of state S'_t. Based on the new states, the observation model generates the new observations O'_t. These observation data are compared with the real observation data O_t and the importance weight of each particle is computed. The importance weights of all the particles are then normalized, and the resampling algorithm draws a set of offspring samples S_t from S'_t which has a probability proportional to the importance weights. These set of resampled states will be the input for the next time step *t*+1.

1.9 Organization of the Dissertation

This dissertation is organized as follows: Section 2 summarizes the related work in this research area; Section 3 presents the new graph based agent oriented model for occupancy simulation. We present the components of the framework and describe how they function together.

We also present an application of the model in simulating occupancy in a tunnel. A hybrid model with nodes having pure agents or graph based agents is also presented. We describe the data assimilation framework using Sequential Monte Carlo method in Section 4. Here we also present a new framework which utilizes the direct sensor data to create new improved particles and present the various experiments performed to validate the graph based agent oriented model and the data assimilation framework. We summarize the results for various scenarios of occupancy in different building sizes. Section 5 describes a framework for improving dynamic data driven simulation using behavior pattern detection. We create an agent based model for the smart environment and perform behavior pattern detection using the Hidden Markov Model. Conclusion provides an overview of the dissertation work along with the possible future extensions.

2. RELATED WORK

Building occupancy simulation is concerned with modeling and analysis of occupant property and behaviors. In [14], the author defines occupancy at four levels varying with time: first the number of occupants in buildings, second the occupancy status of space, third the number of occupants in space and fourth the space location of occupants. The author classifies and reviews the occupancy models and provides new methods to integrate the models into a tool that can be used in different ways for different applications. Building occupancy finds its wide use in conserving resources. In [15], the author has developed a framework known as sensor utility network which uses information from various sensors to dynamically estimate the occupancy and conserver resources. Building occupancy provides a basis for a smart environment where we can simulate our test beds with various sensor and actuators and occupancy. [16] discusses the latest research in smart environment, philosophical and computational architecture considerations, network protocols, intelligent sensor networks and powerline control of devices, and action prediction and identification for the smart environment.

Agent and graph based models have been used extensively for various modeling applications. Agents can be used to represent individuals and assign them appropriate behaviors like movement, vision, collision, congestion avoidance, and so on. In [17], agent based model is used for simulating general pedestrians in groups and evaluate in various scenarios. In [18], the authors present HuNAC (Human's Nature of Autonomous Crowds) model which replicate pedestrian psychological factors to simulate behaviors in crowds. In [19], the authors propose a layered approach to model the dynamics of the pedestrian crowd. The surface of the 2D environment is divided into different layers to indicate the occupancy, the position of static obstacles and possibly the dynamic obstacles situated in the environment. The agent utilizes the

layered environment and uses Markov decision process and semi-Markov decision process approach to finding the correct move given its occupying cell. In [20], the author proposes a model consisting of an environment model and an agent-based model. The environment model consists of a route map, navigation map and the information of each object. To decide a path of an agent, the agent's destination and position are added to the topological graph and the shortest path algorithm is utilized over the graph to find a suitable path for the agent.

Graph based models are used for representing structure information and provide a higher-level view of the model in contrast to agent-based, which focus on the lower level complexity of the agent interactions. In [21], a grid graph-based model known as ESM (Evacuation Simulation Model) is proposed which considers the structural and spatial properties of the indoor space and shows advantages for indoor route analysis in evacuation simulation. The work builds a graph with each vertex representing a room, a segment of corridor or hallways and each edge representing a pipeline that occupant can transport on. The transportation of occupants on the edge of the graph is determined by factors such as social affiliation, access visibility, tenable time, speed, flow rate and the distance between rooms. In [22], the author's approach is based on three different models: an agent-based model, which includes the detailed description of each individual occupant's velocity, behavior and trajectory; a graph model, which represents the building structure and traffic dynamics using a graph, and a kinetic model, which models the congested areas of the building as queues to simulate the situation of congestion in the building. Although graph-based models seem to accommodate the increase in agents, it suffers from some serious limitations. First, the scenario of congestion cannot be easily modeled because the occupants in graph model are treated as homogeneous and so the occupant's individual properties are not considered. In an environment where occupants have different body size, speed, and motives the resulting behavior might be

different depending on the environment. In [23], the authors present a quantitative evacuation model for analyzing evacuation results of a crowd in a stadium.

With the abundance of various sensors which can collect data about the occupants of a building, there has been a recent trend of using data to obtain information about the environment and the occupants in it. In [24] the authors propose to improve the functionality of the sensors to reduce the energy consumption in buildings. The sensors are designed to learn from the activity of occupants, and it detects and adjusts how long the light will be turned off after the detection of occupants' motion. In other works, [11] provides a probabilistic model for inferring occupancy count from the sensor data and [25] utilizes sensor measurements along with historical data regarding building utilization to produce occupancy estimates through the solution of a receding horizon convex optimization problem. The authors in [26] have designed a real-time user tracking system for smart environments from non-invasive binary motion sensor data. They have used Hidden Markov Model with Viterbi decoding for determining occupant path from collected data.

Using agent-based simulations for dynamic estimation of occupancy using sensor data is computationally expensive that it becomes unrealizable as the number of occupants increases highly. Thus, the current works using agent based models can model for occupants in real time in the number of 100s only. Building models at a higher level which are appropriate for real-time estimation regarding computation cost are difficult because of the high uncertainty of occupancy dynamics. For modeling many occupants, graphical models are much suitable. Authors in [22] have presented an agent-based model to simulate the behavior of occupants in buildings but have extracted a reduced model for real-time estimation. Still, the current works are insufficient regarding predicting real-time occupancy in all areas and for a high number of people.

Various methods can be used to utilize the sensor data for improving occupancy estimation. In our work, we use data assimilation which is a process of combining the observations of the current state of a system with the results from a simulation model to produce a new estimation of the state. It combines the information of the current state of a system with the results from a prediction model to produce an analysis. The method of data assimilation for incorporating observations finds its use in various fields of geosciences, weather forecasting, hydrology, and other environmental systems. The analysis techniques used for data assimilation include methods like three-dimensional variational analysis (3D-VAR), four-dimensional variational assimilation (4D-VAR), Particle Filters, Kalman Filters and others. A data assimilation system using 3D-VAR to improve ozone simulations in Mexico City basin is presented in [27] which generates the optimal estimate of the true atmospheric state during the analysis time. In another work, 4D-VAR is used to a regional ocean modeling system to produce an optimal estimation of the real ocean state using satellite remotely sensed observations [28]. Kalman filter [29] can be used to estimate the state of a dynamic system with observations represented by a linear state space model. In [30] three extensions of the Kalman filter; extended Kalman filter, limited extended Kalman filter and unscented Kalman filter has been presented to find the solutions to nonlinear discrete-time state-space.

The data assimilation algorithm used in this research is Particle Filters (PF). Particle Filters can be applied to dynamic systems with non-linear behaviors, by approximating the state of dynamic systems using particles and associated weights. A framework of dynamic data driven simulation based on PF is presented in [31] for the forest fire spread simulation. It presents a data assimilation framework based on PF to improve wildfire spread simulations using DEVS-FIRE, in which real-time data are fed into the DEVS-FIRE simulation to improve the accuracy of wildfire

simulation. In another research [32], PF is used to develop a framework and algorithms to solve the problems of positioning, navigation, and tracking. The author presents a general algorithm based on marginalization enabling a Kalman filter to estimate all position derivatives. The work describes the applications in car positioning, aircraft positioning, target tracking, combined navigation and tracking and car collision avoidance. PF find use in other fields like biology and chemistry as well. In [33], PF is used to create populations of compact long chain polymers and the relationships between packing density and chain length. PF are used in [34] to set up a probabilistic framework providing a basic for process fault diagnosis for the dynamic data rectification.

In literature, we can find a lot of work related to building occupancy simulation. However less research exists for real-time building simulation. And among the existing work, most of them consider few occupants in a normal building. In [22], the authors have used extended Kalman filter to assimilate sensor data with agent based model with occupancy of about 100 people only. [35] presents a data assimilation framework for estimating movement patterns of about six occupants in an office environment. In our previous work [12], we developed a framework for using data assimilation for binary sensor data to predict occupancy behaviors in smart environments. However, all these models are agent based and they are computationally expensive to use as the number of agents increase. With the increase in the number of agents, their state size becomes bigger thus increasing the cost of computation. In this paper, our proposed model utilizes the graph based agent-oriented model which treats the system as a reduced order model where the occupants are treated as individual agents operating with their intelligence. The model reduces the size of the state to be estimated. We then use data assimilation using particle filter to estimate the real occupancy using the available sensor data. The efficiency and low resource consumption of the

model makes the overall process quite efficient for real occupancy estimation of a large number of occupants.

3. GRAPH-BASED AGENT-ORIENTED MODEL

Since an agent-based model considers the detailed information for low-level abstraction, with the increase in agents, the computational cost becomes expensive. Instead, to simulate the scenario where a considerably large number of occupants are involved, a low-resolution model describing the occupancy dynamics at high abstraction level is appropriate. Our model aims to simulate occupancy dynamics for places with thousands of occupants by combining agent based model with graph-based. Thus, the model structure is graph based with agent-driven behaviors of residents. In the scenario where the existing agent-based models can represent the occupants in a range of few hundreds, and the graph based models lack the complex heterogeneity of the agents, our model combines agent-based and graph-based models to reduce the computational cost while maintaining the heterogeneity of the agents. Each occupant is treated as an individual agent with an own set of properties and decision-making capabilities. The graph model manages the structural information with links as queues to model the occupancy flow. In our model, we try to exclude the lower level property and interaction of the occupants to lower the computational cost, and at the same time, maintain the necessary agent-based integrity of the model by treating the occupants as individual agents operating with their intelligence. The agent integrity is maintained by preserving the essential features while removing the unimportant detailed properties. The building structure is represented as a graph, with all the rooms represented as nodes and their connecting doors or passages as edge links. The behavior of occupants in the building is represented using flows among the edge links of the graph which are treated as queues. Since movements of occupants in the network are more orderly organized, simulation of interactions among occupants is not necessarily needed. As such, detailed interaction of agents like collision avoidance, position tracking is not required. Instead of using a coordinate to indicate the occupant’s position, the model represents

only the location of the occupant as its current node position. Removing these detailed properties significantly reduces the computational complexity of the simulation. The framework of our model is represented in Figure 3.1.

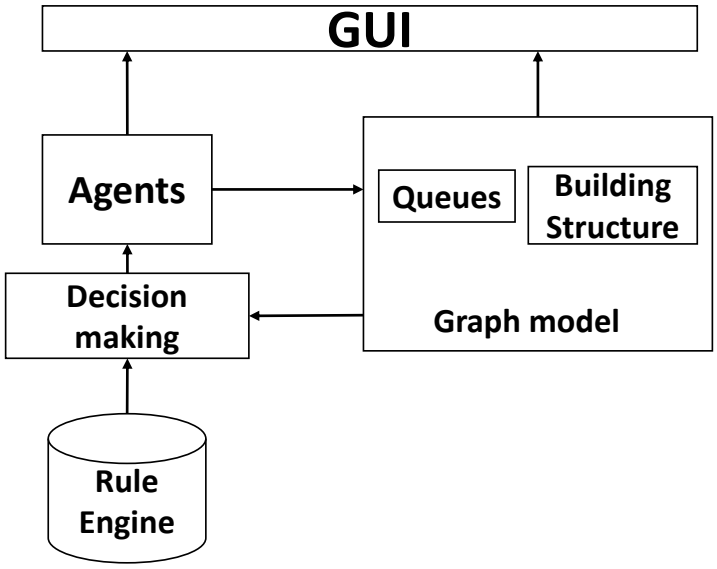


Figure 3.1. Framework for graph-based agent-oriented model

3.1 General structure of the graph model

The graph model consists of two submodels. One is the model of the building, and the other is the model of the occupant. The model of the building defines the graph structure of the building and how the occupants will move from a node of the building to another. The model of the occupant describes the properties and attributes of the occupants. In the model of the building, the building structure is represented using a graph. The vertex of the graph represents a segment of the building structure. Typically, the segment of the building is a section of a corridor or the zone of a room and the edges between the vertices represent the connectivity among the segments of the building. For instance, consider a floor plan of a building structure illustrated in Figure 3.2(a).

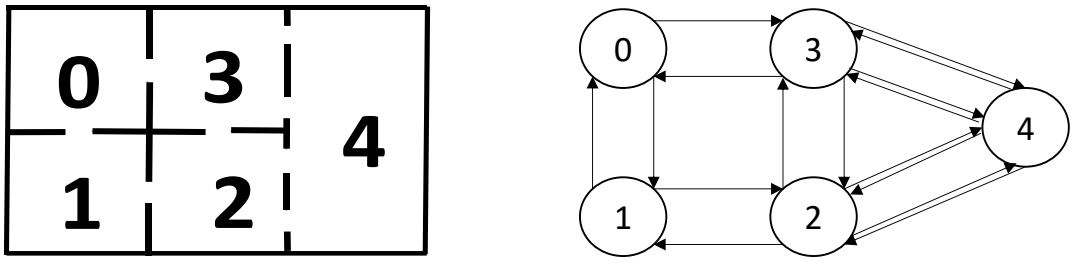


Figure 3.2. a) A building floor plan b) Graph representing the floor plan

Figure 3.2(a) presents a floor plan with five rooms. In this floor plan, there is a door between room 0 and room 3, room 0 and room 1, room 1 and room 2, room 2 and room 3, room 2 and room 4, and finally, room 3 and room 4. We consider that the rooms connected with doors are linked using an edge directed from one room to another. Taking room 0 and room 3 as an example, this means that, there is an edge directed from room 0 to room 3 and an edge directed from room 3 to room 0 because there is a door between room 0 and room 3. The graph representing this floor plan can then be constructed as shown in Figure 3.2(b).

In this graph, vertices are connected using directed links. The vertex has several properties: the id of the node indicates the identity of the vertex. The capacity of the node indicates the maximum number of occupants that can reside in the zone represented by this vertex. The number of occupants of the vertex represents how many occupants are currently staying in this vertex, and the neighbor is an array of vertices that is connected to this vertex with links. To guide the movement of occupants, we use transition tables in each vertex to indicate which direction the occupants staying in that vertex should move to. The transition table defines the probability that the occupants move from current vertex to its neighbor vertices. In the example shown in Figure 3.2(a), for instance, the occupants staying in vertex 2 will have 4 choices of movement, which are moving to vertex 1, 3, 4 and staying in vertex 2 respectively. The transition table defines the

probability the occupants in the vertex perform each of the movements accordingly. In this example, a transition table can be demonstrated in Table 3.1.

Table 3.1 Transition table for vertex 2 a) normal scenario b) evacuation to Node 4

Vertex	Probability
-1	0.4
1	0.2
3	0.2
4	0.2

Vertex	Probability
-1	0.0
1	0.1
3	0.1
4	0.8

In Table 3.1(a), -1 denotes occupants staying in the current node. The probability for the occupants to stay in vertex 2 (same vertex) is 0.4, the probability for the occupants to move to the vertex 1 is 0.2, the probability for the occupants to move to the vertex 3 is 0.2, and the probability for the occupants to move to the vertex 4 is 0.2. The transition table in each vertex can be considered as a profile of movement pattern. In different situations, the profile of movement pattern can be different. For example, consider vertex 4 in Figure 3.2(a) as an exit room. In a normal condition, the probability for occupants in vertex 2 to move to the vertex 4 can be given as 0.25. However, when there is an emergency in the building that requires sudden evacuation, the probability for occupants to move from vertex 2 to vertex 4 may raise to 0.8 because, under pressure, occupants are more likely to communally rush to the exit (Table 3.1(b)). Therefore, there are multiple transition tables representing different profiles of occupants' movement patterns in each vertex of the graph. Specifically, the vertex in the model can be represented using 6 elements variable *v*, where

$$v = < i, c, L', D, E, T > \qquad (3.1)$$

In this equation, *i* is the *id* of the vertex, *c* is the capacity of the room represented by the vertex, *L'* is the collection of links that connect with the vertex, *D* is the distances between the links in *L', E* is a collection of neighbor vertices of the vertex and *T* is a collection of transition tables of the vertex.

The link of the graph also consists of several properties. The "*from*" vertex demonstrates the original vertex of the link and the "*to*" vertex displays the vertex that the link points to. The *flow capacity* of a link defines the maximum number of occupants that can go through this link at each time step and depends on the link width and density at that time step. Flow capacity is expressed in term of movement speed, link width, and density using equation from [37] as

$$c_f = spd * d * w \qquad (3.2)$$

Here, *s* is the speed of movement, *d* is the density, and *w* is the width of the passage. Density is the number of persons in a unit area and indicates the degree of crowdedness in a passage or room. As the density increases, the speed of the movement is reduced as occupants find it difficult to move freely. If *s'* is the normal occupant speed, then new speed of movement due to density is computed using equations from [38] as follows

$$spd = \left(\frac{(s'(d - 0.25))}{0.87} \right) \qquad (3.3)$$

$$d = \sqrt{\frac{1}{D}}$$

Here *d* is the inter-person distance, D is the density, and if $d > 1.12$ then speed is assumed to be unimpeded. Two nodes may have more than one links distinguished by separate ids. To model the congestion in the building, for each link, we define a *move queue* to describe the queue formed by the occupants that is moving through this link. At each time step, the occupant decides

which room to move to and using which link. The occupant then spends a period of time to move to the link connecting the current vertex where the occupant resides to the destination vertex. After that, the occupant registers himself to the move queue of that link. Numbers of occupants that equal to the flow capacity of the link are removed from the move queue at each time step. After being removed from the queue, these occupants arrive at their destination and then again make the decision to either stay there or move to another place. Besides the property of the link itself, there is also a property *link distance* that represents the distance between two links. The link of the graph can be considered as the gate between rooms. The link distance is then the distance between the gates of the rooms. As the link distance is known, we can estimate the time required for the occupant to travel in the room to arrive its destination. Specifically, the link in the graph model can be represented using a five elements variable *l*, where

$$l =< i, v_{from}, v_{to}, c_f, q > \qquad (3.4)$$

In this equation, i represents the id of the link, v_{from} represent the *from vertex* of the link, v_{to} represent the *to the vertex* of the link, c_f represents the *flow capacity* of the link, and q is the *move queue* of the link.

3.2 Occupant Agents

An occupant is represented as an agent, which has basic properties and can make individual decisions. The agent will have node location, destination, speed, body size and their type (rescuer, occupant). The property to the agent can be added depending on the simulation requirement. In the current model, an agent can be an independent, part of a group or follow a herd. The group can be any given size and the agents of the group try to move together during the simulation. Also, the agents can be of two categories: first, regular occupants who knows the information about the exits

and second, new occupants who do not know the exit and follows the crowd during evacuation. During the simulation, the agents represent the emergent behavior with time due to crowd movement across the space.

The model of occupant describes properties of the occupants and how occupant moves from one vertex of the graph to another during simulation. The *state* of the occupant defines what profile of movement pattern will be used to determine the moving direction of the occupant. In other words, if the vertex of the graph contains multiple transition tables, the state of occupant determines which transition table will be used to guide the occupant's movement. The *location* of the occupant defines the *id* of the vertex that the occupant is currently staying in. The *id* of the occupant defines the identification of the occupant. The g_{id} represents the group id of the agent. While moving from one node to another, the occupant is registered to the queue of the link from where it is entering the node. The *isInQueue* property indicates whether the occupant has registered itself into a move queue of a link. The *destination* defines the vertex that the occupant decides to move to. The "*from*" link defines the link that the occupant went over to enter its current location. The "*to*" link defines the link that occupant will use to move to its destination. The *speed* defines the distance an occupant move along in a single time unit and is dependent on the density. The speed will be slow if the density is high since it will impede the free movement of the occupants. The *action* of the occupant defines whether the occupant is moving to other vertex or is staying in his current location. The *elapse time* is the time the occupant has spent in its current location. The *delay* of the occupant is a counter that counts how many time units the occupant will spend in his current location. This counter is reset each time an occupant enters a new vertex. The value assigned to the delay of the occupant is calculated based on the link distance between the "*from*" link and "*to*" link of the occupant and its speed. The counter reduces by 1 at each time step. When

it reaches 0, it means that the occupant arrives at the "*to*" link and is ready to move to its destination vertex. The occupant then registers itself to the move queue of the "*to*" link. The *action* property defines the action that the occupant is performing; if it is 0, it means the occupant stays in current vertex; if it is 1, it means the occupant is moving to another vertex. Specifically, the occupant in the model can be represented using variable o, where

$$o =< ph, v_{position}, id, g_{id}, l_{from}, s >$$
$$s = \begin{cases} staying < t_{elapse}, t_{stay\_delay} > \\ moving < t_{elapse}, t_{move\_delay}, l_{to}, spd > \\ inqueue < V_{destination} > \end{cases} \quad (3.5)$$

In this equation, *ph* represents the phase of the occupant to determine which transition table to be used to guide occupants movements, $v_{position}$ represents the current vertex the occupant reside in, *id* represents the identification of the occupant, l_{from} represents the link that the occupant come from, *s* represents the state of the occupants and has three different values, *staying*, *moving* and *inqueue*. Each of the value is associated with several properties. l_{to} represents the link that occupant is going to, $v_{destination}$ represents the destination of the occupant, *spd* represents the speed of the occupants, t_{elapse} time represents the time elapsed when the occupant stay in current room, t_{stay_delay} represents the time count that the occupant will stay in current vertex, t_{move_delay} represents the time count that the occupant will be moving in current vertex. The collection of *m* occupants in the model is represented by

$$O =< o_1, o_2, o_3, ..., o_m > \quad (3.6)$$

where, o_m is the number of occupant in each of *m* nodes.

3.3 Agent's behavior

The model of the building and the model of the occupant together make the general structure of our agent-directed graph model. At each time step, the graph model evolves per two sets of different rules. One set of rules describes the behavior of the occupants at each time step including how they decide the destination vertex and how they move to the destination. Similarly, the second set of rules reveals the behavior of occupant in the move queue of each link (defined in 4.5). The rules to decide the agent's destination and how they move to the destination depends on their property. For analyzing the behavior of agents, we have developed two types of property for agents. The first property represents the agents as a member of groups with variable group size. This property aims to analyze the behavior of occupants when they belong to a group (family, friends, and colleagues) and try to remain in the group during any situations. So, they will move to the same nodes, wait for the group at the gates before going in, wait for all the group members to reach the room before making any new decision. The second property divides the agents into two categories, one is the agent which knows the information about the building and another which does not. The first kind of agents may be regular occupants, rescuers, firefighters, or safety escorts and will be present in every node. The second kind of occupants are the ones who do not know where to go but will follow a crowd assuming that it is moving in the correct direction. They will follow according to the probability rule. Assume there are *n* occupants in a node and there are 3 exits with 30% exiting crowd in gate1, 50% exiting crowd in gate2 and 20% exiting crowd in gate3. Then from *n* occupants, 30% will choose gate1, 50% will choose gate2 and rest will choose gate3 as an exit. We assume that the probability is based on vision sense of the occupants. In the case of emergency evacuation, we assume that one of the nodes will be the safety node and agents reaching there will remain there till the situation improves.

3.4 Agent movement model

For the model of occupant movement behavior, at each time step, if the delay of the occupant is larger than 0, then the delay of the occupant decreases by 1 and elapse time of the occupant increases by 1. The delay of the occupant serves as a counter to count how many time step the occupant will stay in its current location and the elapsed time of the occupant represents the number of time steps the agent has been staying in the current location. If the occupant's delay is set to -1, this means this occupant has just entered its current location and needs to determine its destination and delay. To determine the destination of the occupant, occupant looks up the transition table (Table 4.1) of the current vertex it resides in and selects movement behavior profile according to its current state. After selecting a destination vertex, the link connecting the current vertex with the destination is selected, and in the case of more than one links, the agent will select the link with fewer occupants in line. After the determination of the destination, the next step is to determine how long the occupant will stay in the current vertex. The occupant stays in current vertex for two reasons: if the occupant's destination is current vertex, it means it chooses to stay in this vertex for some time. If the occupant's destination is one of the neighbor vertices of current vertex, the occupant will spend some time to travel to that neighboring vertex. The duration an occupant takes to stay in current vertex is a random value between given *minStayTime* and *maxStayTime*. The duration an occupant will take to travel to the neighbor vertex is determined by the speed of the occupant and the link distance between the link that the occupant used to enter its current vertex and the link connecting its current vertex to the neighbor vertex it will travel. Specifically, it is a random value between the *maxDelay* and *minDelay*. The *maxDelay* is calculated by the link distance divided by the speed of the occupant, and the *minDelay* is either 0 (in the case *maxDelay* is smaller than *elapsetime*) or *maxDelay* minus *elapsetime* (in the case

maxDelay is larger or equal to the *elapsetime*). After the delay of the occupant is set, it reduces by one at each time step. When it reaches 0, it means the staying time for the occupant has expired, or the occupant has finished traveling and is ready to enter the destination vertex. Then the occupant is queued into the moving queue of the link that connects the occupant's current vertex to its destination vertex.

