JAIST Repository
https://dspace.jaist.ac.jp/

Title	大容量交通流のメゾスコピックモデル化手法に関する研究
Author(s)	上原, 健嗣
Citation	
Issue Date	2022-09
Type	Thesis or Dissertation
Text version	ETD
URL	http://hdl.handle.net/10119/18142
Rights	
Description	Supervisor: 平石 邦彦, 情報科学研究科, 博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

Doctoral Dissertation

A Study on Mesoscopic Modeling Methodologies for Large Volume Traffic Flow Data

Kenji UEHARA

Supervisor: Kunihiko HIRAISHI

School of Information Science
Japan Advanced Institute of Science and Technology

September, 2022

Abstract

With improvements in computing power and the widespread use of sensor technology, highly accurate and high frequency traffic flow data is now readily available. Traffic flow data can be modeled for various applications, but handling these data requires large-scale storage and high-speed computing power. Therefore, this dissertation focuses on *Mesoscopic Modeling* for large-scale traffic flow data to efficiently obtain useful information. A mesoscopic model in traffic flow is a network model in which continuous values are discretized by replacing the main points in the traffic flow as nodes and the movement information with statistical values. Furthermore, when combined with space partition and trajectory clustering, the data can be abstracted to a level that does not miss important traffic flow information while significantly reducing the data size.

Since the modeling process requires time and effort, this method applies process mining techniques to streamline the process of creating mesoscopic models. Process mining is a technique for analyzing event logs output by a system to discover, verify, and improve process models. Applied to modeling, it enables event sequence analysis, state transition analysis, feature extraction of traffic flow, filtering of unnecessary data, and extraction of statistical information, thereby significantly reducing the modeling effort. Furthermore, process mining can be used for conformance checks to ensure that the model is operating correctly according to the design.

The mesoscopic model constructed by this method enables high-speed simulations because the amount of data is greatly reduced while retaining important information. Therefore, the mesoscopic model is very useful and beneficial in the planning phase of traffic measures, when the current traffic flow is to be grasped quickly, or when a large number of simulations with finely varied conditions are to be performed.

Keywords: Traffic flow simulation, Mesoscopic model, Object Petri nets, Process Mining, Trajectory clustering

Acknowledgement

I would like to sincerely thank Professor Kunihiko Hiraishi for his long term guidance in completing this dissertation. This research began with the idea that procedural processing in discrete event systems can be applicable to traffic flow data. From there, it evolved into the modeling of traffic flow on airport surfaces, which led to the construction of the method for modeling traffic flow itself, the subject of this dissertation. He has instructed on object Petri nets, process mining, and simulation modeling, which are necessary building components for this purpose. I would also like to thank for his guidance regarding the submission of many papers. Producing each paper was hard work, and after submitting one, I spent time searching for the next idea. However, these academic days in my life are the freshest I have ever had, and I feel a little bit lonely that they are coming to an end.

I would also like to thank Professor Atsuo Ozaki of Osaka Institute of Technology, Professor Kazuhiro Ogata, Professor Kokolo Ikeda and Associate Professor Daisuke Ishii of Japan Advanced Institute of Science and Technology for serving as reviewers. Thank you for your valuable feedback as we finalize and refine this dissertation. Professor Kokolo Ikeda also guided as the advisor of minor research project, and I was able to compile the outcome of the minor research into a journal paper. I also thank Associate Professor Koichi Kobayashi of Hokkaido University for co-authoring a paper. These papers are the important components of the chapters in this dissertation.

Finally, I send my utmost appreciation to my wife and young children. When I think of my children, who were not with me when I entered the doctoral program and are now growing up healthy, I realize the years I have devoted to this research and the changes that have occurred in the environment around me. In my life of balancing work and research, sometimes I had to spend part of the time I should have spent with my family on research. From now on, I will spend my time with my family, which I value more than anything else. I dedicate this doctoral dissertation to my family.