3.5 Queue Processing

Each link is a queue where occupants are added to move to next node. The rules to decide the behavior of occupant in the move queue of each link is explained as follows. At each time step, the order of the links is randomized before processing them. For each link of the graph, the flow rate is computed depending on the density of the link, then the number of occupants equals to the flow capacity of that link are moved from the moving queue to the corresponding destination. The occupants will be processed on a first-come-first-serve basis, assuming that the first arrived agents will be first in a line of the crowd. The occupants that are removed from the move queue enter their destination vertices where the destinations of the occupants become the locations of the occupants. Meanwhile, the delays, *elapsetime*, *isInQueue* of the occupants are reset, and the "*from*" link properties of the occupants are set to the link that the occupants are removed from. The occupants make their new decision depending on the situation (normal, emergency, conference) of the system.

3.6 Discrete time-based simulation algorithm

Here we define the algorithm for basic discrete time-based simulation of the model. At each time step, the graph model evolves following two sets of different rules for agent movement and queue processing. The first set of rules defines the behavior of the occupants at each time step including how they decide the new destination and how they move towards the destination, second

set of rules defines the behavior of occupant in the queue of each link and how each occupant is transferred to the new node.

Discrete time-based simulation

for time $t = 0$ to $t = simulationtime$

// dynamics of agent

for each agent o

$t_{elapse} = t_{elapse} + \Delta t$

if ($o.s = staying$)

$t_{stay\_delay} = t_{stay\_delay} - \Delta t$

if ($o.t_{stay\_delay} \leq 0$)

make decision to stay or move

if ($o.s = moving$)

$v_{destination}$ = new node

calculate the new t_{move_delay}

else if ($o.s = staying$)

calculate the new t_{stay_delay}

if ($o.s = moving$)

$t_{move\_delay} = t_{move\_delay} - \Delta t$

if ($t_{move\_delay} \leq 0$)

$o.s = inqueue$

if (o.s =inqueue)

//do nothing

end for

//dynamics of queue

for each link l

for $flowrate$ 1 to $l.c_f$

if ((l.$n_{to.c}$ > l.$n_{to.N}$) && ($l.q$!= $null$))

o = remove one agent from $l.q$

$o.n_{position} = l.n_{to}$

o. $t_{elapse} = 0$

$l.n_{from.N} = l.n_{from.N} - 1$

$l.n_{to.N} = l.n_{to.N} + 1$

make decision to stay or move for agent o

if (*o.s = moving*)

$v_{destination}$ = new node

calculate the new t_{move_delay}

else if (*o.s= staying*)

calculate the new t_{stay_delay}

end for

end for

end for

3.7 Experimental results and analysis for the graph based agent oriented model

We use our model to simulate building occupancy for various agent sizes for normal occupancy and emergency evacuation. Here, we present our analysis of the model for emergency evacuation. Here we present a simple building structure, but we can easily expand it to include complex structure like stadiums or high level buildings. We use the building structure shown in Figure 3.2 where we assume that node 4 will be the safe zone i.e. during evacuation everyone will move towards this node. In the current model, we can vary the number of occupants, their speed, the number of agents who knows the direction of node 4, the size of the group and width the doors. In Figure 3.3 we show the change in a number of occupants in node 2 and node 4 during evacuation with total occupants in the building 500, 2000, 3500, 4000, 6000, and 8000 (all agents know how to move to node 4). In node 2 there are two gates connected to node 4, so most occupants can escape to node 4 immediately. Node 1 to Node 2 has only one door so the occupants from node 1 reach to node 2 after sometime due to congestion and then move to node 4. Thus, first we observe sharp decrease and then a decreasing curve for escaping occupants from node 1.

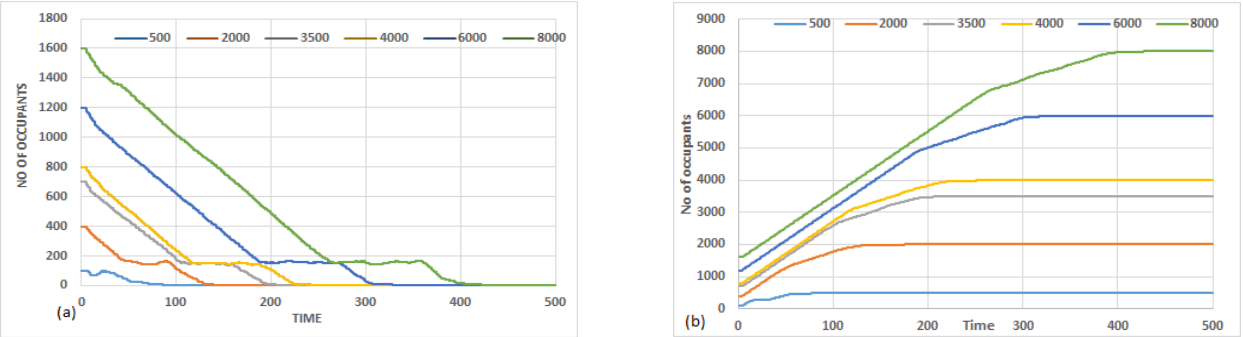


Figure 3.3. Number of occupants (a) node 2 (b) node 4

Figure 3.4(a) shows the effect of increasing the width of the exit link to the safety node. The flow rate of the link is related to the door width and density as shown in equation 4.3. We take 4000 occupants and assume that they will be in rooms other than node 4 which is the safe node. The agents will move towards the node 4 depending on the flow rate of the doors. The width of the doors is set at 1m, 2m, 3m, 4m, 6m and 8m. We assume that there will be no blocking of the link due to the congestion and at least a certain number of occupants can pass the link at each time step. As assumed we find that more occupants can reach the safe node early when the width of the link is increased. Figure 3.4(b) shows the occupants reaching the node 4 safely for various percentage of occupants who knows the direction of the safe node. The simulation is to show the behavior of occupants where they tend to follow a crowd. The given percentage (20%, 50% etc.) of the occupants knows the direction so they move directly towards that node, but the rest just follow a group of crowd as a herd and when they reach the safe node they stay there. The agents follow the group based on the probability of the crowd density at the exit doors. During the evacuation, all the occupants know that they need to head to safety, so they try to reach the safe node. To find the safe node they assume that the door which is most crowded will lead to safety since normally people will go towards that door where the people are trying to flee. As such this

experiment tries to simulate the crowd following behavior of occupants. As assumed, as the percentage of people who know the direction increases the people moving to the safe node also increases. The door width is set at 3m and number of occupants is set at 4000. The non-linearity of the graph is due to the flow rate which lets only a certain number of people through the doors, if the flow rate is high then at 100% the curve will tend to be linear as everyone will come to the room freely.

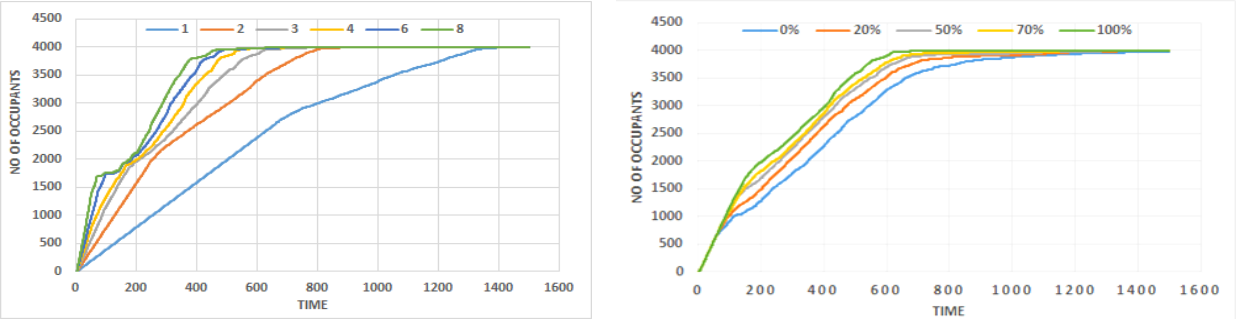


Figure 3.4 (a) Occupants in safe node with varying link size (b) Occupants in safe node with varying agents knowing direction to safe node

Figure 3.5 shows the number of occupants in node 4 when agents forms groups of various sizes (1 i.e. no grouping, 20 and 40). We take 4000 agents and width of each exit as 3m. Grouping is one of the major behavior of occupants and they tend to move in groups during various conditions. We assume that a group will wait for all their group members before leaving a node and when everyone reaches a new place then only decide to move to a new place. All the nodes will have some occupants initially and then they will move towards node 4 in groups. In the figure, we see that for group size 1 more occupants reach in room earlier than larger group sizes. When people tend to move in groups they must wait for everyone before making a new decision as such as the group size increases people take more time to move which is observed in the figure.

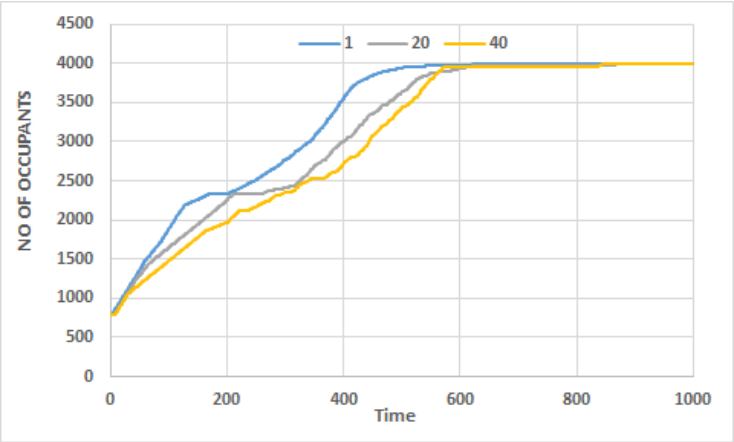
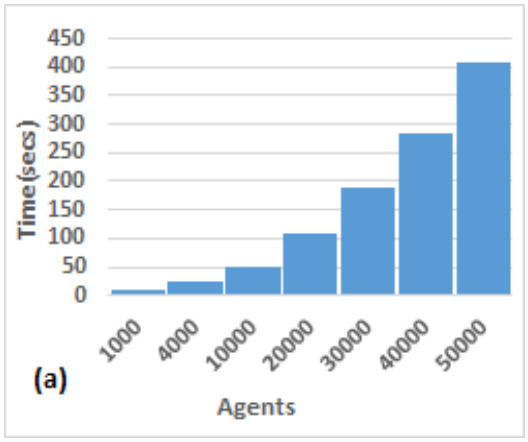
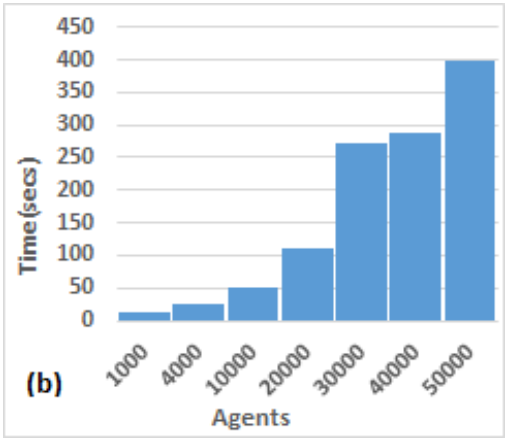


Figure 3.5 Node 4 occupants with varying group size

Figure 3.6 below shows the computation time was taken to simulate for various agent size. Figure 3.6(a) shows the results for 100% agents with direction information and Figure 3.6(b) shows the result for agents with 50% information. In Figure 3.6(c) we show the result of increasing the number of agents for a group size of 20. We run the simulation in a normal PC of Intel Core i5 and 4GB of memory and run for 2000 time-steps. For agent size greater than 10000, we increase the node size and link width. The simulation is run for several times and computational time is averaged. Next, we combine the behaviors of agents, so some of the agents have information of the safe exit, some will form groups (they will also have information of the exit) and some will follow the crowd. In Figure 3.7 we present the computational time for 10% agents having information, 40% forming groups and rest following the crowd. From the graphs, we see that we can achieve good computational time for large agents in a simple machine which gives us direction to use larger machines in case we need to achieve more speedup. At the observed resource use, we can increase the structure and agents as well as their complexity and yet achieve a reasonable speed up than the existing works.





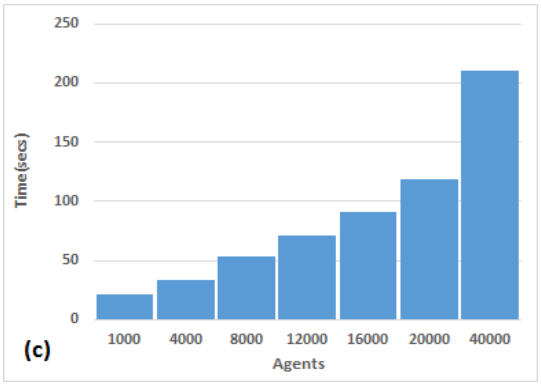


Figure 3.6 Computational time for agents (a) with 100% agents knowing direction (b) with 50% agents knowing direction (c) agents with group size 20

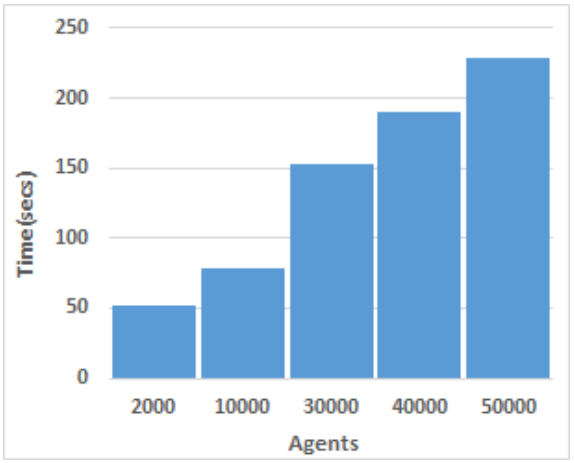


Figure 3.7 Computational time for mixed agents

3.8 Graph based agent oriented model for a pedestrian tunnel simulation

In this section, we use our developed graph based agent oriented model for evaluating a pedestrian tunnel simulation. Tunnels are normally used to provide transportation for vehicles, however, in some places, they provide transportation to pedestrians as well. Globally, tunnels are used in coal mines, underground constructions, and in some cases, even to connect two nearby locations for people to pass through during matches, sporting events, religious, or cultural festivals, and so on. When many people move through a closed structure like a tunnel, even a small incident can cause chaos resulting in the instantaneous will to escape, thus, creating a stampede. The incident at a point may propagate through other regions causing damage to all people inside the tunnel. As such, the creation of such structure requires thorough knowledge and analysis of its capacity and flow. Also, the building requires a study on how to improve the safety of the pedestrian during various dangerous circumstances. In literature, we find research on tunnels for vehicles, but not for pedestrians. However, because a single incident or accident in a pedestrian tunnel may lead to a large number of deaths, there is a need for a study on tunnel simulation. In this work, we have used our model to create a tunnel-like structure where pedestrians move from one end to another. Simulation has been performed for movement of 2000 to 6000 pedestrians. We have computed the average flow of occupants across various sections of the tunnel and presented the result.

In our experiment, the length of the tunnel is 1000 m and width is 20 m. The pedestrians will carry the speed of 2, 1, or 0.8 m/s. The pedestrians arrive at entrance terminal and walk according to their designated speed to the exit of the terminal. The occupants entering the tunnel is simulated by a given input pattern of increasing until a certain number and decreasing to zero. When there are more pedestrians, it will form congestion and will slow down the movement

creating a gap of movement between two or more groups. In such a case, the overall speed of each pedestrian is reduced, and they move slowly until the congestion is cleared.

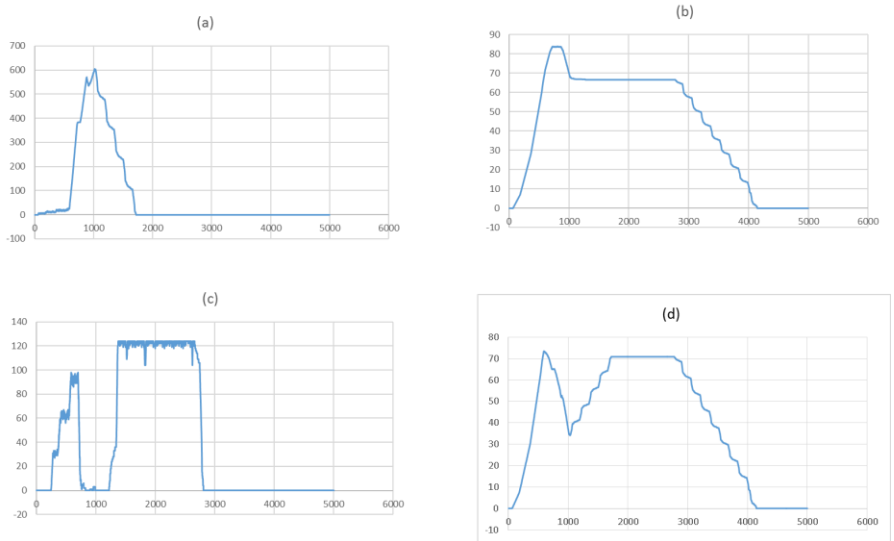
Figure 3.8 shows the structure and the pedestrians moving (in black dots). In the simulation, the pedestrian's input rate is normal in the beginning but increases after a while. As such, the pedestrians move normally at the beginning, but after some time, because of the increase rate, there are congestions as represented by in the blue region in the figure. We can also observe that there is some gap in the middle section which shows that the pedestrians who arrived later are slow due to the congestion.

Figure 3.8. Pedestrians in a tunnel

Figure 3.9(a) shows the input for about 2000 occupants. The number of occupants gradually increases and then reduces. Figure 3.9(b) shows the flow of occupants through a middle section of the tunnel. Figure 3.9(c) shows the average number of pedestrians through the tunnel including the input whereas Figure 3.9(d) shows the average number of pedestrians crossing the tunnel and does not consider the input pedestrian. The figures demonstrate that when the volume of the pedestrian is high, we can observe some gaps caused due to congestion and it takes some time to fill up. Figure 3.10 shows similar results, but for 6000 occupants. In Figure 3.10(c) and (d), we do not observe the change in the number of pedestrians due to delay, since there are many pedestrians and they take a longer time to exit due to congestion. There are more pedestrians in the system which increase the average number of people in the tunnel.

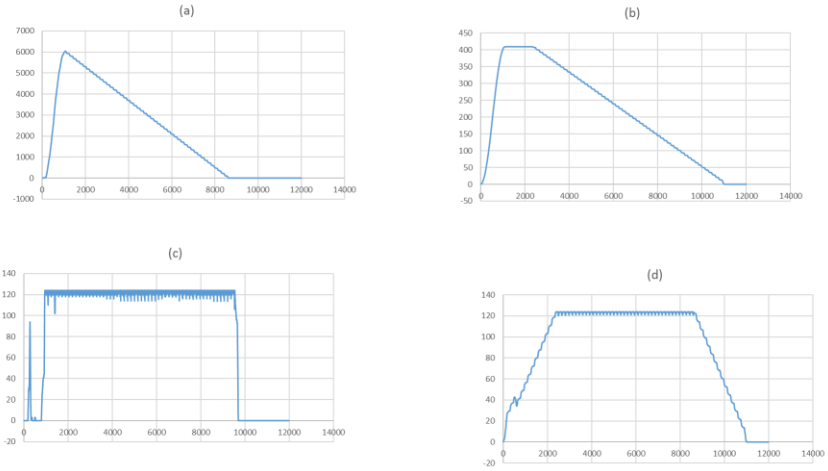
Figure 3.11 shows the flow rate at the exit end for (a) 2000 and (b) 6000 pedestrians, which clearly shows the gap created due to congestion even though there is no gap in input. From the figure, we can analyze the patterns of pedestrians' flow at the mid-section and the end of the tunnel for various size of the input. We see that as the number of pedestrians increase, the graph becomes linear. It is because when there is a high volume of pedestrians, they become congested and they cannot act independently. They are constrained by the space so they can't move freely. As such, their average speed becomes the average speed of the group which is very slow.

In this work, we could observe the behavior of pedestrians when they moved from one end of the tunnel to another. Based on their speed and input rate, we observed their movement dynamics across the tunnel and observed their flow rate. The model allows us to experiment with various input rates, various type of pedestrian and perform the analysis like flow rate, time of exit, pedestrian flow at different sections and time.



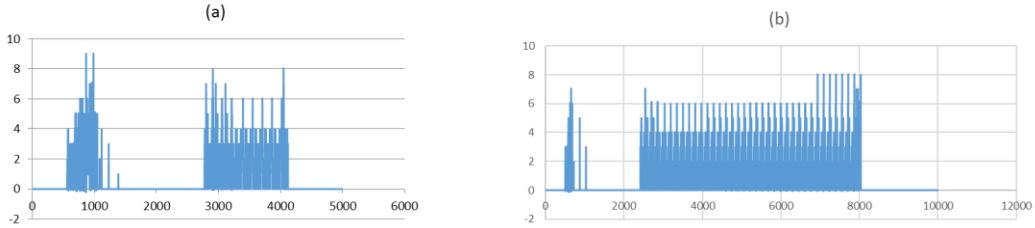
X-axis: Time step, Y-axis: No of pedestrians

Figure 3.9. 6000 Pedestrians (a) Input rate (b) Average flow rate including source (c) Flow rate at mid-section (d) Average flow rate without source



X-axis: Time step, Y-axis: No of pedestrians

Figure 3.10. 2000 Pedestrians (a)Input rate (b) Average flow rate including source (c) Flow rate at mid-section (d)Average flow rate without source



X-axis: Time step, Y-axis: No of pedestrians

Figure 3.11. Flow rate at exit end (a) 2000 Pedestrians (b) 6000 Pedestrians

3.9 Hybrid model

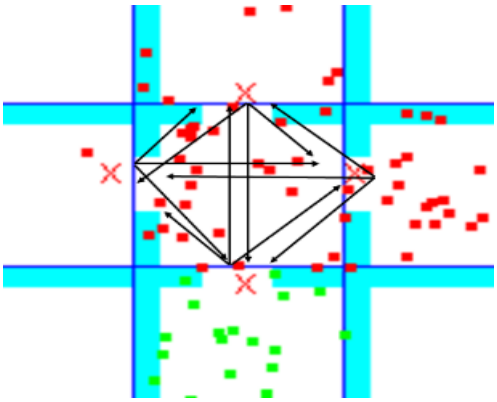


Figure 3.12 Agents crossing paths

In buildings, when occupants move around occupant's path cross mainly in areas like hallways and intersections. The path may be obstructed causing delay or rerouting, and the effect is more when there are more people. Similarly, obstruction due to environment factors like road closure, walls also may cause some effect in the movement. As shown in Figure 3.12, the black arrows show the occupant's possible paths from one exit to others. We can see that the possible path is crossing in different directions. Two or more occupants crossing opposite paths may block each other. This kind of agent interactions with each other and the environment create emergent patterns that might be an important feature for certain regions of the environment. It might not be necessary to model the whole system with agents, but certain regions like intersections, the cross path might be best represented by pure agent based systems since the graph model is unable to create such emergent patterns. In crowds, graph based manage speed using equations based on density and do not consider the effect of agent interactions. Agent collision with each other or environment during crossing paths/obstacles cannot be modeled accurately by graph based.

To properly model such scenarios, we need a model which can effectively switch between agent and graph based as per requirement. Fortunately, our model is quite useful for this implementation. Our graph based agent oriented model is not purely graph based; it consists of agents as well. The model updates the agent properties with the system as well. Hence, we have agents that can represent the state of the system spatially and temporally. As such, it is quite easy to change our model from graph based on pure agent based and vice versa, upon requirement. We create a hybrid model using our graph based agent oriented model, which consists of pure agent based along graph based parts. In the nodes where agent interaction is important, we implement pure agent based and in other nodes, we apply graph based. In the upper level, the whole system is graph based agent oriented, however, in the lower level, we implement pure agents in our nodes

of interest. When an agent enters from graph based region to agent based region, it acquires a specific origin coordinate near the entrance of the node and destination as a location coordinate near the entrance of destination node. Then the agent starts to walk using waypoint graph towards the destination, and after reaching the destination, if the next node is graph based it follows the property of graph based agent. Similarly, when a pure agent from agent based enters the graph based node, it gets its' destination node and computes the move delay and executes. Figure 3.143 shows the transition of the agent from one mode to another. Figure 3.14 shows a snapshot of the hybrid model simulation where the center intersection is the agent based and the black dots represent the pure agents. We can see that the black dots are moving in a formation compared to the red dots that are moving randomly.

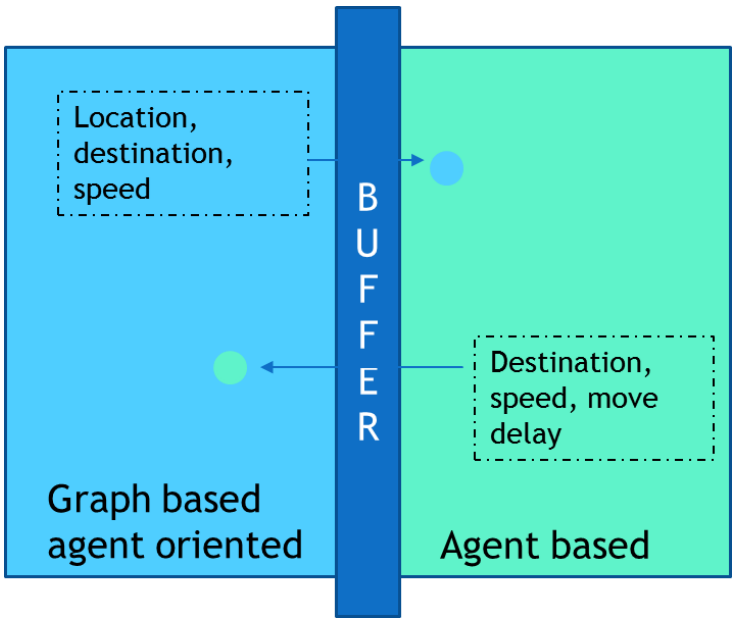


Figure 3.13 Hybrid mode conversion of agents

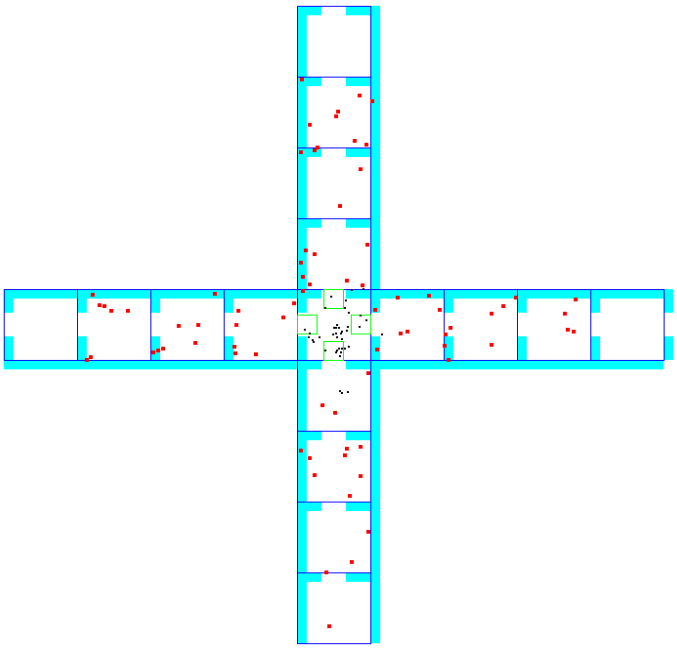


Figure 3.14 Screen shot of hybrid model simulation

4. DATA ASSIMILATION USING GRAPH BASED AGENT ORIENTED MODEL

4.1 Basic Theory

Data assimilation is the process of incorporating observations from the real system into the system model to estimate the state of the system. It is a sequential, time-progressive procedure where the system is compared with incoming new observations and the system state is updated to reflect the observations. The procedure continues to produce a better prediction of the overall system state. A model of a system is not perfect to accurately represent a real system since there might be different inputs and unaccounted changes in the system which might change the behavior of the system from the predicted one. As such, using the observations during the development of the real system assists in keeping the model of the system on track with the real system. Also, the observations from the real system are often sparse, erroneous and do not cover all the aspects of the system, as such the observations cannot be considered as an accurate measurement of the system state.

Data assimilation uses an analysis technique which allows the use of available observations with the available system model to predict the future state of the system accurately. There are two approaches of data assimilation: sequential and non-sequential assimilation. Sequential assimilation considers the observations made in the past up to the point of analysis while non-sequential assimilation uses the observation from future as well. Our work is related with the sequential assimilation which is the case of real-time data assimilation. The dynamic model which represents the real system depends on the type of application. In social science, models like agent based or graph based represent the system, in the atmosphere, and oceanology mathematical models represent the system and so on. These models define the physical constraints and need to have the initial knowledge of the underlying system. The model normally initiates with a known

state and then continuously evolves till a certain specified duration. The observations are the measurements which sample the system in space and time with spatial and temporal scales dependent on the technique used to make the measurements [39]. The measurements provide information of the system and provide an understanding of how the system evolves in space and time.

Both the model and the observations consist of error; observations have an error in precision, bias, and representability, while the model may be incomplete due to lack of understanding or due to processes being omitted to make the problem tractable. Also, the observations are discrete in time and space which will make the information of the system incomplete for the whole space and time. As such appropriate algorithms are required which will combine the observations efficiently with the model [40]. The algorithm provides a set of consistent and objective rules which can evolve the system as true as the real system even with the errors present in the model and observation. A simple approach is a linear interpolation however it cannot effectively model how systems behave in the real world. Least squares estimation is a basic linear method which requires that observation function should be linear and it calculates gain using observation matrix, background error covariance matrix and observation error covariance matrix. The gain is then used to calculate analysis error which is equivalent to a problem that minimizes the cost function calculated by summing background term and observation term. From least squares estimation, optimal interpolation methods can be derived where each observation is assigned with weight based on the statistical property of their errors. Optimal interpolation methods minimize the observation errors by determining the value of the gain. This is similar to the three dimensional variational assimilation and four dimensional variational assimilation methods. These methods aim at minimizing a cost function and its gradient represented by

observations weighted plus analysis and short term forecast by their accuracy. Variational assimilation technologies focus on exploring the initial value of the model but do not aim at adjusting the current state of the system at analysis time. To adjust the system state at analysis time, Bayesian filtering is a viable method. Bayesian filtering methods estimate a system's state from noisy observations and is derived from Bayes theorem. According to Bayes rules, for events *A* and *B*, provided that *P(B)* is not zero, then the following equation holds:

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)} \qquad (4.1)$$

The equation can be applied in the recursive Bayesian filtering problem, and we can get the system evolution model for calculating the transition prior as

$$x_t = g(x_{t-1}, v_t) \sim p(x_t|x_{t-1}) \qquad (4.2)$$

where *x* is the system state variable, *v* is the system noise and *g()* is the system transition function. Also, the observation model for calculating likelihood can be obtained as

$$y_t = h(x_t, o_t) \sim p(y_t|x_t) \qquad (4.3)$$

where *y* is the observation variable, *o* is the observation noise and *h* is the measurement function. Now assuming the system is Markovian with initial distribution of the system state and historical observation vector, we can estimate the posterior distribution of the system using Bayesian theorem as

$$p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_{t|}|y_{1:t-1})} \qquad (4.4)$$

The Kalman filter recursively performs the prediction and update to estimate the system. It is an optimal filter for the linear and Gaussian system, however it is not appropriate for non-linear, non-Gaussian, multimodal state estimation problems. Researchers have proposed various extensions to Kalman filter to over these limitations like Extended Kalman filter, Limited

Extended Kalman filter, Unscented Kalman filter [30]. Ensemble Kalman filter uses a set of ensemble members to approximate the distribution of the state of the target system to solve the problem of high dimensionality. It calculates the covariance of the ensemble members, and then use it to update the estimate of each ensemble members, unlike the standard Kalman filter which calculates the covariance matrix. However, it assumes the system noise and observation noise to be Gaussian, so it does not perform well for non-Gaussian. The Kalman filter requires that the system must be written in analytic forms and requires that system to be linear and system noise and observation noise follow Gaussian distribution, and some cannot handle multimodal distribution. As such we use Sequential Monte Carlo Methods also known as Particle Filters which can be applied to dynamic systems with non-linear behaviors and non-Gaussian noise. Particle Filters approximate the state of dynamic systems using particles and associated weights. PF works by formulating a non-linear state-space model of a generic form containing a state transition function and a measurement function as below:

$$x_t \sim p(x_t|x_{t-1}) \qquad (4.5)$$

$$y_t \sim p(y_t|x_t)$$

where x_t represents the system state at time t evolved from the previous state at time $t-1$, y_t represents the observation at time t for system state at time t. Particle filters use a set of particles to approximate the posterior distribution of the system state. The posterior distribution of the system state in particle filters is represented by some weighted particles whose weights are calculated based on state transition, proposal distribution, and likelihood of the particles. The PF follow a prediction update methodology at each iteration where it samples particles through a proposal density know as importance density. After prediction, each particle is assigned a weight which is calculated through the likelihood density based on observations. Formally, the particle

filter is derived from sequential importance sampling (SIS), where the posterior distribution is approximated by

$$p(x_t|y_{1:t}) \approx \sum_{i=1}^{N} w_t^{(i)}\, \delta(x_t - x_t^{(i)}) \qquad (4.6)$$

Here *w* is the weight of the particle which is updated using

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)},y_t)} \qquad (4.7)$$

The particle filter performs three main steps at each iteration: sampling, weight calculation and resampling [13]. The sampling step evolves the system state of each particle to the next data assimilation time point; the weight calculation step computes the weights of particles based on observation data, and the resampling step selects a new set of particles based on particles' normalized weights. Sequential importance sampling suffers from a major problem of degeneration in which after a few iterations one particle will have high weight, and other have negligible weight thus not contributing to the state estimation. To solve the problem, an additional resampling step is applied for eliminating particles with low weights and multiplying particles with high weights. At each step, the samples are drawn from the proposal distribution $q(x_t^i|x_{t-1}^i, y_t)$ when this distribution is chosen to be the system transition prior $p(x_t^i|x_{t-1}^i)$, equation 4.4 can be transformed to:

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p\left(y_t\middle|x_t^{(i)}\right)p\left(x_t^{(i)}\middle|x_{t-1}^{(i)}\right)}{p\left(x_t^{(i)}\middle|x_{t-1_t}^{(i)}\right)} = w_{t-1}^{(i)} p\left(y_t\middle|x_t^{(i)}\right) \qquad (4.8)$$

So, the update of weight depends only on the likelihood of the particles.

4.2 Data assimilation framework

In our research, we use particle filter as the algorithm for the data assimilation. The observations are the information of the system which in our case for occupancy estimation is the

provided by the sensor data collected from the room, which is the number of occupants collected from the sensor in the room. We use the basic particle filter algorithm also known as the Bootstrap filter algorithm which utilizes the observation to update the state currently. In the pure agent based model, the state of the system is proportional to the number of agents. Hence, as the number of agents increase, the state of the system increases due to which it becomes computationally expensive to maintain a large number of particles. In our work, we consider only the node information as the state which consists of a total number of agents, direction probability of agents and agents in the queue for each node. Formally the state of the system is defined as:

$$S_t = <s^1_t, s^2_t, \ldots . s^n_t > \quad (4.9)$$

where n is the number of nodes, s^n_t is the state of a node at time t and defined as

$$s^n_t = <a_n, a_q, p_d> \quad (4.10)$$

where, a_n is the no of agents in node n, a_q no of agents in each queue of node n, p_d is direction probability for agents in node n. So

$$a_n = <a_1, a_2 \ldots \ldots . a_m > \quad (4.11)$$

where, m is the number of agents in node m. a_q is the number of agents in queue and

$$a_q = = <aq_1, aq_2 \ldots \ldots . aq_m > \quad (4.12)$$

where, q_m is the number of queues going out from the node. It depends on the edges in the current node, p_d is the direction probability, and it is probability pointing the destination of the agents present in the node. It sums to 100% and points how many percentages of the total agents are moving in a certain direction from the node. It also consists of agents staying in the node. Formally we can state direction probability as

$$p_d = <p_{d1}, p_{d2}, \ldots . p_{dn}> \quad (4.13)$$

where *dn* is the total directions each node can have. The directions are user defined and depend on the structure of the building. The system transition function which evolves the system from the current system is formulated formally as:

$$S_{t+1} = GAMTransition(S_t) + Q_t \quad (4.14)$$

where, S_t is the system state at time *t*, *GAMTransition* represents the state transition function, and Q_t is the system processing noise. *GAMTransition* defines how the state of a particle evolves from time step *t* (S_t) to *t+1* (S_{t+1}). This function is based on the graph based agent oriented model developed by us in Section 3.

The observation data for the particle filter is collected from a set of sensors placed in selected node. We do not focus on collecting data from the sensors but assume that the rooms have sensors which can compute the total number of occupants in each time step. If we have *n* number of sensors deployed in our system; the sensors not be in every node, then the observation vector is formally defined as:

$$O = <S_1, S_{2\ldots\ldots} S_n>$$

$$S_n = Obs(n) + m_t \quad (4.15)$$

$$Obs(n) = <o_n, o_{qn}>$$

where, *Obs(n)* is the observation at room n; which is the number of occupants in room *n* (o_n) and a number of occupants queuing from room *n* (o_{qn}). And m_t is the measurement noise for the observation. The sensors are independent of each other so the importance weight of a particle can be calculated from the likelihood of individual observations.

The transition prior is $p(x_t|x_{t-1})$ is chosen to be the proposal distribution and the weight of the particle is calculated based on the likelihood $p(y_t|x_t)$. Since the measurement is a vector we

define the likelihood as a multivariate Gaussian distribution. The weight of particle is calculated as follows:

$$w_t = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (4.16)$$

where, x is the observation from the real system and μ is the measurement of the particle. σ is the user defined value.

The particle filtering algorithm follows the standard sequential importance sampling with resampling (SISR) procedure. The particles are initialized with given initial conditions, which we assume is known to us. So, the particle filtering knows the initialization pattern of the occupants through the input nodes. After that, samples are drawn by advancing the state of each particle using the state transition function for a selected period Δt. The weight of each particle is calculated based on the comparison between the observations generated by the sensor model for each particle and the real observation data. The weights are then normalized so that in the sampling step, particles with higher weights have greater chances to be selected and reproduced.

Since the particles faces the problem of degeneracy we perform standard resampling which avoids the problem by eliminating the particles with small importance weights and concentrating on the particles with higher weights. It samples n times (no of particles) which replaces from the set of particles $x_t^{(i)}$ according to the importance weights $w_t^{(i)}$. The algorithm of PF with re-sampling is as follows:

Sequential importance resampling PF

1. Initialization:

   Set time $K=0$

   For $i=1\ldots.N$ sample $x_0^i \sim p(x_0)$

Set $k = 1$

2. Importance sampling step

   For $i = 1 \ldots N$ sample $\tilde{x}_k^i \sim q(x_k \mid x_{k-1}^i)$

   Set $\tilde{x}_{0:k}^i = (x_{0:k-1}^i, x_k^i)$

3. For $I = 1 \ldots N$ compute weights $w^i{}_k$

   Normalize weights $\tilde{w}_k^i = w_k^i / \sum_{j=1}^{N} w_k^j$

4. Resampling step

   Resample with replacement N particles: $(x_{0:k}^i; i = 1, \ldots, N)$ from the set: $(\tilde{x}_{0:k}^i; i = 1, \ldots, N)$ based on the normalized importance weights $\tilde{w}_k^i$

5. Set $k \to k+1$

Proceed to the importance sampling step as the next measurement arrives

We apply the SIR PF algorithm to our data assimilation. The basic algorithm for SIR in our application is as follow:

1. *Initialization*

   For particle $i = 1 \ldots N$, initialize agents based on given initial parameters and randomly assign destinations

2. *Importance sampling step*

   For all particles execute Δt steps of simulation defined by the system transition model (graph based agent oriented model)

   For all particles, calculate weights based on observation from real system and particles

Normalize the weights

3. *Resampling step*

Resample based on normalized weights

Add noise (direction, no of occupants moving)

4. Repeat from step 2 as next observation arrives

After resampling, we select the particles which represents the best estimates of the system state. From the estimated states, we need to continue our simulation by generating other parameters of the system. In our research, we consider only the node information as the state for the data assimilation. It is done to simplify the complexity of the particles so that we can reasonably increase the number of particles with the dimension of the system. Since after the resampling, we update only the state of the system, we need to update other parameters of the system from the new state. We need to generate the new direction and moving delay time (Section 3.2) of each agents. The new direction of the agents is calculated from direction probability array of the node. For each agents in queue, their next destination is set towards the direction of the queue node. For other nodes, their next destination is generated using the direction probability sampled from the particle filter. The direction array may be direction like left, right, top, bottom, stay etc. After generating the destination, move delay is generated randomly based on the distance between the current node and the destination node. Then the rest of the simulation is continued using the system model, the graph based agent oriented model. The whole system regeneration from the resampled states is a crucial part of the simulation model and we plan to implement other efficient algorithms in the future. The current framework can be represented using the figure below.

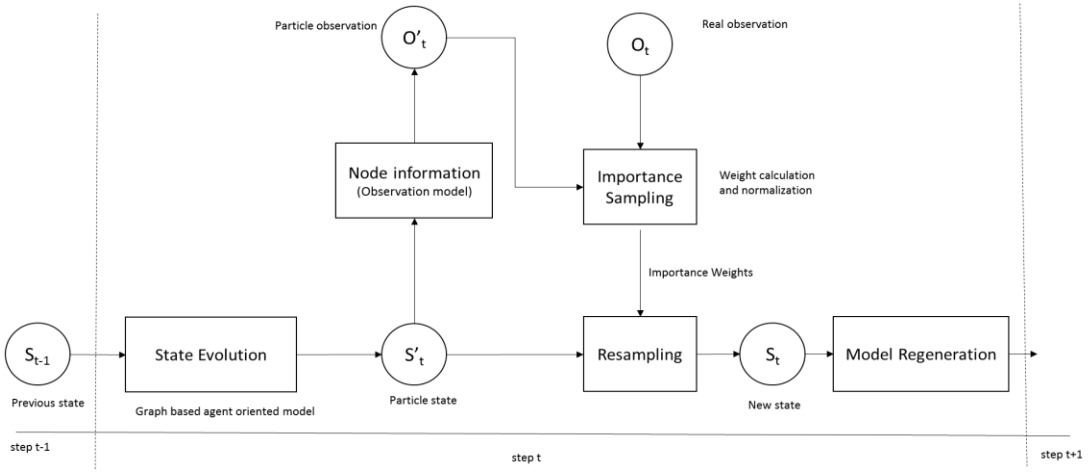


Figure 4.1 Data assimilation framework

4.3 Data assimilation using direct sensor data

In the particle filter algorithm, we utilize sensor data by calculating the likelihood weights of the particle filters. As such, we do not directly use the sensor data into the model. The particles resampled represent the estimated occupancy but in some cases, they may not be able to represent the true state. As discussed before, particle filters suffer from the problem of degeneracy for which we perform resampling; we eliminate particles with lower weights and multiplying particles with higher weights so that the variance of the weight is reduced. However, resampling always eliminates some particles which reduces the diversity and after some runs there is high probability that all particles become identical. As such the particles are unable to represent the real state of the system and it is too late to recover. In such scenarios, we can take the help of available sensor data to recognize the better particle filters. The data obtained from the sensors are inaccurate and cannot represent the real system wholly, but they are the observations of the system and provide some information of the system. For example, a sensor might not identify the exact number of occupants in an area, however it can identify if there are people more than a limit which recognizes as a congestion. As such, using sensor data to detect number of people is not a good practice, but it can

be used to detect certain events like congestion, flow direction, evacuation. In this work, we utilize some information from the sensor data to improve our data assimilation framework, especially in situations when there are long congestions. As the number of nodes and occupants increase, regardless of the model, more computation resources are required. With the increase in complexity of the model, the accuracy of the data assimilation also decreases when we restrict the number of particles. As such utilizing sensor data can help improve the result of sensor data while maintaining the number of particles in the model.

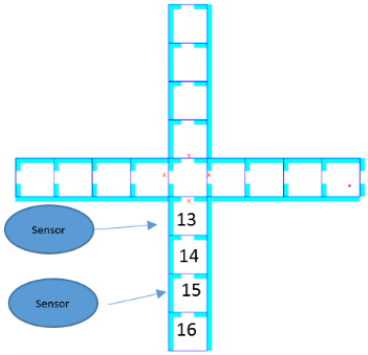


Figure 4.2 Issue caused by placement of sensors

For example, in Figure 4.2, if Node 13 and Node 15 has high occupancy, then most probably Node 14 has high occupancy. But if Node 13 has high occupancy and Node 15 has few, two cases may arise: first Node 14 may be main destination and have high occupancy, second Node 13 may be main destination and Node 14 may not have high occupancy. For the total occupants, if there are few occupants in other sensors, Node 14 is the main destination, otherwise if other sensors have significant number of occupants, Node 13 is the main destination. So, it is difficult to estimate the real occupancy in such ambiguous nodes. In such a case, it becomes useful to utilize the information from available sensor data. Obviously, we cannot directly put the sensor

data as the estimated occupancy, instead we need an algorithm to utilize it in our data assimilation framework.

To utilize the sensor data, we create some new particles from the available sensor data. These new particles are combined with the existing particles to create a new particle. In the standard particle filter resampling, we treat the entire system state as one system and so, in building occupancy a particle consists of various nodes with occupancy information. We can disintegrate our whole system into a system consisting of various smaller states where each state is a sub-state of a single system state. These sub-states, when brought together, represents the overall state of the system. In our building occupancy simulation, we can break the system state of building occupancy simulation into sub-states consisting of individual nodes as a sub-state with state variables like occupancy count, occupancy behavior, and node properties. We consider each node as a sub-state where a node has some occupants, queue size, and each occupant has a destination, speed and other properties. When this information is combined for all the nodes, it provides us with the information of the overall environment as a whole. The sensors used in our work is placed in one of such node and provides information about that sub-state.

Using the real sensor data, we create a particle with occupancy distribution based on the available sensor data. We will have nodes which have sensors in them, as such it is easier to get an estimate for such nodes. However, we will also have nodes which won't have sensor data. For the nodes which do not have the sensors, we can use some methods like Gaussian distribution to calculate the occupancy. Since the sensors in a building occupancy are correlated, the sensor data from a node will influence its neighboring nodes without the sensor. Also, we do not generate occupancy data for all the nodes; rather we randomly select nodes to generate the distribution. As such we will have a certain (predefined) number of particles with sub-states as nodes, and some of

the sub-states might be null. Now we select the standard particles and improve them by mixing a random number of sub-states from the new particles generated from sensor data. These new particles will have their sub-states from both standard and sensor data generated particles. We follow rest of the standard resampling procedure from the bootstrap filter. In this work, we set the number of occupants in each node as the sub-state and we can generate other properties using the data assimilation model. Formally, let us have a set *St* of new n particles generated using sensor data,

$$St = <D1, D2, ...Dn> \quad (4.17)$$

$$Dn = <N1, N2....Nm>$$

where *Dn* represents a particle and *Nm* represents a sub-state at time *t*.

Let us select a set *S't* from the standard particles to use the new particles

$$S't = <D'1, D'2, ...D'n> \quad (4.18)$$

$$D'n = <N'1, N'2....N'm>$$

With these set of particles, a new particle is constructed as

$$\tilde{x} = < \widetilde{D_1}, \widetilde{D_2} \dots \dots \widetilde{D_n} > \quad (4.19)$$

where $\widetilde{D_n}$=*select*(D_n,D'_n)

And select (X, Y) is a function to select a sub-state based on a probability from either X or Y.