Contents

Chapter 1

Introduction

With the improvement of computer performance and the spread of sensor technologies such as cameras and GNSS, highly accurate and frequent traffic flow data have become readily available. Although traffic flow data can be modeled and used for various applications, handling such data requires huge storage space and large-scale computing power. Therefore, we focus on the *Mesoscopic model* in which continuous values are discretized by replacing them with statistical information. The mesoscopic model is a network model that reduces data size by discretization, but leaves the major streams of traffic flow at a level where bottlenecks and stagnation points can be identified. This dissertation develops a mesoscopic modeling framework from raw temporal and spatial data of traffic flows with abstraction based on spatial importance and guaranteed conformance to design. Furthermore, as a demonstration of the modeling process using this framework, we model airport surface traffic flow and airspace traffic flow. We show that it is possible to downsize the amount of data to a few tenths of the original data while creating a model that reflects the characteristics of the traffic flow and enables the model designers to achieve their goals.

1.1 Motivation

1.1.1 Background and motivation

Traffic flow data modeling is useful for calculating traffic throughput, analyzing congestion and stagnation [1], and simulating fuel consumption and CO_2 emissions [2] [3]. As sensor technology becomes more widespread and performance improves, more accurate and high-frequency data becomes available. Recently, the size of the data becomes huge and the required computational power increases. However, the purpose of

analyzing traffic flow data has not changed significantly since the days when the data was low-resolution.

Although an essential role of science is to develop models of nature [4], the benefit is not the reproduction of the event itself, but the ability to simulate further conditions and virtually check the changes in a complex environment. Taking traffic flow modeling as an example, there are various purposes such as road planning to mitigate traffic congestion and introducing traffic policies to deal with economic losses. The greater the speed the better and the greater the capacity the better [5], since the greater the amount of traffic flow can be carried at one time the better, unless there is a specific reason not to. Therefore, if factors that impede speed and capacity and stagnation points exist, it is necessary to introduce countermeasures to improve them [6]. However, as long as costly and time-consuming measures are taken, the expected effects must be achieved. In particular, public organizations that develop infrastructure, such as roads and ports, are always required to achieve cost-effectiveness for their budgetary resources, and therefore cannot make a decision to introduce a system if the possibility of achieving the desired effect is unclear. Therefore, the model is used to understand the current traffic flow and to simulate the effects of introducing new measures [7]. This objective is not expected to change significantly in the future.

Data that is larger than necessary due to higher resolution may improve the accuracy of the simulation model, but it requires more time and processing power for analysis and may make it difficult to achieve the original purpose of modeling. If the analyst only wants to get an overview of the traffic flow, the resolution of the data is minimal and does not need to be high resolution. Therefore, this dissertation focuses on the *Mesoscopic model*, which can obtain the information that the model designer wants to know even if the huge raw data are down-sampled to a certain level of discretization [8].

While mesoscopic modeling has the above advantages, there are two challenges in its creation. The first is the extraction of major paths in the space and the creation of an abstracted route graphs. To reduce the amount of data while retaining important information, it is necessary to eliminate unimportant trajectories and generate an abstracted route graph after selecting major routes and key traffic nodes. Mesoscopic models are created based on the abstracted route graphs, but optimal route and node

selection is not easy, as it depends on the modeling objectives and characteristics of the traffic flow data. The second point is to confirm that the model is behaving as the designer intended. Even if a mesoscopic model is created, it cannot be used for user's purposes unless it behaves correctly. Therefore, the behavior of the model must be inspected, which requires the same amount of effort as creating the model. In addition, a mechanism for inspection needs to be built into the model, and it is difficult to say that the same mechanism can be designed for all types of traffic flow data.

Through this dissertation, we solve the first problem by developing a method for creating abstracted route graphs. We will also apply *Process Mining* techniques to modeling to improve creation efficiency and check conformance to specifications, thereby resolving the second problem. The ultimate goal is to build a framework for mesoscopic modeling of large-scale traffic flow data.

1.1.2 Research questions

Based on the motivation of 1.1.1, this dissertation sets up the following three problems. This dissertation will clarify the answers to these questions through the construction of the mesoscopic modeling approach. The answers this dissertation provides to Q1-Q3 are presented in the final chapter.