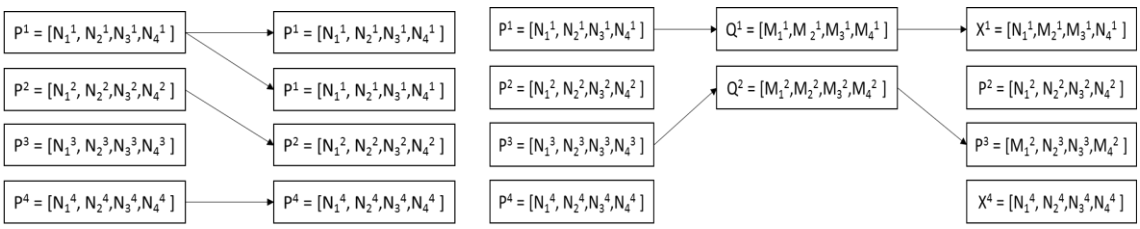


Figure 4.3 a)Standard Resampling b). Resampling with particles from sensor data

Figure 4.3 represents how the new resampling method utilizes the sensor data works. In this example, there are four particles each having four sub-states *N1, N2, N3* and *N4*. In Figure 4.3(a), we can see the standard resampling procedure in which, based on weights particle *P1* is sampled twice, *P2* and *P3* are sampled once and *P3* is not sampled. In Figure 4.3(b), we have two new particles generated from sensor data. We select particle *P1* and *P3* randomly to combine with the new particles and get two new particles and two original particles. In Figure 4.3(b), we have the 1st and 3rd particle as a new particle in which the first particle has 1st and 4th sub state from original particle and 2nd, 3rd from the new particle from sensor data. These particles now go through the steps of resampling and have a better probability of representing the real system. We can define the algorithm for resampling using the new method as follows:

Data assimilation using particles from sensor data

Resampling step:

Input: The set of particles $< \tilde{x}_t^{(i)} : i = 1 \dots N >$ with associated weights $< \widetilde{w}_t^{(i)} : i = 1 \dots N >$,

Set of particles from sensor data $< S_t^{(D)} : D = 1 \dots M >$

Output: A new set of particles $< x_t^{(i)} : i = 1 \dots N >$

Select k particles from original particle set to create set St'

For 1 to k

Select a particle *D* from set *St*

Select a particle *D*' from set *St'*

Get sub-states with a probability *p* using *select(D, D')*

Construct new particle $x_t^{(i)} : < c_1^i, c_2^i \dots . c_m^i >$

End for

Normalize the weights based on:

$$w_t^{(i)} = \frac{\widetilde{w}_t^i}{\sum_{j=1}^{N} \widetilde{w}_t^{(j)}} \qquad (4.20)$$

4.4 Data assimilation experiments

4.4.1 Experiment settings

For our experiment, we plan to implement our model in a building occupancy of a large number of people. It is difficult and expensive to obtain the real scenario of building occupancy and sensor information from a physical environment. As such, we have used identical twin experiment to implement our data assimilation framework in a building occupancy simulation. We have considered an occupancy scenario of a busy airport terminal which has different rooms as terminal from where people board airplanes. We use graph based agent oriented model to simulate a real terminal simulation without implementing data assimilation. In real applications, this model will be replaced by a real system. Airport terminal is a good application where there are a lot of people continuously moving towards a destination. The people when reaching the destination, stay there till the boarding announcement and then they board the flight. Often time when there is boarding delay, congestion is created which might affect other people moving through that zone as well. We assume that each room is a node and we employ sensors in selected nodes. The sensors give us the number of occupants in the node and queue. These observations are considered as "true" result and recorded in real time. We conduct another simulation using the data assimilation where the system transition is the graph based agent oriented model. The data assimilation algorithm uses the particle filter algorithm where each particle represents a possible system state. Each particle will also have their sensors which are similar to the real system. The data assimilation compares the observation from the real sensor and observation from the particles to make next

state prediction. Since we assume that we know the initial conditions of the occupants, we use the information in our data assimilation model when initializing the occupants.

The observation data for data assimilation is obtained from the various kinds of sensors placed in different areas of the building. The sensor information is filled with noise, and they are not able to provide 100 % information of the building. Due to various factors like cost, security, convenience sensors cannot be placed in every part of a building, and, they are sampled at certain time intervals. As such it might be difficult to get information of the system in certain time and space. Our research aims in predicting occupancy in areas which are not covered by any kind of sensors. As such we will predict the possible number of occupants in nodes which do not have sensors as well. In our work, we assume that video sensors are installed in selected nodes only so we will have information only in certain areas. We consider the sensor to be 60% accurate and so use a Gaussian noise with mean as the correct number of occupants and variance as 40% of the total occupants in a single node.

We use two kinds of building environments for our experiment. One is a small structure consisting of 17 nodes as shown in Figure 4.4 and another is a bigger structure consisting of 79 nodes as shown in Figure 4.5. The bigger layout is similar to the smaller structure but consists of more nodes and junctions for simulating complex scenarios with a larger number of people. The numbers inside the nodes represent the id of the node. The structure mimics an airport/train terminal type of environment where people enter the system through one or more entrances; then they move to one of the terminals for boarding. Boarding represents the activity of leaving the structure like when boarding a bus, train or plane. The nodes can have various size and capacity, and the size of the edge will be dependent on the size of the room. There will be only one door connecting two nodes and occupants will need to choose that door to go to the node. We will create

various scenarios in both the layouts and present the results for real-time occupancy estimation using our framework.

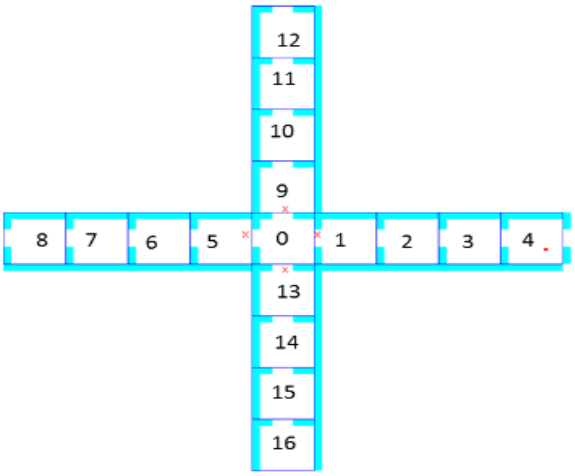


Figure 4.4. Experimental structure for small layout building

Each node represents a section of the terminal, which may be a boarding zone or a walking hallway. We assume that node 4 is the entrance and node 1, 2, 3 are the hallway; node 0 is the intersection and Node 5 to Node 16 are the boarding zones. As such all occupants enter from node 4 and they have a destination to one of the nodes from 5 to 16. Each of the nodes has area of 75 x 75 meters; the intersection node is three times the regular nodes. The capacity of the nodes to accommodate the occupants is directly proportional to their area size. In our experiment, we assume that the capacity of each node is 112 occupants, and the intersection is 3 x 112 occupants. The edge between two nodes has a thickness of 1m and length of 25 meters. For the real system, we set the destination for each experiment, and we create a congestion which clears out slowly after some time. In the data assimilation, the destination of the agents are random and keeps on updating as the particle filter updates the state based on observation. We perform a prediction for special cases which is congestion in a boarding node due to the high volume of occupants arriving in the node before boarding time. Often in airport terminals, people come some hours before the

boarding time. As the boarding time is near, more people arrive at the terminal, as such, it creates congestion at the checkpoint. Normally there is a single checkpoint for security, and due to high volume, it can create congestion at the security checkpoint node. Also, normally the people arrive at their boarding zone, wait till the boarding announcement and start boarding. However, sometimes when there is a delay, they stay there, and other people may arrive there for boarding. This scenario might create congestion as the crowd develops and the congestion will propagate to the incoming nodes. We create various scenarios with different destinations, boarding time and incoming rate and use data assimilation to predict the real scenario.

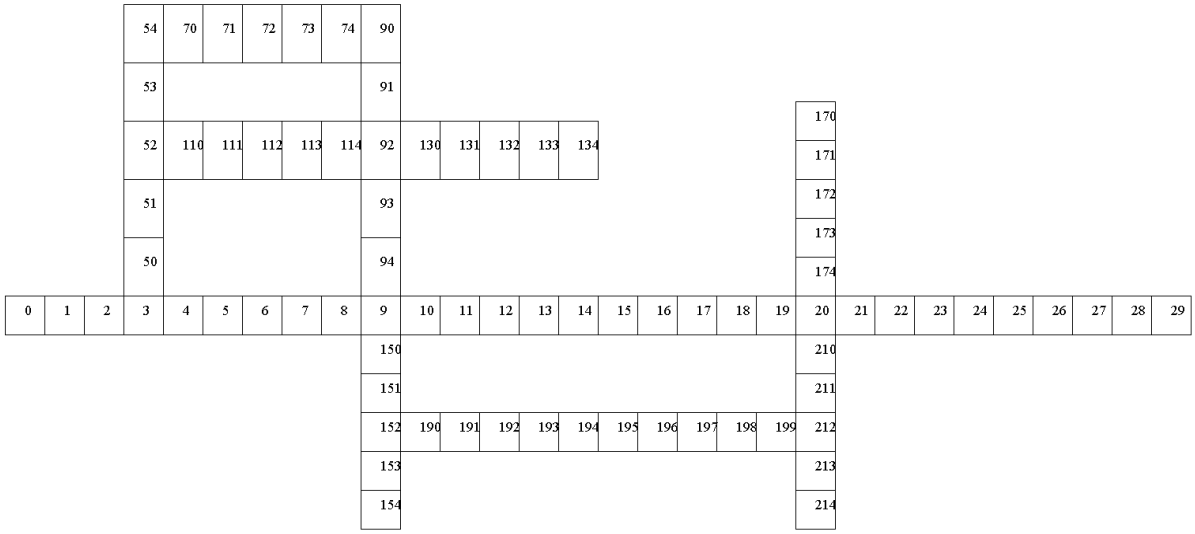


Figure 4.5 Experimental building structure for larger layout

Table 4.1 outlines the experiment motivation with parameters used and expected result and remarks. We perform various experiments to validate our data assimilation model in various building occupancy simulation.

Table 4.1 Experimental Design

Exp.	Motivation	Parameters	Expected Result	Remarks
1	Evaluate the performance of data assimilation for smaller layout	Destination: one, three No of Agents: vary Scenario: congestion, normal	The model efficiently estimates the real system state i.e. the occupancy in various nodes	Experiment verifies the use of data assimilation for building occupancy simulation during various scenarios of congestion and normal behavior
2	Evaluate the performance of data assimilation for larger layout	Sensors: half, one-third, one fifth of the nodes	The model estimates the occupancy with decreasing number of sensors	Experiments verify the use of data assimilation with graph based agent oriented model for occupancy estimation with limited number of sensors
3	RMSE comparison for different sensors	Sensors: half, one-third, one-fifth of the nodes	Accuracy decreases with decrease in sensors	The difference is very less. Thus the model is stable for estimation with few sensors
4	Comparison using particles from direct sensor data	Sensors: Half of the nodes	Performs better than bootstrap filter without using sensor data for few particles and more occupants	Using particles from sensor data improves the estimation when there are large number of occupants so using few particles allows efficiency

4.4.2 Results for smaller layout

We create various scenarios of occupancy to test our data assimilation framework. In the first scenario. we set up sensors in all the nodes to verify the model. We set the destination of the people with 50% to node 7 and 50 % to node 11. The simulation starts from 6 am and the people keep on arriving at the nodes since the start till 10 am. Due to heavy incoming of people, the crowd keeps on increasing and only at 10 am boarding starts. We see that the result is good and matches almost perfectly with the real sensor. Nodes 8, 11, 13, 14, 15, 16 have some occupants as the model currently does not reroute the destination. As such the particles will have a few of the agents with destinations at various nodes.

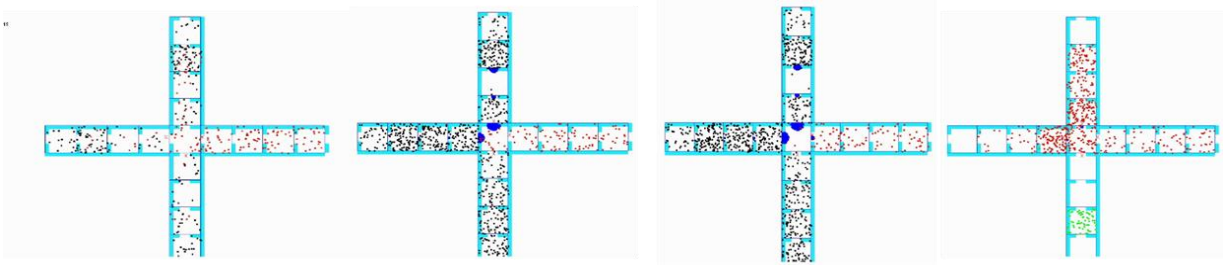


Figure 4.6 Real simulation at increasing time steps for first scenario

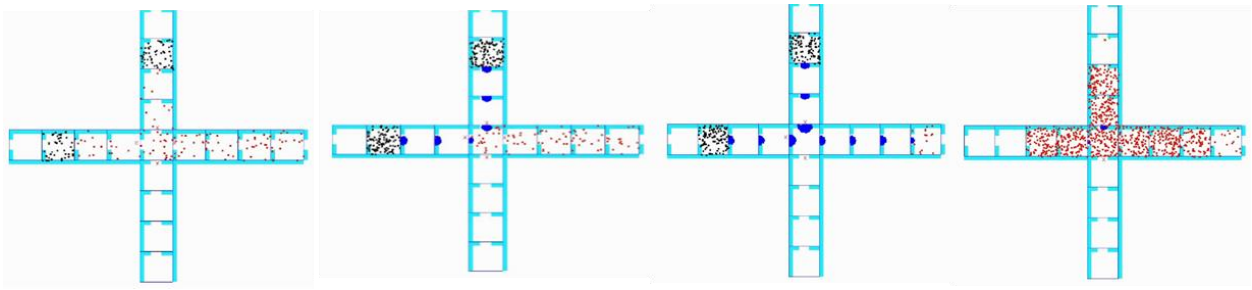


Figure 4.7 Data assimilation simulation at increasing time steps for first scenario

Figure 4.6 shows the screenshots of the real simulation run in which we see that initially, people move to node 7 and 11 (shown by red dots). People keep moving there, but they do not board, which creates congestion show by blue dots near the door. Black dots represent the people reaching their destination, which is node 7 and 11 in this experiment. The congestion in real

simulation spreads towards the entrance node 4. Finally, after the boarding time congestion is cleared and people move towards their destination to board. In Figure 4.7 screenshots of data assimilation run are shown in which we can see people moving randomly, yet more people are moving towards node 7 and 11 (shown by dense black dots in node 7 and 11). Like the real simulation, since more agents move towards node 7 and 11, congestion is created and keeps propagating to the nodes from where people keep coming. Finally, after the boarding starts, the congestion clears and people move to node 7 and 11 for boarding. In some nodes, we can observe some green dots representing the error occupant direction (node 15). These errors can be reduced by increasing the number of particles and using better noise parameters.

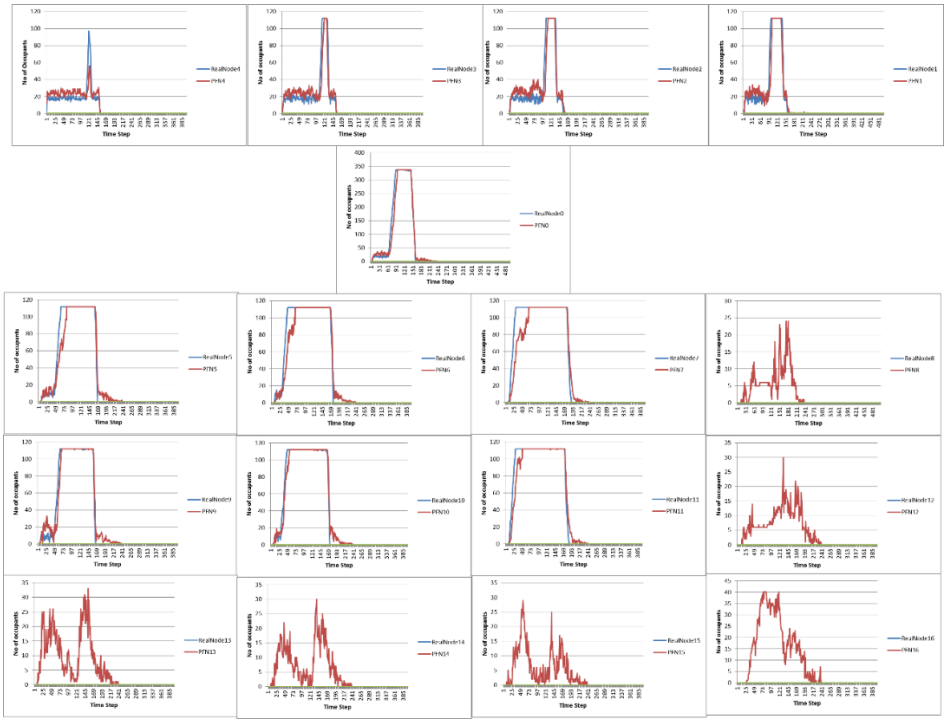


Figure 4.8 Comparing results for first scenario

In Figure 4.8, we show a graph comparing the number of occupants in real and data assimilation simulation. The data assimilation simulation is done for five runs, and the results are averaged. Blue graph shows the real data, and red graph shows the data assimilation result. We

consider mainly the results in destination nodes and the path from the junction (node 7, 6, 5, 0 and node11, 10, 9, 0). We observe that the data assimilation results match the real simulation result and provide a good estimate of the number of occupants in all the nodes. We observe some high error in path node 15, 14, 13, 0 and nodes 8 and 12 which is due to the randomness of the model. The people who move to that branch do not get the opportunity to come back to correct destinations in the current model. We plan to improve the result in future work by applying a filter to those noises.

We use root mean square error (*RMSE*) to compare the results of data assimilation with the ground truth. *RMSE* measures how much error there is between two datasets. *RMSE* usually compares a predicted value and an observed value. We compare the occupancy counts in each node from the data assimilation with the real number of occupants from the real system (ground measurement). *RMSE* is calculated as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \qquad (4.1)$$

Table 4.2 Average RMSE for nodes

Node	Average RMSE
0	11.58126
1	5.759036
2	5.777794
3	5.515986
4	5.141853
5	6.747146
6	6.374019
7	7.577623

8	7.009612
9	5.79183
10	5.55902
11	6.217388
12	7.29934
13	8.318082
14	7.209448
15	5.211005
16	12.94126

Here, x_i is the real data and $\hat{x}_i$ is the predicted data, *N* is the total number of data. We compute *RMSE* for all the nodes and get the average for each Nodes. From the table, we can see that the average *RMSE* is about 5 for node 15 which is quite good, also *RMSE* for other nodes are also in acceptable range

Since our work is to estimate the occupancy of the building even in areas where there are no sensors, in the second scenario, we place sensors only in Node 5,9,13,7,11 and 15. Since Node 0, 1, 2, 3 are hallways where people only move across, the patterns is formed by input and it is not required for sensors to be placed there. Also, we focus on estimating occupancy mainly on the boarding nodes, as such we place sensors in nodes separating by one node. We plan to do future work on where to place sensors so that we can efficiently and accurately predict the occupancy using data assimilation. In scenario 2, we place selected sensors and create a scenario in which the 80% people go to node 15 for boarding and rest go to random nodes. There is a congestion at node 15 since high number of people go to that node which slowly clears as people start boarding after boarding time. We use 200 particles for data assimilation and resample in every 120 time step

where each time step is 1 second. In Figure 4.9, we see the results for the occupancy count in each of the nodes. X-axis represents the time step, Y-axis represent the number of people, blue graph is the real result and green is the result of data assimilation. We see that the result for Node 4, 3, 2, 1, 0, 13, 14, 15 (sequence of nodes for path up to Node 15) is quite good. We find some noise for remaining nodes which are under acceptable range.

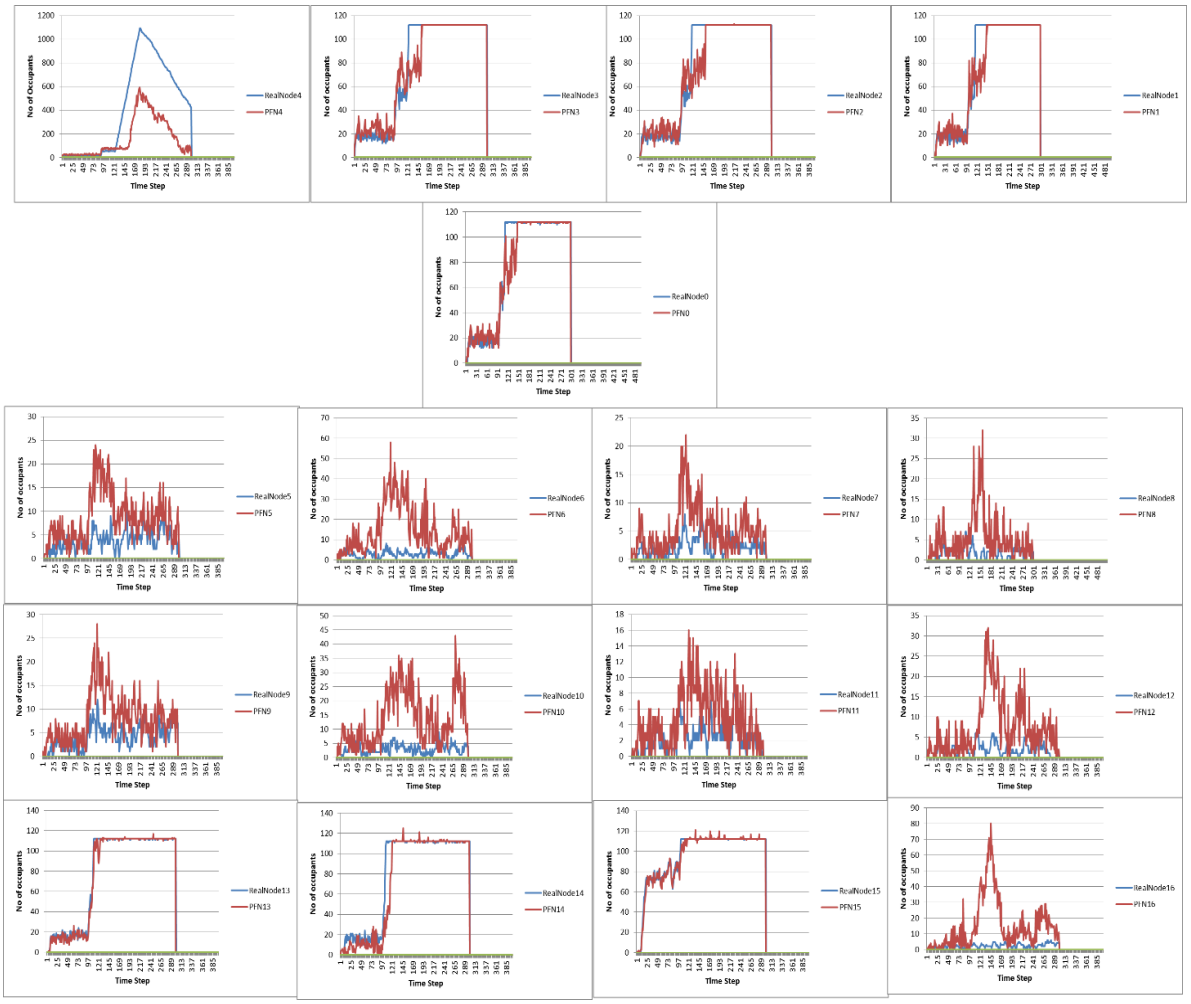


Figure 4.9 Comparing results for second scenario

In the third scenario, 50% of the people have destination node 11, 35% will have node 6 and rest will have random nodes. Here also we use sensors only in Node 5,9,13,7,11 and 15. As for the initialization, assuming the simulation starts at 6 am, we assume that one person arrives

second except for from 7 to 8 am, 5 people arrive every 10 secs and between 11 and 12 am 3 people arrive every 10 secs. This increase in rate will create congestion thus creating high peaks in graphs as shown in node 15. We assume that all the people reaching their destination will board taking time range from 0 to 30 min. Here node 6 does not have sensor yet the results are within acceptable range. Figure 4.10 shows the result of data assimilation and we observe good results. Here also blue graph is the real simulation and red graph is the particle filter result.