Q1 Modeling with down-sampled data and its accuracy: When modeling from large volume traffic flow data, is it possible to reduce the amount of data by down-sampling to some extent instead of dealing directly with continuous data and still obtain the results the model designer desired?

Q2 Selectable modeling abstraction: Is mesoscopic modeling possible, where the amount of data reduction can be adjusted according to the designer's objectives and the degree of macro and micro can be selected?

Q3 Checking model correctness: How can we check that the model is behaving correctly? Is it possible to introduce a common checking method for traffic flow mesoscopic models instead of installing a special mechanism in the model?

1.2 Dissertation Contributions

As mentioned above, the goal of this dissertation is to build a framework for mesoscopic modeling that abstracts the raw data of high-volume traffic flows and simulates them in a way that is comparable to using the raw data. We build a methodology by systematizing the necessary processes. The advantages and novelties of the method are as follows:

1. **Significant reduction in data size.** In the mesoscopic modeling process, continuous raw data is discretized by replacing it with statistical values as movement information, which significantly reduces the data volume. This facilitates analysis and reduces the required computing power. Even though the data size can be less than 1/100th, the traffic flow features are preserved, allowing analysis of flows and bottlenecks.

2. **Application of process mining techniques to improve modeling efficiency and ensure conformance to specifications.** Since modeling work requires time and effort, this framework applies process mining techniques to streamline the process of creating mesoscopic models. Process mining is a technique for analyzing event logs output by a system to discover, verify, and improve process models [9]. Applying this to modeling enables event sequence analysis, state transitions analysis, feature extraction of traffic flow, filtering of unnecessary data, and extraction of statistical information, thereby significantly reducing the modeling effort.

3. **Lightweight and fast simulation.** The mesoscopic model constructed with this framework enables high-speed simulation because the amount of data is significantly reduced while retaining important information. Therefore, the mesoscopic model is useful for quickly grasping the current traffic flow in the planning phase of traffic measures, or for performing a large number of simulations with various conditions.

4. **Abstracted route graphs.** We identify key points in the traffic flow and create abstracted route graphs. Traffic flow data is not only simple and straightforward, such as vehicles on a road or trains on a railroad track, but also highly random data where

the path is not constant or where an altitude factor is added, such as aircraft. It is not easy to extract abstraction graphs from these data, but this dissertation proposes a method to extract abstraction graphs from traffic flow data with non-constant trajectories, using airport surface and airspace traffic flow data as an example.

5. **Development of a generalized workflow for modeling large volume traffic flow data.** Through this dissertation, we present a generalized workflow for creating simulation models from large volume traffic flow data. While many previous studies have shown methods for individual events, this dissertation presents a systematic and generalized method, which is one of the novelties presented in this dissertation.

1.3 Structure of the Dissertation

The modeling framework proposed in this dissertation aims to abstract large volume traffic flow data without missing necessary information and finally create mesoscopic model. The required technology and method building are organized in the following systematics.

- **Chapter 2: Existing Technologies**
  To build a mesoscopic modeling framework for traffic flow, we first introduce related works on traffic flow modeling. All traffic flow models, which are the output of this dissertation, are composed of object Petri nets and process mining technique. Since object Petri nets can replicate instances from a base Petri net, they are used to reproduce traffic flow by generating a large number of moving objects. Process mining is also used to efficiently extract statistical information from raw traffic flow data, in addition to checking model conformance to specifications. Therefore, these existing technologies are explained as an introduction to the mesoscopic modeling methodology to be developed in later chapters.

- **Chapter 3: Discrete Event Modeling with Object Petri Net**
  In this chapter, a method for creating a discrete event model is developed and applied

to mesoscopic modeling in the later stages of this dissertation. The output model created abstracts traffic flows and makes each moving object operate in parallel. In addition, each moving object operates procedurally according to predefined rules. Based on these characteristics of the modeling target, we adopt object Petri nets as the model description language. In this dissertation, we show how to construct a hierarchical large-scale model by combining many small subnets with object Petri nets.

- **Chapter 4: Process Mining Approach for the Conformance Checking of Discrete-Event Simulation Model**

  In this chapter, we apply process mining techniques to the specification conformance checks of the mesoscopic models created, and build inspection methods to reduce the workload on developers. In order for the simulation results to have validity, it is necessary to show that the model is working correctly as designed. Here, process mining is used to show that the model behaves correctly as intended by the designer and that at least the inspected model output contains no errors and is safe for use in traffic flow analysis.

- **Chapter 5: A Method for Mesoscopic Modeling of Large Volume Traffic Flow Applying Process Mining Techniques**

  In this chapter, the airport surface traffic flow is abstracted leaving the network framework at a level where bottlenecks and stagnation area can be identified, and is represented by an abstraction graph consisting of nodes and links. Furthermore, a mesoscopic model is created based on the graph. By applying process mining techniques to the modeling, we propose an efficient method for identifying traffic patterns, removing noise, and extracting statistical information, thereby significantly reducing the time cost and work required.

- **Chapter 6: A Framework for Extracting Abstracted Route Graphs Toward Air Traffic Flow Modeling**

  While mesoscopic modeling reduces the amount of data, it is important to determine the level of abstraction to avoid missing information necessary to reproduce traffic

flow. Therefore, we consider eliminating unimportant trajectories and extracting major routes in the airspace. The node selection that results in the optimal level depends on the purpose of the modeling and the characteristics of the traffic flow data. For example, it is more difficult to extract features of traffic flow for aircraft than for vehicles or railroads because the trajectory is not constant for the unavailability of fixed paths such as tracks or roads, and the complexity is increased due to altitude factors. Therefore, it is necessary to keep only important spaces and exclude unnecessary spaces, as well as to extract routes frequently passed by aircraft as the main points of traffic. In this dissertation, we propose a framework that combines the methods of space partition, clustering, and skeleton extraction to extract major routes of air traffic flow and create abstracted route graphs.

- **Chapter 7: Mesoscopic Modeling, Simulation and Evaluation for Air Traffic Flow in the Airspace**

  Based on the abstract graph, mesoscopic modeling is performed using object Petri nets. The model output is further compared to real data to evaluate that the mesoscopic model downsizes the raw data of the traffic flow while still being accurate enough to meet the analyst's objectives.