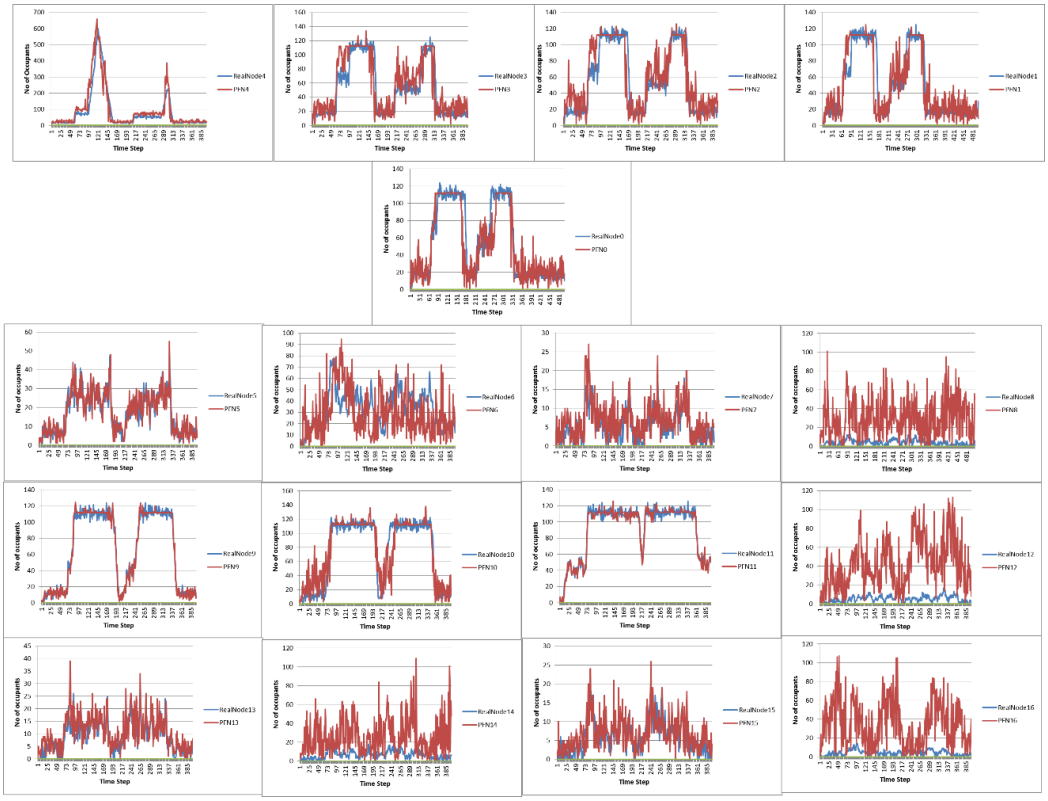


Figure 4.10 Comparing results for third scenario

In the fourth scenario, 30% of the people have destination node 7, 15% have destination node 13 and rest have node 12. Every 20 seconds, a person enters the building until 8 am, and until 10 am 3 persons enter after which the rate is zero. We set the boarding time at 11 am and the boarding rate as 30 minutes. Since people start arriving at their destination at 6 am but the boarding for all nodes start only at 11 am congestion is created which propagates up to the entrance node

(node 4). As such even after the rate is zero, people move from the queue to their destinations from node 4. We can see the results in Figure 4.11 below. Here we find that the results of the data assimilation using particle filter is very good for the main destinations 7, 12 and 13, however there are some errors in other nodes.

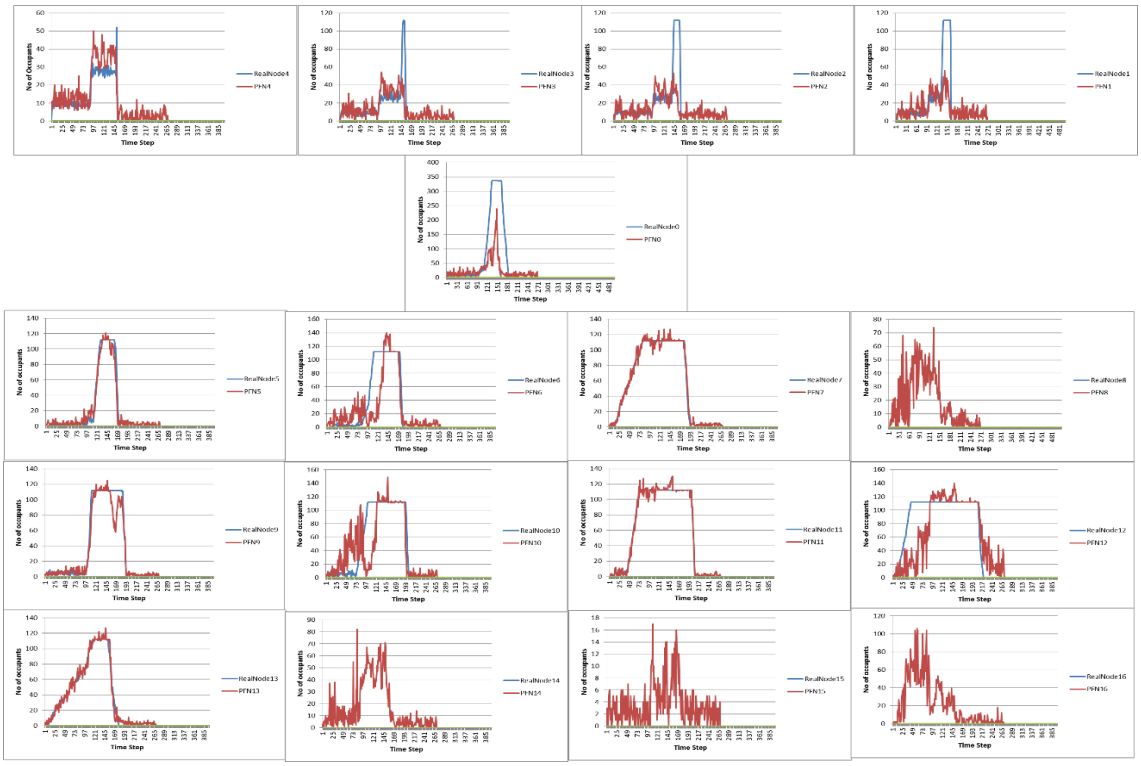


Figure 4.11 Comparing results for fourth scenario

In the fifth scenario, we run simulation to accommodate large number of occupants. We increase the capacity of the nodes to three times and the simulation consists of a maximum of 3000 agents during a simulation time step. The agents keep on entering and leaving the system, hence the total number of agents is much higher (about 15000). The simulation and the data assimilation (200 particles) were running smoothly in our simulation environment, hence we can easily scale our model to include more agents. Here we set node 7, 11 and 3 as destinations with equal probability and 2 people enter the room every 20 second till 11 am after which 4 people enter till

2 pm. In the Figure 4.12 we see that results for node 6 and 10 are not very good. These nodes do not have sensor hence sometimes the result is not perfect, yet they can predict the peak congestion accurately.

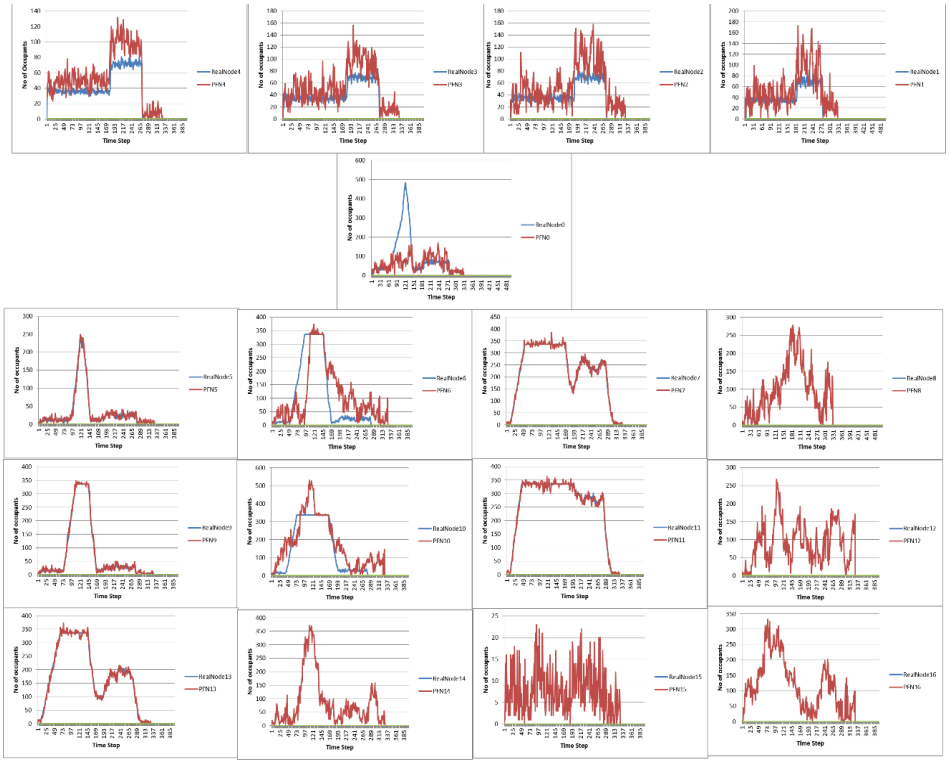


Figure 4.12 Comparing results for fifth scenario

We create a bit complex scenario to test our model in sixth scenario. We divide the boarding time in two halves so the people arrive in two halves first; before 9 am, and second; before 2pm. Till 9 am, the people have three destinations of node 7, 11 and 14. Then people will arrive to board for second half and will have destination of nodes 7, 10 and 13. From 11 am to 2 pm, one person arrives every 20 second and from 3 pm to 5 pm, 3 people arrives every 20 second. The boarding rate is 0 to 20 minutes and node 7 boards at 9 am, node 11 and 14 board at 10 am in the first half. In the second half, node 10 boards at 2 pm, node 7 boards at 4 pm and node 13 boards at 6 pm. In this way, we create a scenario in which in a single simulation run, during first half three

of the nodes have different boarding time and people arrive as per their destination. Similar case occurs in second half but for different destination creating a complex scenario which might occur in real system. In the Figure 4.13, we see that because of the complexity of the system, the output is not as good as the previous simulation however the results of the particle filter show similar pattern as the real system. We hope that the results for this kind of complex scenario can be improved using our method of using observations to create particles directly.

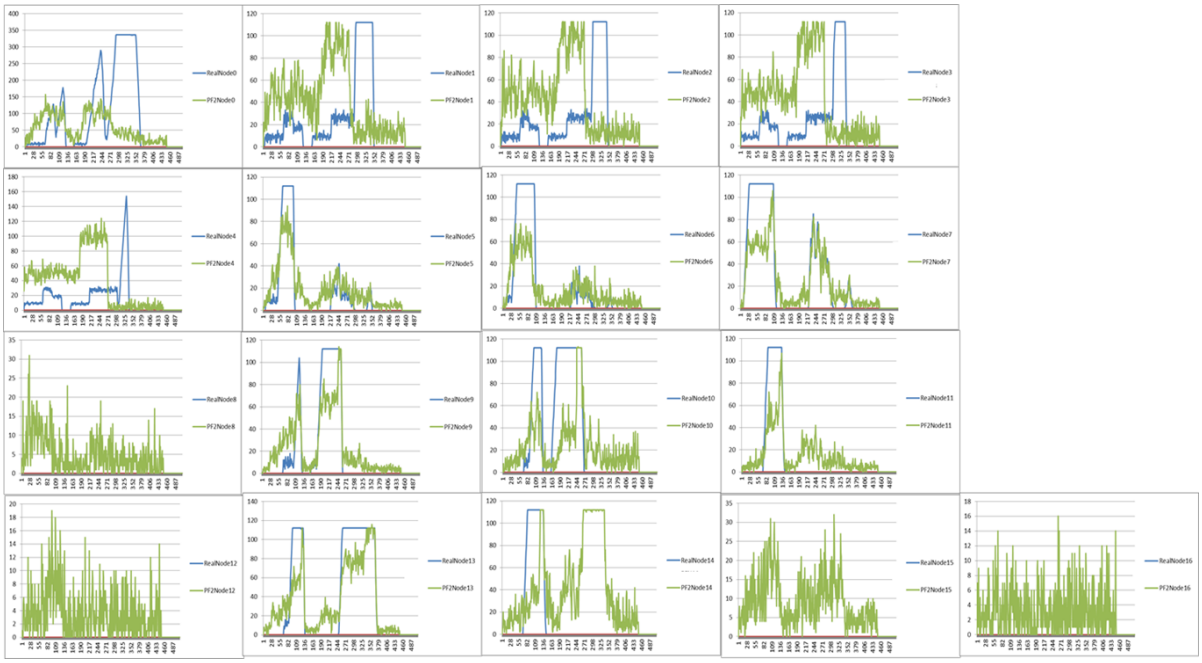


Figure 4.13 Comparing results for sixth scenario

In this scenario, we run a simple case without any congestion to see how our model behaves in a normal scenario without congestion. We set 70% destination to node 7 and rest to others, the input rate is 1 person every 10 second and boarding starts at 10 am. There is no congestion to cause block the entrance as in previous experiments. In Figure 4.14 below, we can see that the data assimilation framework can predict the occupancy in node 7 with good accuracy. As such we show that our model will work even if there is no congestion.

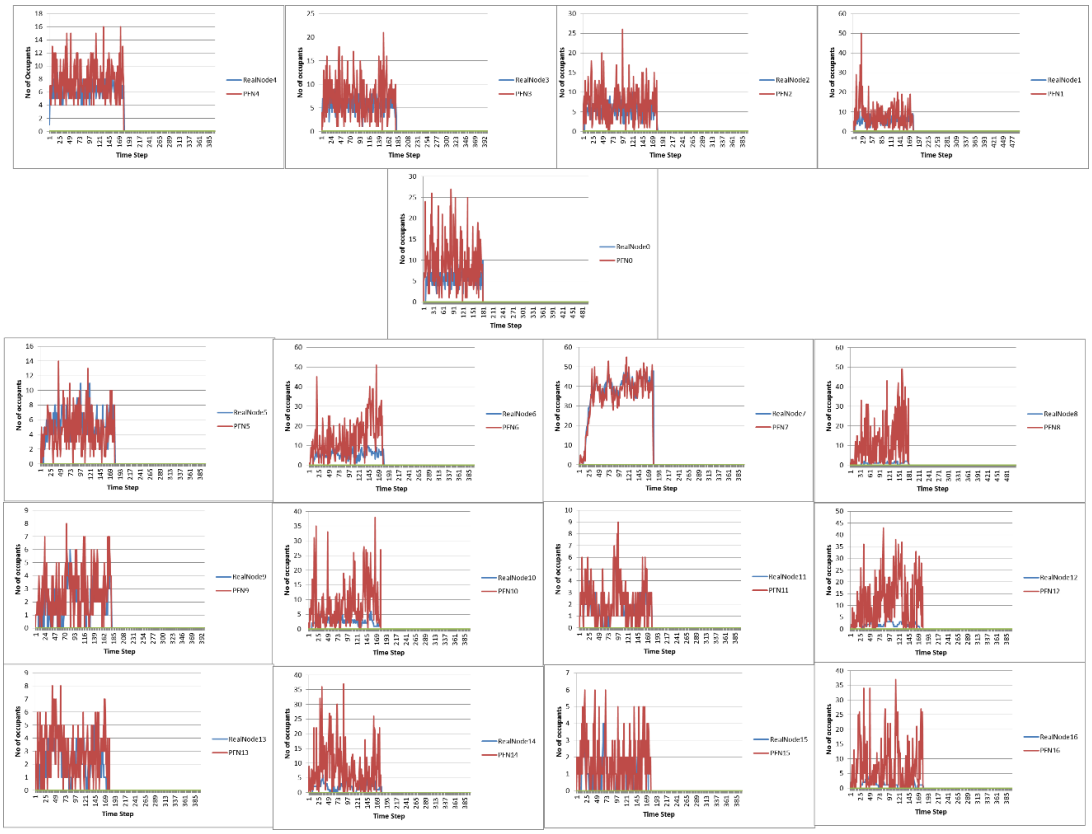


Figure 4.14 Comparing results for seventh scenario

4.4.3 Results for larger layout

This experiment analyzes the accuracy of the data assimilation model. We perform experiment for a simulation time of 8 hours for 8000 agents where two nodes are the sources from where the occupants enter the structure. The capacity of each room is 120 and the width of the door is 25m. The input rate is 4 occupants per 10 secs for the first 4 hours and afterwards 3 occupants per 10 secs. 50% of the occupants have node 113 and node 195 as their primary destinations (25% each) and rest agents will have randomly other nodes as their destinations. The occupants after reaching their destination will have a boarding time which ranges from 0 to 40 minutes. Since the input rate is high during the first 4 hours, it creates congestion in the nodes 113 and 195 which propagates to their neighboring nodes. We run data assimilation with 300 particles

and sample every two minutes, i.e. 120 time steps. For this experiment the sensors are distributed in every other node, so we will be using about half of the sensors compared with the total nodes. We compare the results in every node and analyze the output of the data assimilation with the real data. Since the number of nodes are high, we present output for only the nodes concerned with our experiment (main destination and intersections). In this experiment, we use sensor noise with standard deviation of 3.

We run simulations for 10 runs and average the results. In Figure 4.15 we can see the comparison of the number of occupants in the nodes with the simulation time. We observe that the output of the model follows the pattern as the real data.

Table 4.3 RMSE for using sensors in (a)half (b)1/3rd (c)1/5th of the nodes

	RMSE	Node ID
Min	2.27	193
Max	27.93	3
Average		7.478862
Dest 1	13.67	113
Dest 2	15.68	195

	RMSE	Node ID
Min	2.77	72
Max	23.75	3
Average		7.67
Dest 1	14.39	113
Dest 2	17.96	195

	RMSE	Node ID
Min	2.63	193
Max	22.48	114
Average		7.489
Dest 1	14.7	113
Dest 2	18.62	195

In Table 4.3(a) we can find the average RMS of the nodes. We see that the minimum RMSE is 2.27 and maximum is 27.93, also for the main destination 113 is 13.67 and for 195 is 15.68. Considering a large number of agents, the RMSE of these values are acceptable for prediction.

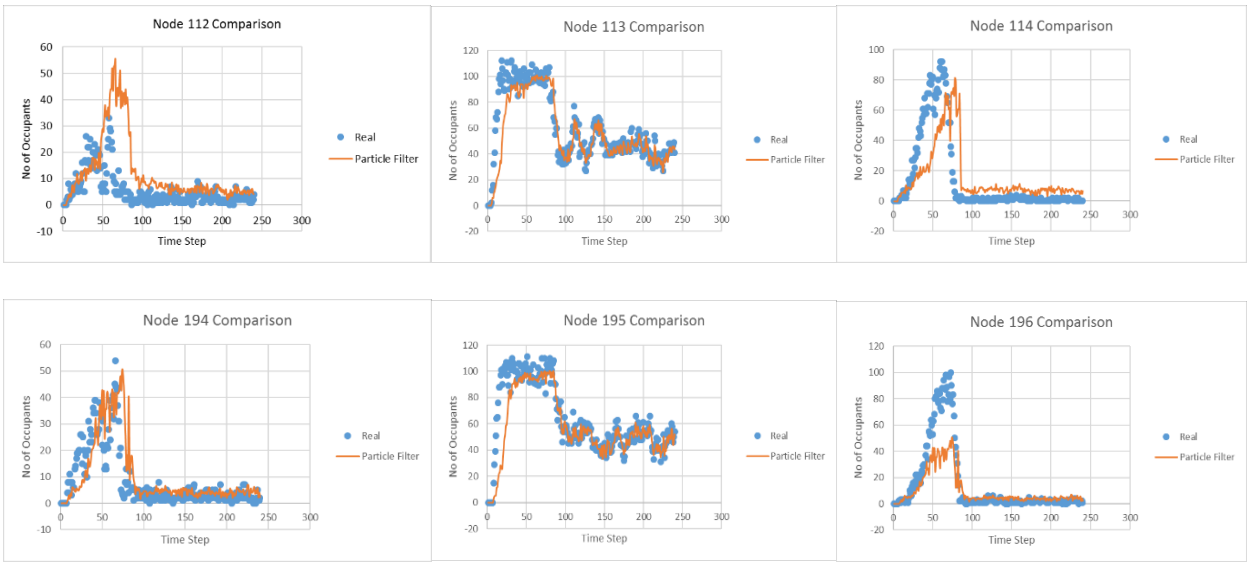


Figure 4.15 Data assimilation results in larger layout using sensors in half of the total nodes

Figure 4.15 shows the distribution of occupants in the main destination nodes and their neighbors across time. Blue circular dots represent the number of occupants in a real simulation, and orange color represents the average number of occupants from the particle filter simulation. The main nodes consist of the sensor, so the distribution of the occupants is almost accurate with the real simulation. However, both the neighbors do not contain sensors, yet we can see that the distribution of the occupants follows the same pattern as the real simulation. X-axis represents the simulation time where 1 time steps are 120 seconds, and Y axis represents the number of occupants.

In the next experiment, we run the simulation same as experiment 1 but we reduce the number of sensors to 1/3rd of the nodes and also, we generate the sensors with some randomness. We fix sensors in the main destinations, then for other nodes we randomly generate the sensors. To maintain the distribution of sensors, we group the nodes so that the neighboring nodes are in the same group. Then for each group, we generate sensors randomly. This will remove the

possibility of randomly generating sensors in only one area of the layout. The reason for keeping a sensor in the main destination can be justified with the reason that we are trying to estimate the occupancy distribution in all the nodes and not only the destination. We know that the congestion will be caused by having most of the occupants going to selected destination, but we want to estimate how it affects other nodes. As such it becomes important to correctly identify the occupancy in the main destination first. Our graph based agent oriented model can identify the distribution in other nodes using the particle filter even if there are no sensors there. However, as the number of sensors reduces while the number of nodes increases, the degree of uncertainty increases and error in prediction increases. From Table 4.3(b) we find that although we reduced the sensors we have obtained acceptable the RMSE for the rooms. Also in Figure 4.16, we can find the similar patterns of the average output of the occupancy distribution compared with the real occupancy in the nodes of interest.

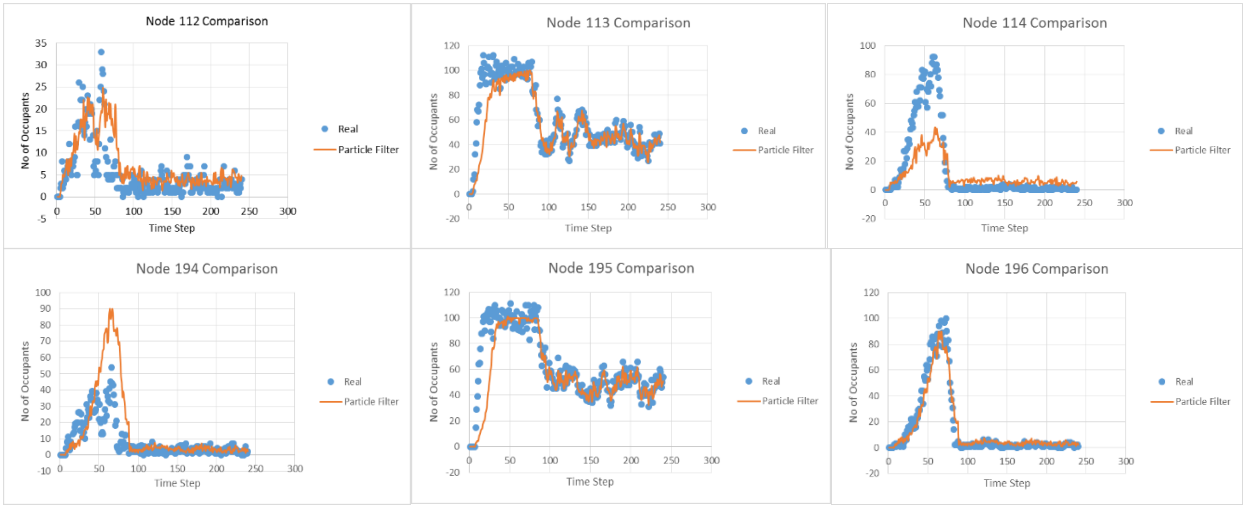


Figure 4.16 Data assimilation results in larger layout using sensors in 1/3rd of the total nodes

Next, we reduce the number of sensors to about 1/5th of the total number of nodes. Like the previous experiment, we randomly select sensors in each of the group. From the graph in Figure 4.17, we observe that the output of the particle filter can model pattern of the occupancy distribution in the real system. The result of the RMS as shown in Table 4.3(c) is also in acceptable range.