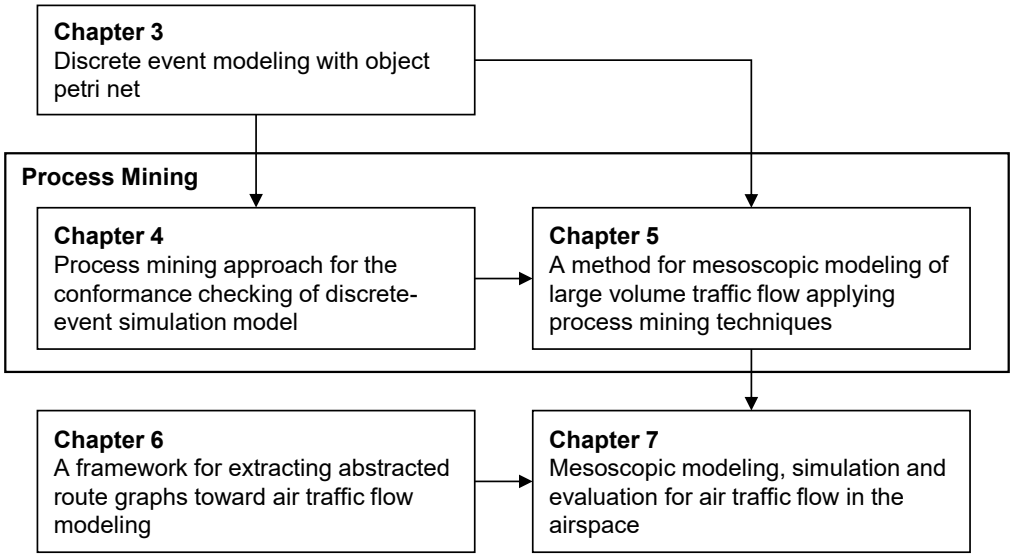


Fig. 1: Core structure of the dissertation and relevance of each chapter

Fig. 1 shows the role and relevance of each chapter. Chapter 2 describes the modeling of traffic flow, which is the subject of this dissertation, and deals with the existing technologies used throughout the chapters: object Petri nets and process mining. In Chapters 3 and 4, discrete event modeling and specification conformance checking methods are developed. Based on this, mesoscopic modeling techniques are established in Chapters 5 through 7, and actual airport surface traffic flow and airspace traffic flow models are developed in Chapters 6 and 7. We discuss the created mesoscopic model and the remaining future issues in Chapter 8, and finally, the usefulness of the model and how it can be used is discussed in Chapter 9.

Chapter 2

Existing Technologies

There are many different types of traffic flows, including vehicular flows, aircraft and vessels, and human and animal behavior. This chapter describes related research on traffic flow modeling, which is the main theme of this dissertation. In this dissertation, mesoscopic models are actually created from large volume traffic flow data in order to establish a modeling framework, and common techniques used throughout the chapters are described.

2.1 Traffic Flow Modeling

As mentioned in Chapter 1, traffic flow modeling can be broadly divided into macroscopic and microscopic models based on their granularity. This dissertation deals with mesoscopic models, which are intermediate between the two models.

The macroscopic model is a mathematical model that formulates the characteristics of traffic flow, such as traffic flow density and velocity. This model allows traffic flow to be considered in the same way as a fluid in natural phenomena, treating the entire traffic flow as a single entity. Traffic flow is considered a fluid composed of vehicles, but individual vehicles do not explicitly appear in the theory [10]. The history of macroscopic models of traffic flow is long, and the first model was the LW-model by Lighthill and Whitham (1955) [11], who compared traffic flow on a road with water movement in a river and expressed it in terms of nonlinear first-order partial differential. It is a one-dimensional fluid and the first fluid model. Richards complemented the LW model and introduced the idea of shock waves to complete the LWR model (1956) [12]. The LWR model was further applied to a complex network model consisting of intersections and roads [13] [14].

One of the reasons why traffic flow was treated like a single natural phenomenon

and formulated as a fluid by the macroscopic model is that computers were immature, and it was difficult to reproduce even the motion of individual particles. However, with the development of computers, the microscopic model, in which the individual motions of many components are analyzed directly by simulation and the traffic flow event is treated as the total of these components, has developed rapidly since the 1990s. Chowdhury et al. viewed traffic flow as a system in which a large number of constituent molecules interact and explained it dynamically as a phenomenon due to collective motion [10]. Microscopic models are broadly classified into car (vehicle) following models, Cellular Automata (CA) models, and multi-agent models.

The car-following model focuses on the motion of individual vehicles, and the driver controls his or her own vehicle in response to the behavior of the vehicle directly in front of him/her. The model is simple: if the car in front stops, the following car stops, and if the car in front moves, the following car moves. A representative tracking model is the Optimal Velocity (OV) model by Bando et.al [1]. The OV model is a one dimensional follow-up model, the size of the car is not considered, and all driver and car performances are assumed to be the same. The only properties that the constituent vehicle motions must have in order to describe the traffic flow are the following behavior and the exclusion effect. There are other models such as the Gipps model [15] and the Intelligent Driver model (IDM) [16], the simple behavior is the same: stop if there is a blockage in front and go forward if the road ahead is clear.

Furthermore, the use of the CA greatly simplifies the problem and makes it easier to handle. After all, if there is only one vehicle, it can be expressed as a simple function of speed and time. However, when there are many of them, competing with each other for resources (time and space), interference occurs, and complex behavior is manifested. This interference can be simulated simply as a relationship between two moving vehicles, and CA is a model that simplifies to just adjacent cells. the Rule 184 model by Wolfram is a typical application of CA and is the most basic traffic simulation model [17].

Microscopic models reproduce complex behavior in which individual elements aggregate and appear as a whole. A multi-body or multi-agent model describes the behavior of simple individuals or agents, such as human society or individual organisms, interacting in large numbers [18]. Although theoretical analysis is possible with multi-body models, computer simulation is suitable for reproducing complex traffic flows by

collecting large numbers of individuals with simple behaviors and rules and operating them on a large scale.

The model presented in this dissertation as an example of a realization of the proposed methodology is structured with agents with rules moving around on an abstracted network model. The aircraft passenger boarding model, airport surface traffic flow model, and airspace traffic flow model, which will be constructed in the next chapters, are all designed with the same concept, in which many agents (humans and aircraft) repeatedly progress and stop depending on whether there are obstacles in front of them, and head toward their destination. A large number of individuals performing simple actions are collected and operated concurrently to reproduce complex traffic flow.

2.2 Mesoscopic Modeling of Large Traffic Flow

Mesoscopic means between the macroscopic and microscopic scales [8] and is generally used in the field of nanotechnology. In nanotechnology, the properties exhibited by an observable may change between the macroscopic (bulk) and microscopic scales, and the intermediate scale at which this mutation occurs is called a mesoscopic system. On the other hand, mesoscopic in the traffic flow model treated in this dissertation is used only in terms of spatial and temporal data granularity, noting that the nature of the observation target does not change with scale. In other words, from the opposite perspective of nanotechnology, the emphasis is on the ability to discretize data and mesoscopic modeling to show the same traffic flow properties/characteristics regardless of scale.

Here, we discuss data granularity and scale in traffic flow. As an example, let us consider the case of analyzing traffic flow on highway. If the analyst's goal is to get a rough characterization, then macroscopic is appropriate for capturing the entire flow. In this case, data that are too granular rather inhibit the identification of characteristics [8] [19], so discretization is necessary to capture the target as a group, without focusing on the behavior of individuals [19]. On the other hand, when the detailed behavior of individual vehicles, dynamics such as acceleration/deceleration and braking, are the purposes of analysis, the behavior of each individual vehicle must be modeled microscopically, since abstracted data cannot accurately reproduce them [8].

We attempt to establish a mesoscopic modeling method which can adjusts the degree of abstraction between the macro and micro levels, with the main goal of reducing the amount of data without losing the desired information. For example, if the objective is to identify bottlenecks or stagnation areas in a particular space, and if the traffic flow is not completely random but has some pattern, a mesoscopic model that abstracts continuous data without losing the characteristics of the traffic flow is sufficient.