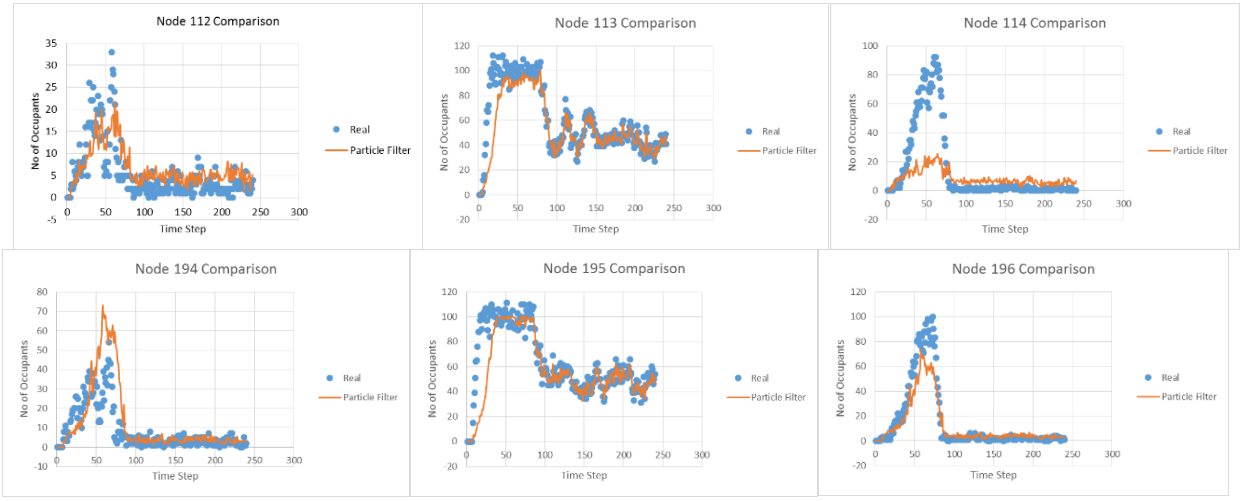


Figure 4.17 Data assimilation results in larger layout using sensors in 1/5th of the total nodes

In the next set of experiments, we analyze the result of the data assimilation model for estimating occupancy dynamics in a building with a large number of nodes and occupants. We perform an experiment for a simulation time of 8 hours for about 5500 agents where two nodes are the sources from where the occupants enter the structure. The capacity of each room is 100, and the width of the door is 25m. In the experiment, we create a scenario in which the occupants have node 113 and node 195 as their primary destination in the first half. In the second half, the occupants have node 113 and node 151 as their primary destination. Thus, we can observe two congestions in node 113, and one congestion during first and second half on node 195 and 151

respectively. We create the congestion manually for some hours, after which the occupants will have a boarding time which ranges from 0 to 25 minutes. We run data assimilation with 100 particles and sample every two minutes, i.e. 120 time steps. We gradually decrease the number of sensors from half to one-third and one-fourth and present our results. We observe that we can get good results even when we decrease the sensors. Thus, our model can get good results with limited sensors as well. We compare the results in every node and analyze the output of the data assimilation with the real data as graphs. In the figure, we show the result for the main destination and their neighbors to observe the estimation for the congestion. We also show the root mean square error for each of the simulations. The root mean square is calculated as average root mean square for all of the nodes in each time step. We run simulations for five runs and average the results. In all our results, we can estimate the congestion in the nodes. In node 152, we observe that there is no congestion in real simulation but our model shows there is congestion, it is due to the location of the node. In the real model, we use the shortest path algorithm, Dijkstra's algorithm from the graph based agent oriented model to generate the path. As such in the case of node 151, due to the specific location, the occupants move only from the direction of one neighbor, node 150 and not from 152 since it is the shortest path from both sources. In all other nodes, the occupants move from both the neighbors since the shortest path consists of each neighbor for each of the two sources. As such, in the data assimilation, the model estimates that the congestion in a node has a cause/effect to the neighboring nodes. In this work, we do not consider the errors in such nodes at a specific location and leave it for future work.

We use sensors in half of the nodes manually and place the sensors in the destination nodes, but there are no sensors in the neighboring nodes. We use 100 particles for data assimilation to detect the congestion. We observe that we are able to detect the patterns of congestion in all the

nodes. In Figure 4.18 below, the blue graph represents the real simulation, and orange graph represents the data assimilation estimation. X-axis represents the time step of the simulation and Y-axis represents the actual number of occupants in the node. As in the real simulation, data assimilation shows that in node 113 and it's neighboring nodes there are two stages of congestion, while in node 195, 151 and its neighboring nodes there are only one stage of congestion. The congestion in node 152 is an estimation error due to the model which has been described previously.

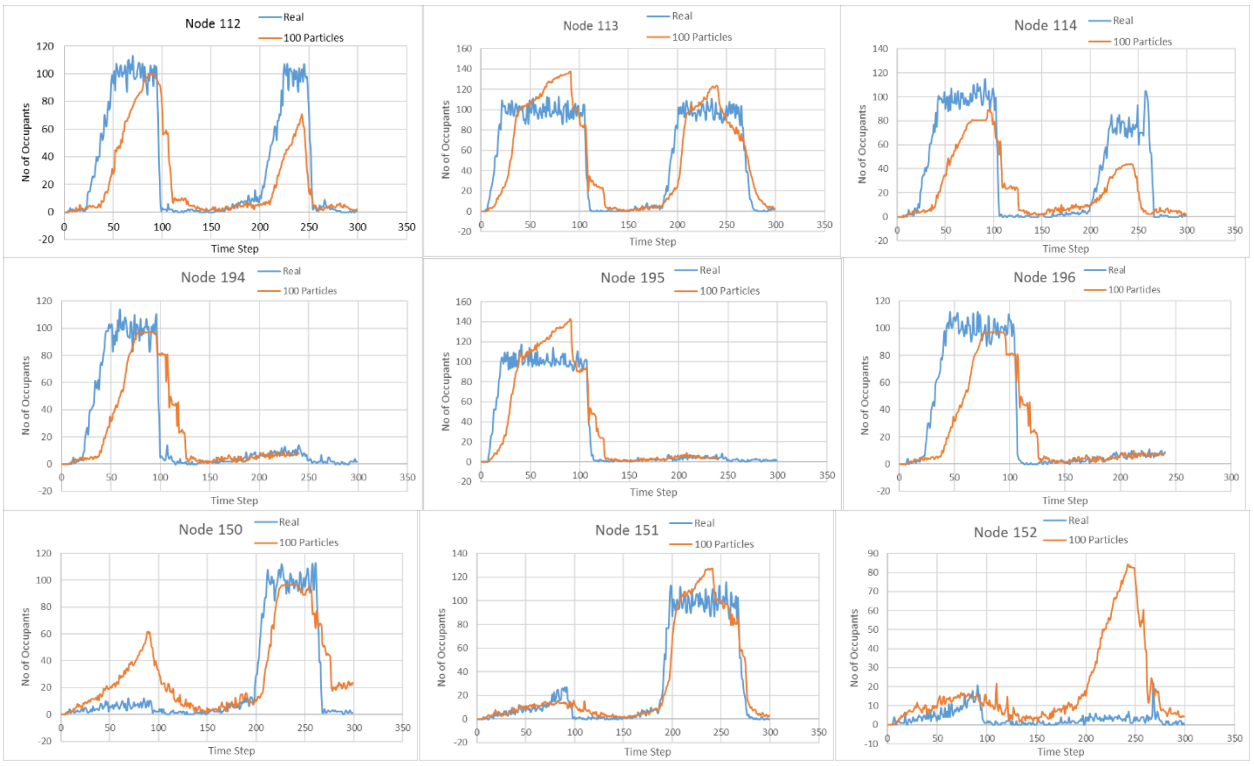
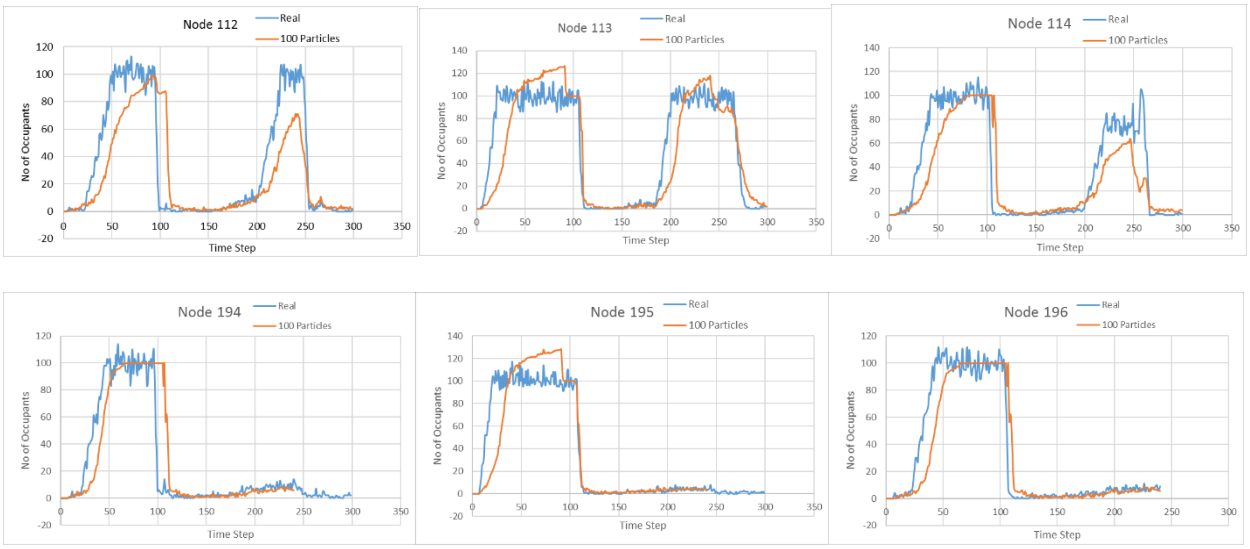


Figure 4.18 Data assimilation results using sensors in half of the nodes

For the next experiment, we run the simulation same as the previous scenario, but we reduce the number of sensors to one-third of the nodes, and we generate the sensors with some randomness. We fix sensors in the main destinations, then for other nodes we randomly generate the sensors. To maintain the distribution of sensors across the building, we group the nodes so that

the neighboring nodes lie in the same group. Then for each group, we generate sensors randomly. This will remove the possibility of randomly generating sensors in only one area of the layout. The reason for keeping a sensor in the main destination can be justified with the reason that we are trying to estimate the occupancy distribution in all the nodes and not only the destination. We know that the congestion will be caused by having most of the occupants going to selected destination, but we want to estimate how it affects other nodes. As such it becomes important to identify the occupancy in the main destination first. Our graph based agent oriented model can identify the distribution in other nodes using the particle filter even if there are no sensors there. However, as the number of sensors reduces while the number of nodes increase, the degree of uncertainty increases and error in prediction increases. In our model, we can maintain the error in prediction low even by decreasing the sensors as shown in our results. In Figure 4.19, we can find the similar patterns of the average output of the occupancy distribution compared with the real occupancy in the nodes of interest and observe that the results are comparable with the simulation using half of the sensors. We further try to reduce the sensors and observe if we can maintain the estimation error.



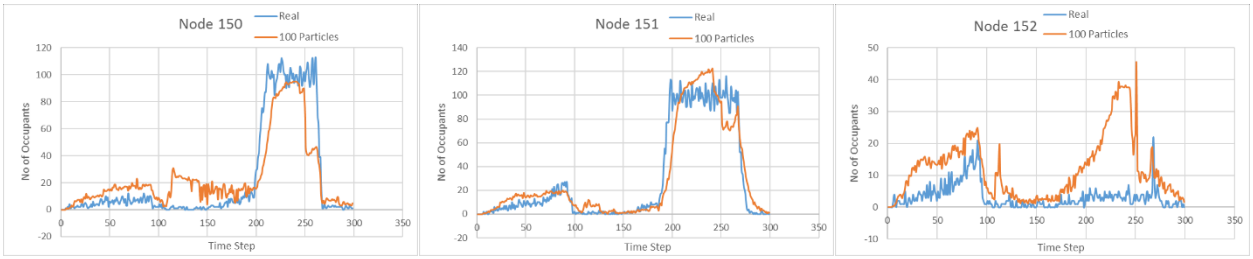


Figure 4.19 Data assimilation results using sensors in one third of the nodes

Next, we reduce the number of sensors to about one fifth of the total number of nodes. Like the previous experiment, we randomly select sensors in each of the group. From the average results, we observe that the output of the particle filter is able to model the pattern of the occupancy distribution similar to the real occupancy distribution. Hence, we can use sensors in just about one fourth of the total nodes to estimate the occupancy dynamics using our model and achieve a good estimation. The results are shown in Figure 4.20 below.

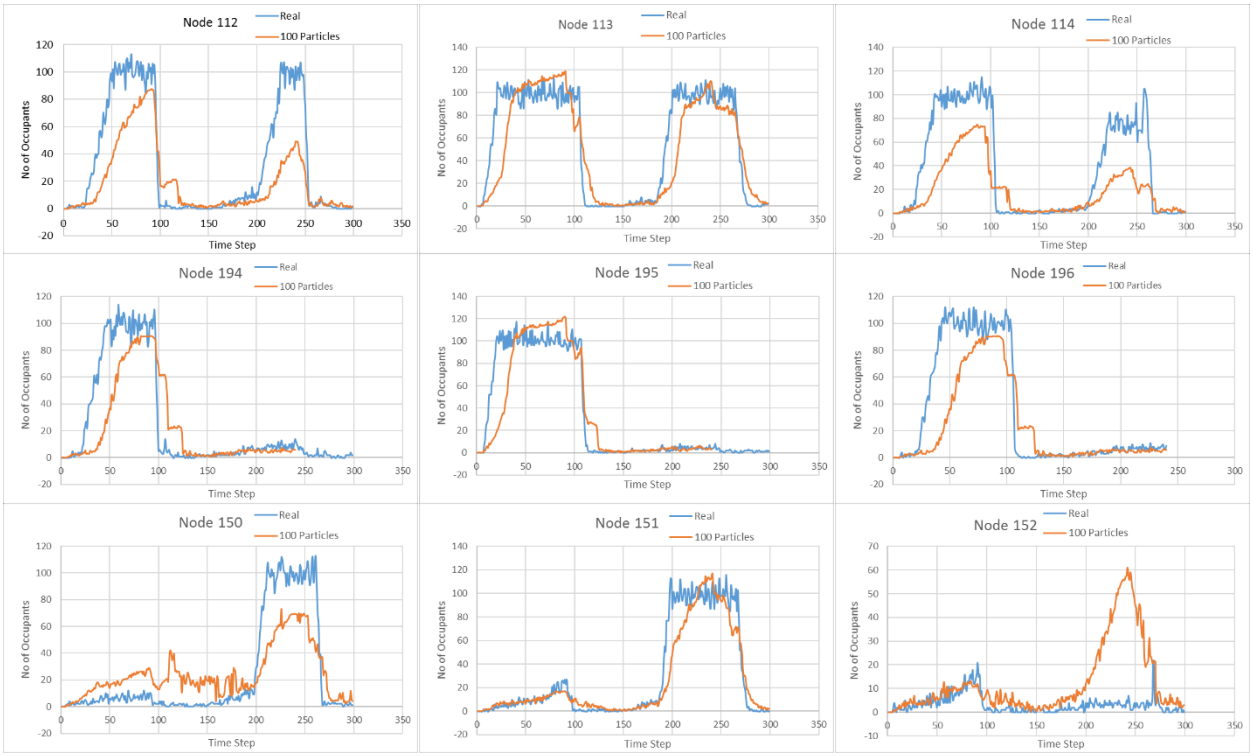


Figure 4.20 Data assimilation results using sensors in one fifth of the nodes

We compute the average root mean square error (RMSE) for each simulation by computing the averages of RMSE for each node at each time step. In Figure 4.21 below, we can observe that for all runs, RMSE increases and then decreases and again increases. It corresponds to the scenario of the experiment with two congestion stages when the number of occupants increase, thus increasing error. In Figure 4.21 the blue, orange and gray graph represents the result of using sensors in half, one-third and one fourth of the nodes respectively. We observe that the average RMSE remains almost the same even when decreasing the sensors. We also observe that the average RMSE is slightly less for using sensors in half of the nodes than using in one-third which is less than using in one-fourth of the nodes. This shows that error increases as we reduce the number of sensors and would be much evident if we reduce the number of sensors drastically.

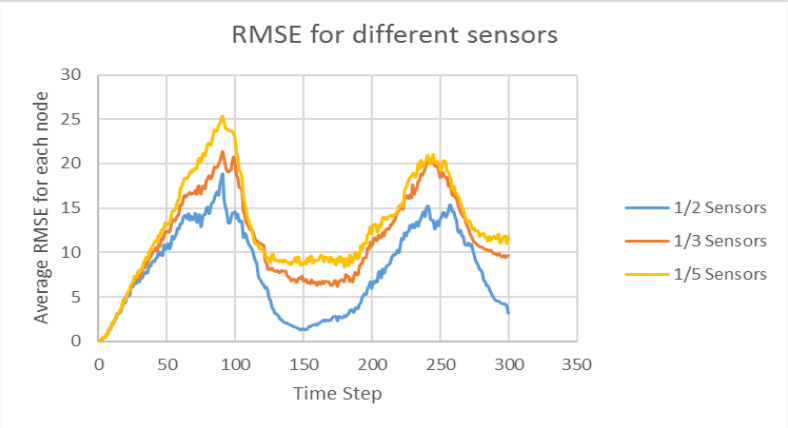


Figure 4.21 RMSE comparison using different number of sensors

4.4.4 Comparison of resampling using sensor data particles

In this section, we conduct an experiment using particles from sensor data and compare the results with the standard resampling. As the number of agents increase, it becomes difficult for the standard particles to converge with the true system state, and hence the prediction accuracy of the particle filter decreases. For particle filters, it becomes difficult to converge in the case of events

which are rapid in nature. Events like congestion which occurs in short periods of time are difficult to detect due to the lack of time to converge and complexity of the system. We would require really large number of particles to properly detect occupancy in such scenarios. However, using the particles from sensor data, we can maintain the accuracy since the sensor data obtained can help the particles not diverge from the real state by a large margin.

To verify it, we increase the capacity of the room to 200, the length of the edge to 35m and the number of occupants to about 16,000 so that the complexity of the model is increased and it becomes difficult for the general data assimilation to detect the occupancy behavior. The task of estimating an exact number of occupants is highly difficult at such complex system state hence we focus on detecting congestion i.e. when there are very high number of occupants in a node. For this, we can define a threshold value which defines the minimum number of occupants to create a congestion situation. When a node contains more than that number of occupants, it is in a congestion state. Using the identical twin experiment, we create a real system with congestion in two nodes (Node 113 and 195) for some period and observe the congestion which propagates across the neighboring nodes. We set up sensors in about half of the nodes and run both data assimilation filters.

Figure 4.22 shows the result of comparison where blue graph represents the occupancy from a real system, yellow from the standard resampling and red from the resampling using sensor data particles. For convenience, we compare the results of the main destination nodes and their neighboring nodes. We observe that the standard particle filter is not able to detect congestion in time compared with the particle filter using sensor data. We create a $1/5^{th}$ of the total standard particles as a set of particles using sensor data and randomly select the sub-states from both sets of particles. We see that in the node the neighboring nodes the standard resampling is slow to

detect congestion. However the resampling using particles from sensor data can predict congestion (occupants more than half the capacity) at the same time as the real system.

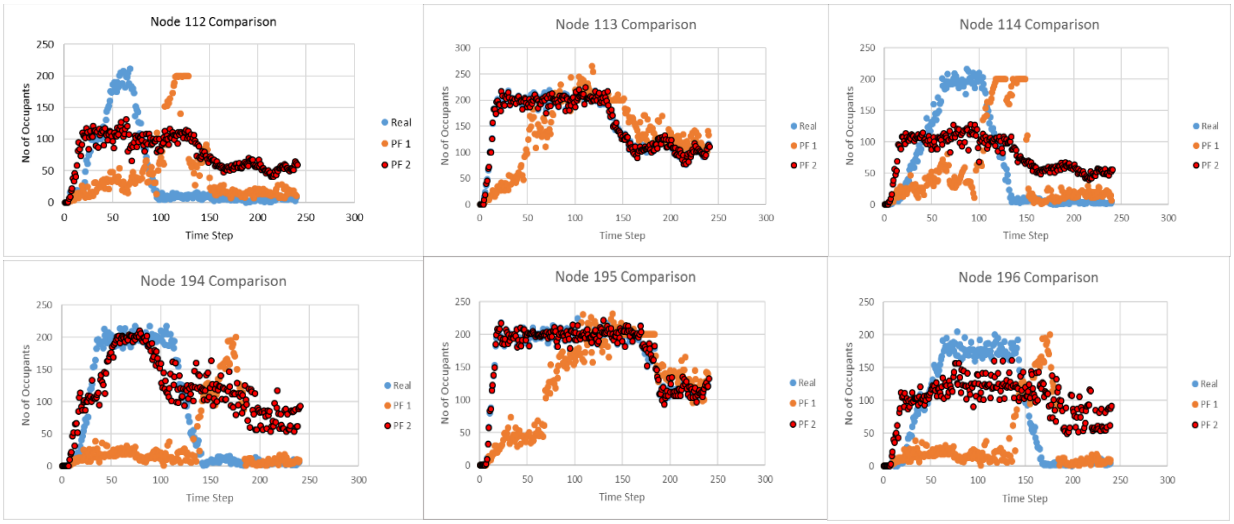


Figure 4.22 Comparison of standard resampling with resampling using sensor data particles with 200 particles

In the second experiment, we increase the size of the room to 400, edge size to 35 and use about 9500 agents for a congestion simulation. Congestion are created in nodes 113 and 195 in two stages and we use only 100 particles to detect the occupancy. Since the number of particles is quite few, the occupancy is difficult to estimate using the standard bootstrap particle filter. However, using our new algorithm we are able to estimate the occupancy pattern for the congestion with the limited number of particles. Figure 4.23 shows the result of the comparison, here blue graph is the occupancy from real simulation, orange graph is the simulation using the standard particle filter and red graph is the result of data assimilation using direct sensor particles. We observe the result of data assimilation using direct sensor data is much better than using the standard particle filter. The standard particle filter is able to make some estimation for the main

destination nodes but is not able to estimate the occupancy for the neighboring nodes. However, the data assimilation using direct sensor data is able to make good estimation for neighboring nodes as well. As such, we see that the new algorithm is able to make good predictions utilizing the direct sensor information and use few particles compared with the standard particle filter.

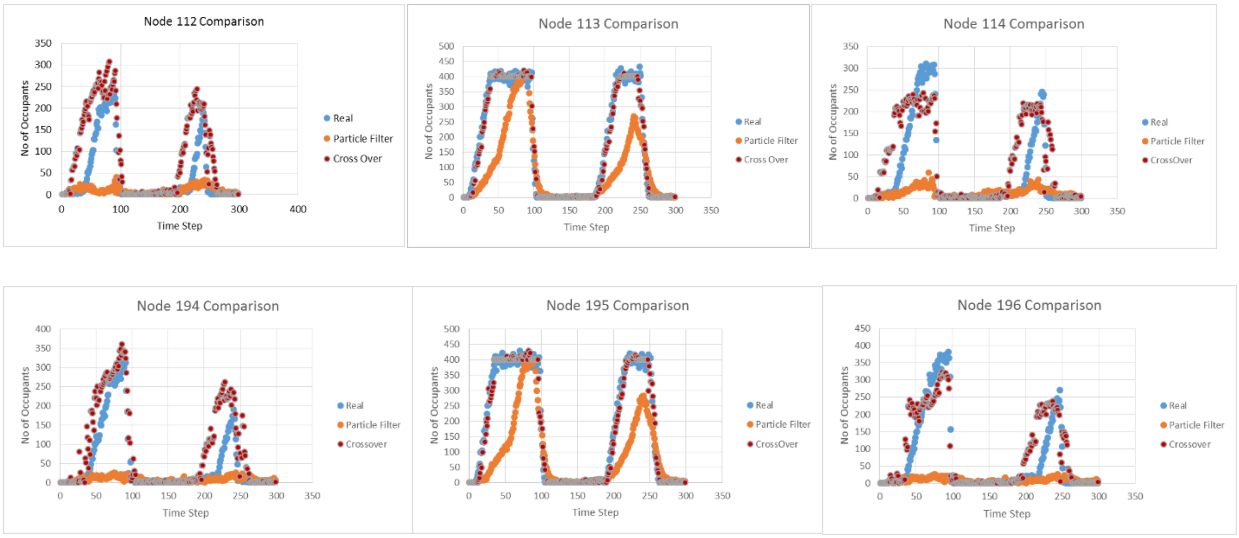


Figure 4.23 Comparison of standard resampling with resampling using sensor data with 100 particles

5. TOWARDS ACTIVITY INFORMED DDDS FRAMEWORK

5.1 Case Study: A Smart Office Environment

We consider a smart office environment and plan to estimate the system state (e.g., density and locations of people in the environment) from sensor data to support emergency response planning for scenarios such as fire alarm and evacuation. To study the sensor data, we set up a smart office environment at the Department of Computer Science at Georgia State University on the 14th floor. It consisted of 20 TelosW static wireless sensor nodes (Figure 5.1(a)) deployed throughout the 30 meter × 30 meter floor workplace environment. The sensors are fixed on the ceiling and are triggered when a person walks under it, but it does not detect a person sitting motionless under the sensor. The sensor nodes are deployed mainly in the hallways, key positions like entry, exit, positions with high motion activities in workday like a conference room, printer room, kitchen, busy lab. The nodes are equipped with Panasonic AMN-31111 PIR (passive infrared) motion sensor. The detected motion data are sampled at 10 Hz when an event triggered by motion. The base station collects the data through multi-hop communication (formed a 5-hop network) and stored in a back-end database.

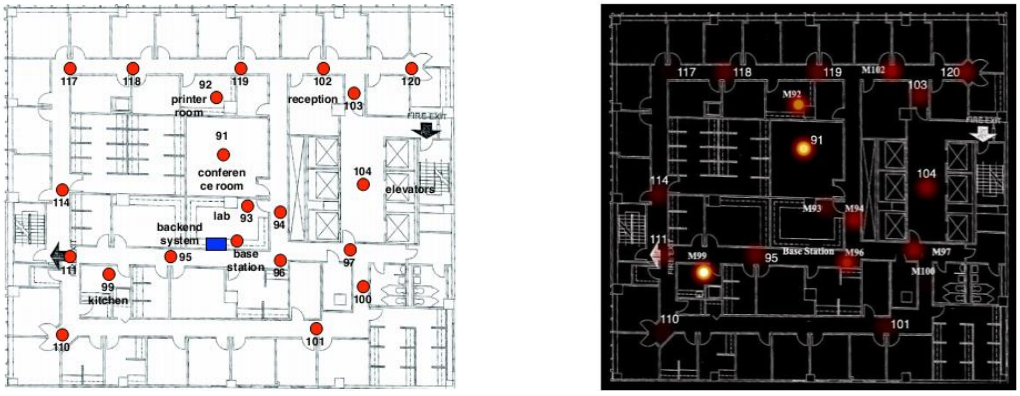


Figure 5.1 (a) Smart Office with sensors (b)Heat map based on sensor count in a week.

The data can be visualized in the form of a two-dimensional matrix, where the columns denote sensors and the rows denote the time intervals. The time interval of the row is 100ms and a duration of 2 seconds is typically observed for detecting a person crossing under a sensor. Figure 5.1(b) uses a heat map showing the sensor counts counting the activities on each sensor, based on data collected for a consecutive week from morning 8 am to evening 20 pm. The data for each node is counted, which provides the activity occurring at that node. In the heat map, brighter colors indicate higher counts of occupancy. From the figure, clearly, node M99 (Kitchen) and 91(Conference) have the highest count followed by node M92 (Printer room).

By looking at the data from the sensor network, we can identify regions with various densities of activities. For example, a region where a meeting is going to take place will record a high number of activities for that duration. The hallways leading to that region of high activity will also record a high-density corresponding for the entering and exit of the people. When the meeting is over, the people will leave the meeting room and may go to a cafe or their respective rooms. This behavior pattern will lead to reduced activity in the conference room and increased activity in the cafe and other office rooms. Hence, we will have activities depending on time and space. By understanding this spatial and temporal relationships, we can develop a model to represent human behaviors in space and time, and then recognize the “current” behavior patterns from real-time sensor data.

5.2 Activity informed DDDS framework

In this section, we discuss a new Data Driven Dynamic simulation framework for smart environment known as activity-informed DDDS framework. A dynamic data driven simulation (DDDS) is a model where the real-time data streams continually influence a simulation system for better analysis and prediction of a system under study [41]. A major task of DDDS is data

assimilation that assimilates sensor data collected from the real system into the simulation model. Typically, the real system's states, which change over time, cannot be directly observed and is unknown to the simulation model. This makes the simulation start from a state different from the state of the real system, leading to inaccurate simulation results. Thus, there is a need to dynamically estimate the "current" state of the real system and then feed the estimated states to the simulation model. This is achieved through data assimilation that assimilates real-time sensor data for inference of the "current" system state. In this work, we present a framework that adds a new layer of behavior pattern recognition from sensor activities on the top of data assimilation and uses the recognized behavior pattern to inform the simulation model. Figure 5.2 shows this activity-informed DDDS framework. As it can be seen, at the bottom layer is data assimilation of the simulation model. The data assimilation uses real-time sensor data and the simulation model to infer the state of the system and to tune the model parameters in real-time.

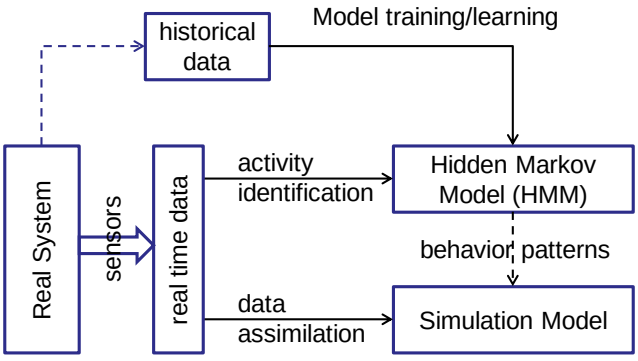


Figure 5.2 Activity-informed DDDS framework.

On top of data assimilation is the activity identification and behavior pattern recognition layer. In this work, we propose to use Hidden Markov Model (HMM) [42] and extensions of HMM for the task of behavior pattern recognition from spatial-temporal sensor activities. HMM is defined as quintuple (*S, E, P, A, B*), where $S = \{S_1 \ldots S_N\}$ are the values for the hidden states, E =

$\{O_1 \ldots O_T\}$ are the values for the observations, P is a probability distribution of the initial state, A is the transition probability matrix, and B is the emission probability matrix. HMM is used to recognize events/patterns in many different applications ([43] [44] [45]). In our framework, HMM supports behavior pattern recognition from real-time sensor data and outputs the recognized behavior patterns to the simulation model to inform the latter for a better simulation. Figure 5.2 also shows that the HMM needs to be trained from historical data before it can be used in real time for behavior pattern recognition.

This framework separates the two layers that have different concerns: the simulation model captures the low-level dynamics of the system behavior, and the HMM recognizes the high level "behavior pattern" to inform the simulation model. In the next section, we focus on the behavior pattern recognition layer and use a smart office environment as the application context to show how to recognize behavior patterns form spatial-temporal sensor data using HMM.

5.3 Agent based model for building occupancy

In this section, we present our agent based model designed for simulating a smart office environment. Agent based model simulates each occupant as an agent and thus allows us to observe emergent behaviors due to agents' interactions. Agents can be given properties as of an actual occupant and allowed to interact in an environment consisting of obstacles like walls, doors, and path like stairs, doors. Thus, the agent will behave based on their property in a given environment at various kind of situations and generate occupancy dynamics. The model runs in a stepwise fashion and models individual occupants' moving behavior of navigation and collision avoidance. In this work, we develop a relatively simple agent based simulation model and use it to model a small smart office and a relatively larger shopping mall building structure. Figure 5.3(a) shows a layout of a smart office with a conference room, break room, copier room and a reception office.

Blue lines represent the walls, and black points represent the way points described later. Figure 5.3(b) shows a layout of a shopping mall with the single intersection. The black dots are the agents who represent the occupants moving across the mall. There is only one intersection, and since everyone uses the same intersection, there is a crowd formation when many people pass through it. Using agent based model we can observe the formation of crowd behavior in the environment at a detailed level.

Each agent is modeled to have two basic behaviors which a general moving occupant display in the real world: get to destination and avoidance. The agents display behavior-based control mechanism, in which the agent may choose to select a behavior matching current world condition to conflict resolution when there are competing alternatives [36]. The first behavior allows the agent to navigate to a given destination by generating a preplanned route from the current location to the destination. The agent follows the path until they reach the destination, after which they may be made to go to the new destination or stay there. The avoidance behavior enables an agent to avoid collision or blockade with another agent or an obstacle. It does so by following three properties: the first it keeps a certain comfortable distance with its neighbor agents. If an agent finds another agents or obstacle near its comfortable radius, it will move to a comfortable distance away from it. Second, if an agent's path is blocked, it moves sideways to generate an open path. Third, the agent may generate a new route to the destination if its current route is blocked.

An agent can be formally defined as $\{l, v\}$ where l is the coordinate location of the agent and v is the velocity. An agent generates its route to the destination using a predefined waypoint graph in the environment. The graph is created in such a way that for any position in the environment floor map, a path can be generated to a point in the graph, known as a way point, without crossing any obstacles. The way points are set at every intersection and rooms so that each

location can generate the shortest path to it. Then the agent can follow the waypoint graph connections to reach any destination. Thus, waypoint graph allows the model to connect every coordinate of the environment. In Figure 5.3(a), a layout is shown for a smart office where black circles represent points and lines between them represent the connectivity among the way points. An agent will compute the shortest path between its source and destination through the waypoint graph. The route of the agent consists of series of way points, and an agents' moving direction is determined based on its current location and the next route point. Formally, the waypoint graph can be defined as a vertex edge graph $G = \{V, E, D\}$ where V, E, D are the sets of vertices, edges, and distance of edges respectively.

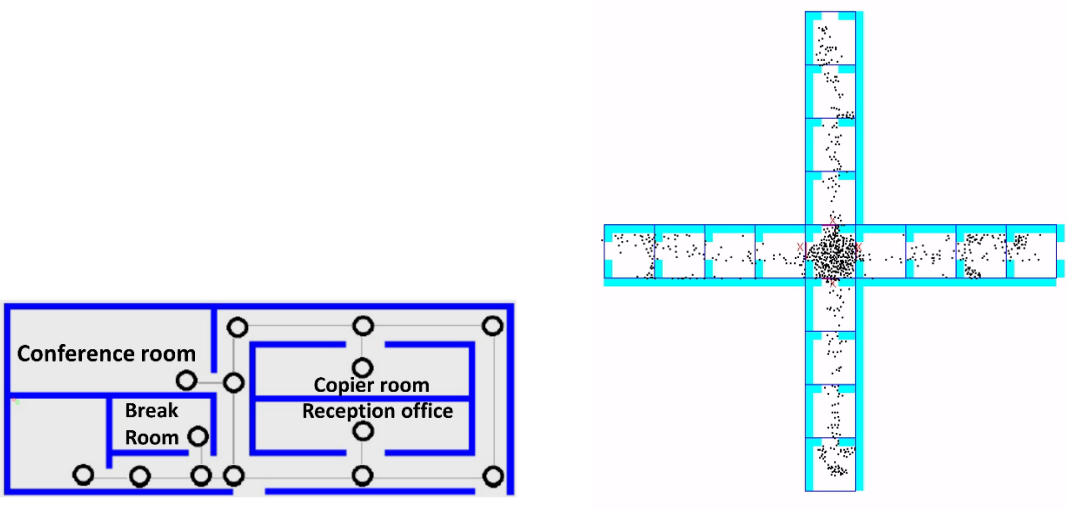


Figure 5.3 (a) Office layout with waypoint graph (b) A shopping mall layout with agents

The environment consists of a sensor which is modeled as a binary passive infrared sensor. The sensor can detect the number of people moving through its range, which is a circle of predefined radius. It is a basic sensor and can only detect if there is any motion across its range. The output is binary, 1 for a detection and 0 for no detection in a sample step. Thus, it is unable to identify if multiple people pass through its range since it will be 1 for any detection. The sensors

are assumed to be mounted at ceilings and placed at places where people mostly move, like intersections, entrances and exits of the room. The reason for choosing binary sensors is their cheap cost and non-invasive nature in areas like an office.

Using the model, we can simulate smart environments with few occupants like an office, home or hospitals. The model allows occupants to move across the rooms thus simulating various activities that occur in those environments. The sensors collect the real-time information from the environment which can be used in data assimilation framework for dynamic estimation of occupancy.

5.4 Behavior patterns of occupants in a building

We apply behavior pattern detection to a smart office case study example. The case study example, in this section, focuses only on the behavior pattern detection using Hidden Markov Models (HMM) [51]. The HMM is a statistical Markov model in which the hidden system states are predicted based on the visible outputs. The advantage of HMM is that it can learn the parameters from the historical data and use it for state estimation in future. In this case study, we create a smart office environment using the agent-based model [52]. With the agent-based simulation model, we can easily create some occupants and create different scenarios. We are interested in a scenario of a conference event when agents move inside the conference room to start the conference and then move outside the conference room when it ends. The smart office environment is deployed with simple binary sensors that report 1 if an occupant enters its sensing area and reports 0 otherwise. The binary sensor provides anonymous position information and cannot identify multiple occupants in its sensing area. The data collected by the sensor contains errors and is subject to environment clutter.

In this work, we focus on correctly identifying the behavior patterns in real time. Sensor data are collected from the binary sensors and HMM is used to identify the behavior pattern states during a conference. The binary motion sensors are placed at the doors to capture the motion and are triggered only when users pass through their range. To use the HMM model, first we need to train the model to recognize the observations. For that we learn the HMM parameters from the historical data. We train the HMM for several scenarios (with data generated from simulations) of a general conference and learn the HMM parameters. We then use the trained HMM to estimate the behavior patterns in new scenarios.

To create training dataset, we create all the possible scenarios concerning the conference room where the users attend the conference. For a system consisting of an environment like a conference room, the behavior pattern will be behaviors like "*entering the conference room", "leaving the room", "attending the conference".* The observational data are the real-time data from the binary motion sensors. The behavior pattern is the states of the real system which are reflected by the sensor observations. For our experiment, we consider six different behavior patterns (a state represents each in HMM): *outside, inconference, few_entering, high_entering, few_leaving and high_leaving.* The state *outside* represents the behavior pattern when there is no conference so that all the occupants are outside the conference room, the state *inconference* represents the behavior pattern when the occupants are inside the conference room for attending conference. The state *few_entering* represents the behavior pattern for a small number of occupants entering the conference room and *high_entering* represents a large number of occupants entering the conference room. Similarly, *low_leaving* and *high_leaving* represents the behavior patterns when a small and large numbers of occupants leave the conference room, respectively.

It is a challenging task to extract high-level information like behavior patterns directly from binary sensor data. So, as a pre-processing, we check the triggered rate of the sensors for a fixed sample period. For the experiment, we used a sample of 15 time steps and based on the amount of sensor triggered rate during that time sample, assign sensor data for each sample according to three values: zero sensor count, low sensor count, and high sensor count. Zero sensor count is the result of either when there is no conference so no occupants are entering/leaving the room or when the conference is happening so all users are seated inside the room. Low sensor count is the result of a small number of occupants (about one or two) entering or leaving the room. High sensor count is the result of either at the beginning or ending of the conference as the majority occupants enter or leave the conference room at the same time. At the beginning and end of a conference, most the occupants enter and leave the room, but during the conference, they do not move so no sensor is triggered.

Table 5.1 Initial probability for HMM

States	Initial Probability
outside	0.99
few_entering	0.002
high_entering	0.002
inconference	0.002
few_leaving	0.002
high_leaving	0.002

Table 5.2 Emission probability for HMM

Behavior States	Emission probability		
	O(Zero)	***O(Low)***	***O(High)***
outside	0.90	0.05	0.05
few_entering	0	1	0
high_entering	0.024	0.976	0
inconference	0.90	0.05	0.05
few_leaving	0	0.053	0.947
high_leaving	0	0.053	0.947

We select some training data from historical sensor data and use the Baum-Welch algorithm to learn the HMM parameters [53]. Then in real time, we used the learned HMM to predict the maximum likelihood of each state based on the real-time sensor data. The state with the maximum probability estimates the behavior of the system. We can estimate the probability of each state to represent the possibility of each behavior pattern at the current time based on the observation. We utilize this information to predict a behavior pattern of the real system. Table 5.1 shows the initial state probability for each of the states and Table 5.2 shows the emission probability used for HMM computation. Figure 5.4 shows the learned state graph and shows the transition probability between the states. With the learned HMM, we then use it to predict the behavior pattern of new scenarios. Several scenarios have been used to test the HMM, including one when all occupants attend a conference and leave at the end, and one when a few occupants

enter the conference room to check if there is a conference and leave the room as the room is empty.

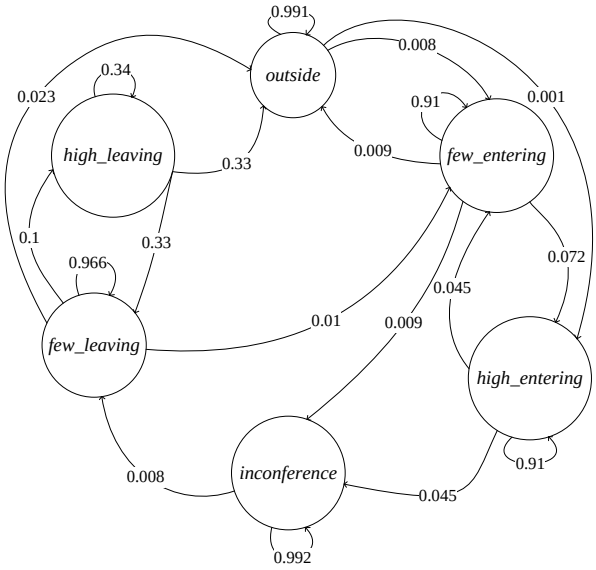


Figure 5.4 Behavior pattern states transition probability for HMM

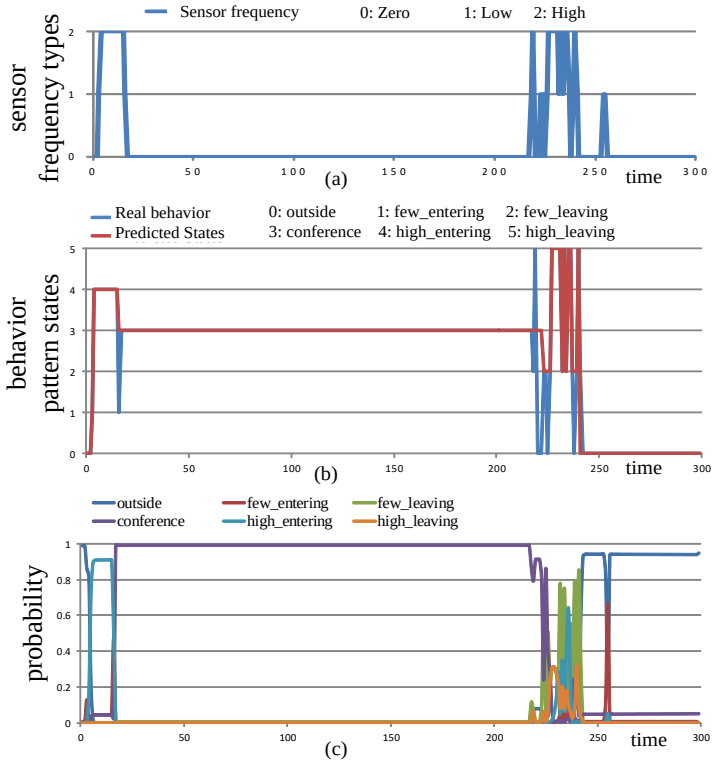


Figure 5.5 a) Sensor frequency data (b) Comparing the real and predicted behavior (c) Normalized probability for the behavior pattern in real time

We can compute the maximum likelihood probability for each time step and predict the current behavior pattern as shown in Figure 5.5(b). In the figure, the blue line represents the real behavior of the state, and the red line represents the estimated behavior of the system using HMM. The estimation is done in the real time when the sensor data is received. We can see that we could correctly estimate the behavior of the conference scenario and distinguish the conference state from the outside state, even though both of them give the observation of zero sensor count. Figure 5.5(c) shows the normalized relative probability for the different states in real time. As can be seen, the probabilities for the conference state and the outside state dominate during the conference time and the outside time, respectively.

The experiment aims at evaluating the possibility of detecting behavior patterns from the sensor data, and the evaluation of the accuracy is calculated as:

$$\frac{\sum_{k=1}^{T} S_t^k - S_t^{real}}{T} \qquad (5.1)$$

where, T is the total simulation steps and S is the behavior pattern state.

Table 5.3 shows the average accuracy for recognizing the behaviors from the observed sensor data using HMM. We assume that the behavior states always start from all users outside the conference room. From the results, we see that we have a good accuracy for recognizing the behavior patterns from the noisy sensor data in the real time. The accuracy for behavior pattern *few_entering* is low because the occurrence of that behavior is very low compared to other behavior patterns.

Table 5.3 Average accuracy for behaviors

Behavior State	**Average Accuracy**
outside	85.53
inconference	90.44
few_entering	18.40
high_leaving	75.01
few_leaving	63.35
high_leaving	71.07

For this example, the behavior pattern when there is a conference or no conference is successfully recognized with high accuracy. The information of relative probability as shown in Figure 5.5(c) can be used by the data assimilation component for improving state estimation as discussed in the behavior pattern informed data assimilation framework.

We recognized the behavior patterns of a system from sensor data in real time that can provide useful information to improve particle filter-based data assimilation. A framework of behavior pattern informed data assimilation has been presented and a smart office case study example is shown as how the behavior pattern detection works.

5.5 Behavior Pattern Recognition using Coupled HMM

Within the application context of the smart office environment, this section uses a simple example to show how to recognize behavior pattern from spatial-temporal sensor data using Coupled HMM (CHMM) [46]. We consider a simplified smart office environment that has a conference room and a cafeteria room connected by a hallway, as shown in Figure 5.6. People attend the meeting in the conference room and go to the cafeteria during breaks. Suppose the three places have binary sensors that capture the motions of the people. We can say that during the

breaks, there will be more counts in the cafeteria and increased counts across the hallway during start and end of the break. At other times, more counts will be in the conference room. Also at each location, the sensor activities will depend on various events like the start of the conference, short breaks, change of speakers, end of conference and others.

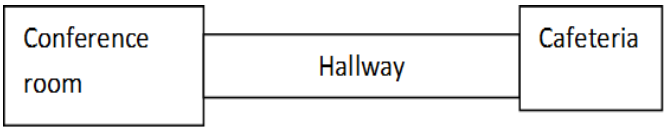


Figure 5.6 A simple example of a smart office environment consisting of a conference room and cafeteria connected by a hallway.

Since the activity consists of people moving across the rooms at various time intervals, we can assume that each room exhibits the Markov property in the temporal domain. Then we can create a single HMM for each room based on the observation of the sensor data. Observing each spatial location separately, a localized model can be achieved by creating a single HMM for each room separately. Figure 5.7(a) illustrates an HMM for the conference room.

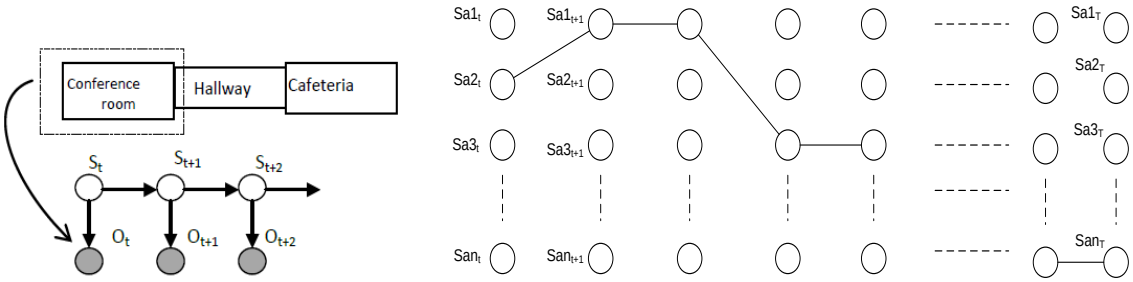


Figure 5.7 (a) HMM for each location (b) A single path through an n-state HMM

In Figure 5.7(a), the circles represent the states. The shaded circles are the observed states and others are hidden one. The hidden states correspond to the activity of each location. Figure 5.7(b) shows a detail representation of a conventional HMM for a location with n states. The straight line shows the transitional probability and represents the hidden states selected for each

time step. For example, the states of a conference room may be *begin_conference*, *having_conference*, *short_break*, *end_conference*, etc. Similarly, the cafeteria may have states like *empty*, *break_time*, etc. These states will differ according to the time period and will also have relation with the neighboring locations. Using the HMM for each spatial location, we can capture the temporal behavior pattern relationships within each location. Incorporating the temporal information will enable the detection of behavior pattern with varying temporal transitions.