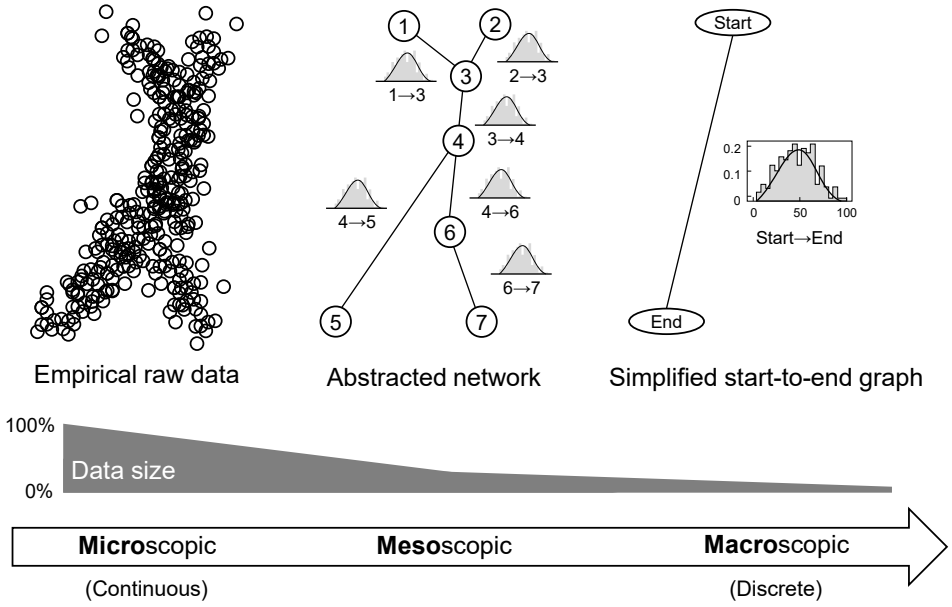
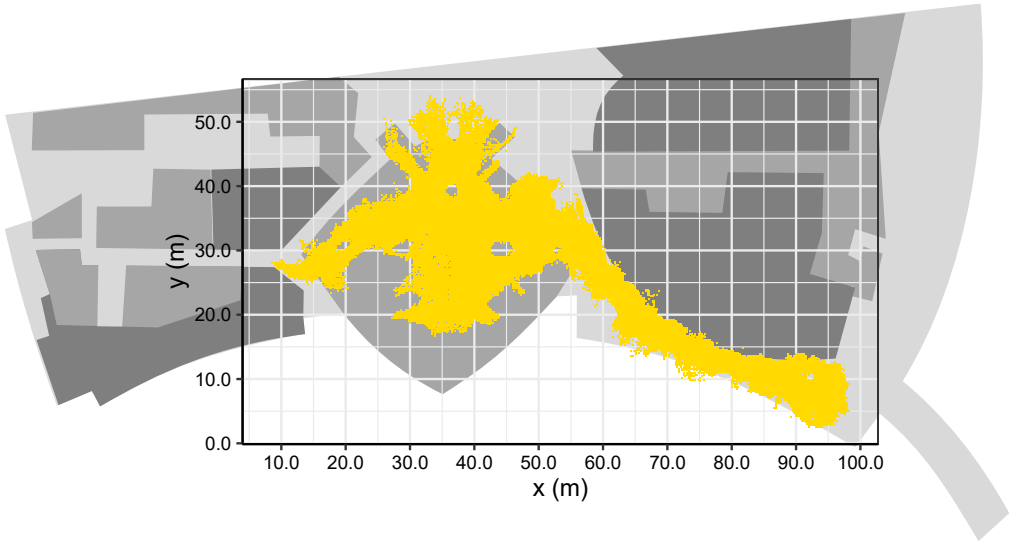


Fig. 2: Perspectives on data granularity in traffic flow data. Mesoscopic is the middle between raw data of continuous values (micro) and simplified two-point data (macro).

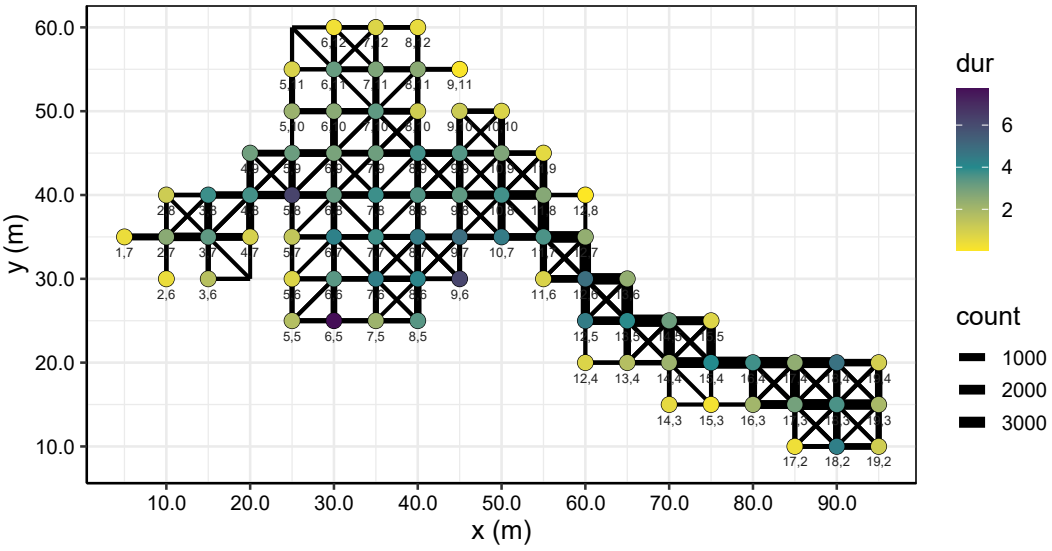
Fig. 2 shows the modeling perspective and the required data granularity. There are two poles: a microscopic model that deals directly with continuous data, and a macroscopic model that abstracts traffic flow to only start and end points. The mesoscopic model is intermediate between the two and is a network model [19] consisting of nodes (intersections) and edges (connecting paths). The size of traffic flow data is enormous when handled directly as raw data, but decreases as the data approaches macroscopic levels, eventually becoming only a two-point graph and statistical information. The analysts can decide the level of abstraction depending on the properties they want the model to reproduce.