As discussed before, there is also a dependency between different places. When a person moves from one place to another, the neighboring places of the source exhibit certain activity pattern. For example, in Figure 5.6, during the break when people go to cafeteria from conference room, they go through the hallway. Thus, the flow of motion will show a certain pattern from the conference room to the hallway and then to the cafeteria. This means the sensor activity will have a strong correlation within the spatially neighboring regions. The sensor activity in a place will have relationships with the activities in close proximity regions. We can model this strong relationship between spatially neighboring regions to account for the activity relationship over space. In this paper, we choose to use coupled Hidden Markov Model to capture this correlation behavior of the neighboring regions.

Figure 5.8(a) shows a coupled Hidden Markov Model (CHMM) that captures the relationship between the activity occurring in the conference room, hallway and cafeteria. In the figure, S_a, S_b, and S_c represent the states of the conference room, hallway, and cafeteria respectively. The solid straight lines represent the transition probabilities for states corresponding to one location, and the dashed lines represent the coupling probabilities affecting the states between different locations. Figure 5.8(b) shows the detailed model and displays a single path for two locations where location *a* has three states and location *b* has four states. The coupled HMM

captures the spatial-temporal correlation between the neighboring locations. It can be trained to help recognize the behavior patterns from sensor data.

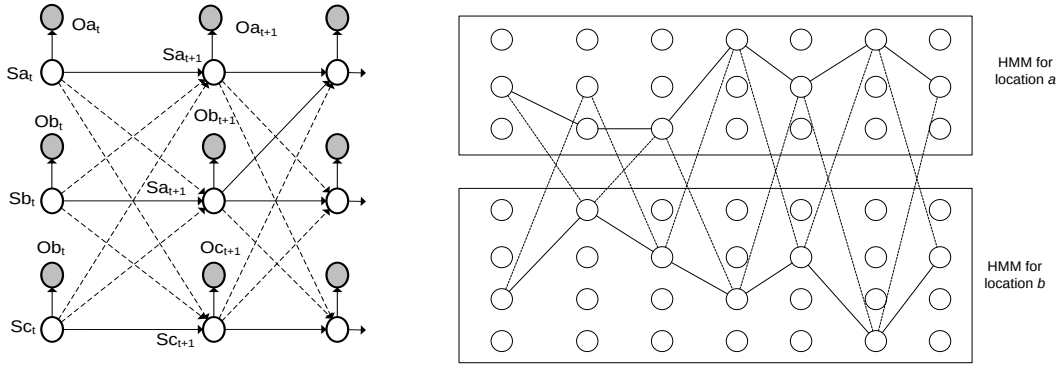


Figure 5.8 (a) CHMM for 3 locations (b) Detail CHMM for location a having 3 states and location b having 4 states

The posterior of a state sequence through the coupled HMM of two locations *a* and *b* can be obtained by using equation by Brand [46] as follows:

$$P(S \mid O) = \frac{\pi_{a_1} P(O_1^a \mid a_1)\pi_{b_1} P(O_1^b \mid b_1)}{P(O)} \prod_{t=2}^{T} P_{a_t|a_{t-1}} P_{b_t|b_{t-1}} P_{a_t|b_{t-1}} P_{b_t|a_{t-1}} P(O_t^a \mid a_t) P(O_t^b \mid b_t) \quad (5.1)$$

where π_{a_1} and π_{b_1} are the initial probabilities of states, $P_{a_t|a_{t-1}}$ and $P_{b_t|b_{t-1}}$ are the inner-state transition probabilities, $P_{a_t|b_{t-1}}$ and $P_{b_t|a_{t-1}}$ are the coupling probabilities modeling the interactions between two HMMs, $P(O_t^a \mid a_t)P(O_t^b \mid b_t)$ are the output probabilities of the states. From the observed data, we need to find a state sequence *S,* which maximizes *P(S|O)*. For each state, we need to compute both the inner-chain transition and coupling probabilities and using in equation (1) outputs the best state sequence *S* which involves recognized activity state sequences S_a for location *a* and S_b for location *b*. We can expand equation 5.1 for more than two locations, thus modeling the spatial-temporal relationship for activities in the environment.

Specifically, we follow the CHMM model used by [47] to carry out behavior pattern recognition. We develop an HMM for each of conference room, hallway, and cafeteria, and establish couplings between the three HMMs. We consider our model as a special case of dynamic Bayesian networks and consider each stream of an HMM as a continuous mixture. For the learning process of CHMM, the parameters can be defined as follows:

$$\pi_o^c(i) = P(q_t^c = i) \tag{5.2}$$

$$b_t^c(i) = P(O_t^c \mid q_t^c = i) \tag{5.3}$$

$$a_{i|j,k,l}^c(i) = P(q_t^c \mid q_{t-1}^c = j/k/l) \tag{5.4}$$

where q_t^c denotes the state of the coupled node for the location c at time t, $\pi_o^c(i)$ is the initial state probability, $b_t^c(i)$ is the observation probability and $a_{i|j,k,l}^c(i)$ is the transition probability for cth location at time t for the transition from state j, k and l state of other locations to state i .

The observation probability for Gaussian mixture components [48] is given by

$$b_t^c(i) = \sum_{m=1}^{M_i^c} (w_{i,m}^c N(O_t^c, \mu_{i,m}^c, U_{i,m}^c)) \tag{5.5}$$

where $\mu_{i,m}^c$ and $U_{i,m}^c$ represents the mean and covariance matrix of the i^{th} state of the coupled node with m_{th} component of the associated mixture node in the cth channel, M_t^c is the number of mixtures and $w_{i,m}^c$ represents the conditional probability $P(s_t^c = m \mid q_t^c = i)$ when s_t^c is the component of the mixture node in the c^{th} stream at time t. In the training phase, for each training observation sequence r, the data in the stream is managed according to the number of states of coupled node. A K-means algorithm [49] with M_i^c clusters can be used to determine a sequence of mixture components for each stream.

The new parameters of the model can be estimated as:

$$\mu_{i,m}^{c} = \frac{\sum_{r,t} \gamma_{r,t}^{c}(i,m) O_{r,t}^{c}}{\sum_{r,t} \gamma_{r,t}^{c}(i,m)} \quad (5.6)$$

$$U_{i,m}^{c} = \frac{\sum_{r,t} \gamma_{r,t}^{c}(i,m)(O_{r,t}^{c} - \mu_{i,m}^{c})(O_{r,t}^{c} - \mu_{i,m}^{c})^{T}}{\sum_{r,t} \gamma_{r,t}^{c}(i,m)} \quad (5.7)$$

$$w_{i,m}^{c} = \frac{\sum_{r,t} \gamma_{r,t}^{c}(i,m)}{\sum_{r,t} \sum_{m'} \gamma_{r,t}^{c}(i,m')} \quad (5.8)$$

$$a_{i|j,k,l}^{c} = \frac{\sum_{r,t} \varepsilon_{r,t}^{c}(i,j,k,l)}{\sum_{r,t} \sum_{j} \sum_{k} \sum_{l} \varepsilon_{r,t}^{c}(i,j,k,l)} \quad (5.9)$$

where,

$$\gamma_{r,t}^{c}(i,m) = \begin{cases} 1, & if\ q_{r,t}^{c} = i, s_{r,t}^{c} = m \\ 0, & otherwise \end{cases} \quad (5.10)$$

$$\varepsilon_{r,t}^{c}(i,j,k,l) = \begin{cases} 1, & if\ q_{r,t}^{c} = i, q_{r,t-1}^{c} = j/k/l \\ 0, & otherwise \end{cases} \quad (5.11)$$

$$s_{r,t}^{c} = \max_{m=1....M_i^c} P(O_t^c \mid q_t^c = i,m) \quad (5.12)$$

For finding the hidden states, we can use the Viterbi algorithm [50] for the CHMM as below:

Initialization:

$$\begin{aligned} &\delta_0(i,j,k) = \pi_0^A(i)\pi_0^B(j)\pi_0^C(k) b_t^A(i) b_t^B(j) b_t^C(k) \\ &\psi_0(i,j,k) = 0 \end{aligned} \quad (5.13)$$

Recursion:

$$\begin{aligned} &\delta_t(i,j,k) = \max_{l,m,n}\{\delta_{t-1}(l,m,n) a_{i|l,m,n} a_{i|l,m,n} a_{k|l,m,n}\} b_t^A(l) b_t^B(m) b_t^C(n) \\ &\psi_t(i,j,k) = \arg\max_{l,m,n}\{\delta_{t-1}(l,m,n) a_{i|l,m,n} a_{i|l,m,n} a_{k|l,m,n}\} \end{aligned} \quad (5.14)$$

Termination:

$$\begin{aligned} &P = \max_{i,j,k}\{\delta_T(i,j,k) \\ &\{q_T^A, q_T^B, q_T^C\} = \arg\max_{i,j,k}\{\delta_T(i,j,k) \end{aligned} \quad (5.15)$$

Backtracking:

$$\{q_T^A, q_T^B, q_T^C\} = \psi_{t+1}(q_{t+1}^A, q_{t+1}^B, q_{t+1}^C) \qquad (5.16)$$

where q_T^A, q_T^B, q_T^C represents the hidden states for each of the HMMs recognized during backtracking step of the Viterbi algorithm.

Recognizing the behavior patterns of a system from sensor data in real time can inform a simulation model for more accurate simulations of the system under study. In this work we propose an activity-informed dynamic data driven simulation framework and focus on behavior pattern recognition from sensor data that reflect the spatial-temporal activities of the system. We can use coupled HMM to carry out behavior pattern recognition and apply it to a smart office environment example.

CONCLUSION

This dissertation makes an important contribution in building occupancy simulation. The graph based agent oriented model is efficient in modeling a large number of occupants in any kind of building. It improves the performance of simulating a large number of occupants by representing only the basic features of the agents. In a crowd, an agent's behavior is limited by the factor that their free movement is restricted due to congestion and all the people need to move at the same speed. As such, omitting the unnecessary features can conserve resource and improve performance highly. In our model, we use only the required features like moving, staying or waiting and add other features on top of it when required. We model the basic occupancy dynamics using mathematical models like queuing theory and flow model and were able to obtain the same occupancy behavior as other existing models while improving the performance.

The graph-based agent-oriented model takes a significant leap to build models which require a large number of agents because rather than relying on a singular model, it synthesizes two popular methods of agent and graph-based. Such an amalgamation promises added benefits to any model that initiates from the technique that we follow because one of the primary goals of our model is also to avoid the limitations of existing models. The model can be applied not only to building occupancy simulation, but also where a large number of agents need to be modeled for analyzing their emergent behaviors. In our experiments, we could simulate a large number of agents and yet observe the behavior developed by the individuality of agents. In our future work, we plan to compare and validate our model with the existing works and real scenarios. We also plan to analyze the level of complexity we can simulate as we increase the number of agents.

We used data assimilation for real-time estimation using observations from sensor data. Video sensors we used which allowed us to get information like occupancy count and flow in the

rooms. We use Sequential Monte Carlo method, also known as Particle Filters as our data assimilation algorithm. SMC method is suitable for a nonlinear, non-Gaussian and non-analytical model like graph based agent oriented where occupants may exhibit emergent behaviors based on their individual property. We developed data assimilation framework which assimilates the sensor data with the graph based agent oriented model using SMC algorithm. PF consists of particles each of which represents a possible system state with some weight. In each iteration, new particles are generated and are resampled based on weight.

To reduce the computational cost of data assimilation and improve the performance, we set the state size minimal and consider only the node information. This significantly reduces the computational cost since number of nodes is quite less compared to number of agents which might in number of thousands. The graph based agent oriented model is in-itself a computationally cheap model. Thus, the overall data assimilation framework is also efficient and much faster than the existing real-time agent based models. We ran experiments for two different layouts of buildings for a various number of occupants in various scenarios. All the scenarios consist of congestion, and we could accurately estimate the congestion in real-time. We reduced the number of nodes having a sensor and still maintained the accuracy of estimation. We also developed a new data assimilation framework which creates particles from the available sensor data to improve the existing particles.

In this dissertation, we also presented other works done in building occupancy simulation. We developed an agent based model for simulating smart office environment. The model is appropriate for the environment with few number of occupants. We also used Hidden Markov Model to estimate behavior patterns of smart office environment by training from historical data. We have also presented a coupled HMM framework for utilizing the learned behavior pattern in

dynamic data driven simulation. It will allow to make a real-time estimation of the simulation and provide a higher-level information to data assimilation in improving the performance. We also developed a hybrid model which allows pure agents in areas where it is important to observe emergent behaviors.

Several future research exist for the continuation of the current work. The graph based agent oriented model may be used in other applications where agents are computationally expensive like traffic, boids simulation, etc. Various learning methods may be applied to the sensor data to analyze occupancy patterns. These learned properties can be used to improve the performance of data assimilation. In this work, we used only one sensor, however, more than one sensor might be available in buildings which can provide various kinds of information. We can develop an appropriate framework to assimilate different kinds of sensor data. Another remaining work is to validate the model in a real system with real sensor data.

REFERENCES

[1] E. D. Kuligowski, R. D. Peacock, and B. L. Hoskins, *A review of building evacuation models*. US Department of Commerce, National Institute of Standards and Technology Gaithersburg, MD, 2005.

[2] I. Gaetani, P.-J. Hoes, and J. L. M. Hensen, "Occupant behavior in building energy simulation: Towards a fit-for-purpose modeling strategy," *Energy Build.*, vol. 121, pp. 188–204, Jun. 2016.

[3] C. Wang, D. Yan, and Y. Jiang, "A novel approach for building occupancy simulation," *Build. Simul.*, vol. 4, no. 2, pp. 149–167, Jun. 2011.

[4] C. M. Macal and M. J. North, "Agent-based Modeling and Simulation," in *Winter Simulation Conference*, Austin, Texas, 2009, pp. 86–98.

[5] G. Franz, H. A. Mallot, and J. M. Wiener, "Graph-based Models of Space in Architecture and Cognitive Science – a Comparative Analysis," in *Proceedings of the 17th International Conference on Systems Research, Informatics and Cybernetics (intersymp'2005), Architecture, Engineering and Construction of Built Environments, Pages 30–38. the International Institute for Advanced Studies in System R*, 2005, pp. 30–38.

[6] F. Bouttier and P. Courtier, "Data assimilation concepts and methods March 1999," *Meteorol. Train. Course Lect. Ser. ECMWF*, 2002.

[7] C. Rottondi, M. Duchon, D. Koss, G. Verticale, and B. Schätz, "An energy management system for a smart office environment," in 2015 International Conference and Workshops on Networked Systems (*NetSys)*, 2015, pp. 1–6.

[8] M. Green, *Building Codes for Existing and Historic Buildings*. John Wiley & Sons, 2011.

[9] N. Gilbert, *Agent-Based Models*. SAGE, 2008.

[10] J. Hutchins, A. Ihler, and P. Smyth, “Modeling Count Data from Multiple Sensors: A Building Occupancy Model,” in *2nd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2007. CAMPSAP 2007*, 2007, pp. 241–244.

[11] K. P. Lam *et al.*, “Occupancy detection through an extensive environmental sensor network in an open-plan office building,” in *Proc. of Building Simulation 09, an IBPSA Conference*, 2009.

[12] S. Rai and X. Hu, “Behavior Pattern Detection for Data Assimilation in Agent-Based Simulation of Smart Environments,” in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2013, vol. 2, pp. 171–178.

[13] A. Doucet, N. de Freitas, and N. Gordon, “An Introduction to Sequential Monte Carlo Methods,” in *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon, Eds. Springer New York, 2001, pp. 3–14.

[14] X. Feng, D. Yan, and T. Hong, “Simulation of occupancy in buildings,” *Energy Build.*, vol. 87, pp. 348–359, Jan. 2015.

[15] M. Maasoumy, “Estimation of Occupancy Distribution in Buildings,” *Univ. Calif. Berkeley*, 2009.

[16] D. Cook and S. Das, *Smart Environments: Technology, Protocols and Applications (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2004.

[17] S. Bandini, F. Rubagotti, G. Vizzari, and K. Shimura, “An Agent Model of Pedestrian and Group Dynamics: Experiments on Group Cohesion,” in *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, R. Pirrone and F. Sorbello, Eds. Springer Berlin Heidelberg, 2011, pp. 104–116.

[18] O. Beltaief, S. E. Hadouaj, and K. Ghedira, “Multi-agent simulation model of pedestrians crowd based on psychological theories,” in *2011 4th International Conference on Logistics*, 2011, pp. 150–156.

[19] B. Banerjee, A. Abukmail, and L. Kraemer, “Advancing the Layered Approach to Agent-Based Crowd Simulation,” in *22nd Workshop on Principles of Advanced and Distributed Simulation, 2008. PADS ’08*, 2008, pp. 185–192.

[20] Q. Qin and J. Wei, “An agent-based approach for crowd dynamics simulation,” in *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, 2010, vol. 2, pp. 78–82.

[21] T.-S. Shen, “ESM: a building evacuation simulation model,” *Build. Environ.*, vol. 40, no. 5, pp. 671–680, May 2005.

[22] R. Tomastik, S. Narayanan, A. Banaszuk, and S. Meyn, “Model-Based Real-Time Estimation of Building Occupancy During Emergency Egress,” in *Pedestrian and Evacuation Dynamics 2008*, W. W. F. Klingsch, C. Rogsch, A. Schadschneider, and M. Schreckenberg, Eds. Springer Berlin Heidelberg, 2010, pp. 215–224.

[23] Lidi Huang, Deming Liu, and Yongyi Zhang, “Dynamics-Based Stranded-Crowd Model for Evacuation in Building Bottlenecks,” *Math. Probl. Eng.*, pp. 1–7, Jan. 2013.

[24] V. Garg and N. K. Bansal, “Smart occupancy sensors to reduce energy consumption,” *Energy Build.*, vol. 32, no. 1, pp. 81–87, Jun. 2000.

[25] Z. Yang, N. Li, B. Becerik-Gerber, and M. Orosz, “A Multi-sensor Based Occupancy Estimation Model for Supporting Demand Driven HVAC Operations,” in *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, San Diego, CA, USA, 2012, p. 2:1–2:8.

[26] D. De, W. Z. Song, M. Xu, C. L. Wang, D. Cook, and X. Huo, “FindingHuMo: Real-Time Tracking of Motion Trajectories from Anonymous Binary Sensing in Smart Environments,” in *2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 163–172.

[27] N. Bei, B. de Foy, W. Lei, M. Zavala, and L. T. Molina, “Using 3DVAR data assimilation system to improve ozone simulations in the Mexico City basin,” *Atmospheric Chem. Phys.*, vol. 8, no. 24, pp. 7353–7366, Dec. 2008.

[28] J. Wilkin, J. Zavala-Garay, J. Levin, and W. G. Zhang, “Four-Dimensional Variational Assimilation of Satellite Temperature and Sea Level Data in the Coastal Ocean and Adjacent Deep Sea,” in *IGARSS 2008 - 2008 IEEE International Geoscience and Remote Sensing Symposium*, 2008, vol. 3, p. III-427-III-430.

[29] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*, 2nd ed. Upper Saddle River, N.J: Prentice Hall Information and, 2000.

[30] C. Antoniou, M. Ben-Akiva, and H. N. Koutsopoulos, “Nonlinear Kalman Filtering Algorithms for On-Line Calibration of Dynamic Traffic Assignment Models,” *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 4, pp. 661–670, Dec. 2007.

[31] H. Xue, F. Gu, and X. Hu, “Data Assimilation Using Sequential Monte Carlo Methods in Wildfire Spread Simulation,” *ACM Trans Model Comput Simul*, vol. 22, no. 4, p. 23:1–23:25, Nov. 2012.

[32] F. Gustafsson *et al.*, “Particle filters for positioning, navigation, and tracking,” *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 425–437, Feb. 2002.

[33] J. Zhang, R. Chen, C. Tang, and J. Liang, “Origin of scaling behavior of protein packing density,” *J. Chem. Phys.*, vol. 118, no. 13, pp. 6102–6109, Apr. 2003.

[34] T. Chen, J. Morris, and E. Martin, “Dynamic data rectification using particle filters,” *Comput. Chem. Eng.*, vol. 32, no. 3, pp. 451–462, Mar. 2008.

[35] M. Wang and X. Hu, “Data Assimilation in Agent Based Simulation of Smart Environment,” in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, New York, NY, USA, 2013, pp. 379–384.

[36] R. Aylett and M. Cavazza, “Intelligent Virtual Environments-A state-of-the-art report,” *Proc. Eurographics 2001 STARs*, vol. 3, 2001.

[37] J. J. Fruin, *Pedestrian planning and design*. Metropolitan Association of Urban Designers and Environmental Planners, 1971.

[38] K. Ando, H. Ota, and T. Oki, “Forecasting the Flow of People,” *Railw. Res. Rev.*, vol. 45, pp. 8–14.

[39] R. H. Reichle, “Data assimilation methods in the Earth sciences,” *Adv. Water Resour.*, vol. 31, no. 11, pp. 1411–1418, Nov. 2008.

[40] W. A. Lahoz and P. Schneider, “Data assimilation: making sense of Earth Observation,” *Front. Environ. Sci.*, vol. 2, May 2014.

[41] X. Hu, “Dynamic data driven simulation,” *SCS MS Mag.*, vol. 1, pp. 16–22, 2011.

[42] L. Rabiner and B. Juang, “An introduction to hidden Markov models,” *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[43] W. A. Hoff and J. W. Howard, “Activity Recognition in a Dense Sensor Network.,” in *SNA*, 2009, pp. 67–72.

[44] L. Kratz and K. Nishino, “Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 1446–1453.

[45] L. Wang, T. Gu, X. Tao, and J. Lu, “Sensor-Based Human Activity Recognition in a Multi-user Scenario,” in *Ambient Intelligence*, M. Tscheligi, B. de Ruyter, P. Markopoulus, R. Wichert, T. Mirlacher, A. Meschterjakov, and W. Reitberger, Eds. Springer Berlin Heidelberg, 2009, pp. 78–87.

[46] M. Brand, “Coupled hidden Markov models for modeling interacting processes,” 1997.

[47] A. V. Nefian, L. Liang, X. Pi, L. Xiaoxiang, C. Mao, and K. Murphy, “A coupled HMM for audio-visual speech recognition,” in *in International Conference on Acoustics, Speech and Signal Processing (CASSP’02*, 2002, pp. 2013–2016.

[48] D. Reynolds, “Gaussian Mixture Models,” in *Encyclopedia of Biometrics*, S. Z. Li and A. K. Jain, Eds. Springer US, 2015, pp. 827–832.

[49] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2 edition. New York: Wiley-Interscience, 2000.

[50] I. Rezek and S. J. Roberts, “Estimation of coupled hidden Markov models with application to biosignal interaction modelling,” in *NEURAL NETWORKS SIGNAL PROCESS PROC IEEE*, 2000, vol. 2, pp. 804–813.

[51] A. Krogh, B. Larsson, G. von Heijne, and E. L. L. Sonnhammer, “Predicting transmembrane protein topology with a hidden markov model: application to complete genomes1,” *J. Mol. Biol.*, vol. 305, no. 3, pp. 567–580, Jan. 2001.

[52] A. K. Dey, G. D. Abowd, and D. Salber, “A context-based infrastructure for smart environments,” in *Managing Interactions in Smart Environments*, Springer, 2000, pp. 114–128.

[53] A. B. Poritz, “Hidden Markov models: a guided tour,” in *, 1988 International Conference on Acoustics, Speech, and Signal Processing, 1988. ICASSP-88*, 1988, pp. 7–13 vol.1.