Using pedestrian flow data as an example, let us divide the space into grids and

replace it with movement information between each grid (node). Here, we use pedestrian movement data in a shopping center [20].



(a) Pedestrian movement (continuous, microscopic)



(b) Pedestrian movement (abstracted, mesoscopic)

Fig. 3: Pedestrian tracking data in a shopping mall. (a) shows a direct plot of the raw data of continuous values. (b) shows the data discretized on a 5.0 m grid. In (b), each node is colored by time spent, and line width of each edge represents the occurrence frequency of transition between two nodes.

Fig. 3 shows the result of replacing continuous values with movement statistics

information between nodes and gridding the pedestrian data. The frequency of traffic is reflected in the thickness of the line segments between adjacent nodes, and the dwell time at each node is shown in color. It can be seen that downsizing to this level still provides an understanding of congestion points and major traffic flows. The data set used contains dynamic information for approximately 18 000 people for one day, and the data size exceeds 1GB. Here, the node is formed on a 5.0 m grid, and the amount of data is only 8MB. Even at this level, the granularity is sufficient to understand the pedestrian flow.

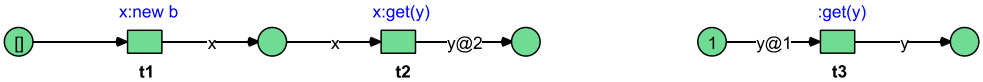
The example here is a 5.0 m grid as described above, but the abstraction level is very important and should be determined carefully. The abstraction level depends on what the model designer wants to know using the model, which in turn determines how granularly the traffic flow needs to be reproduced. It is desirable to reduce the amount of data as much as possible while retaining necessary information, but a significant reduction in data volume should not result in the loss of even necessary information. A balance between the degree of downsizing and the retention of information is important.

2.3 Object Petri Nets

We use a Petri net based tool as the modeling language of the simulation model. Furthermore, the traffic flow model addressed in this dissertation uses object Petri nets that allow replication of instances, since a large number of mobile objects with identical properties are placed in the field and operate concurrently. Object Petri nets are a class of Petri nets in which each token has internal data and methods [21]. We here choose RENEW (Reference Net Workshop) [22]. RENEW is a modeling environment based on object Petri nets. In the standard definition of Petri nets, each token (black dot '●', colorless token) represents existence of something and only the number of tokens is a matter of concern. In object Petri nets, each token is an object or the pointer to an object, where a net structure can also be an object. This enables us to model a situation in which communicating multiple objects behave concurrently in a complex environment. Net instances of RENEW are implemented with Java multithreading, so if strict concurrency is required in the simulation, full parallel processing can be achieved by improving the execution environment (e.g., multi-core processor) [23]. It is a Petri net tool having

object-oriented features. In the modeling by RENEW, instances of a net object can be generated dynamically in the execution. Petri nets can visually represent concurrent systems that operate sequentially while competing for resources. Moreover, timing information such as delay in movement and loading luggage can be expressed in the model. The simulation result is recorded in a log file that is useful for verifying models. We use object Petri nets for all mesoscopic modeling of traffic flow in this dissertation.

Using a RENEW model shown in Fig. 4, we explain how instances of objects are generated and communicated with each other. A RENEW model consists of several net definitions. In this figure, there are two net definitions, Net a and Net b. A place can have token objects, which are '[]' (the same as black dot, colorless token), numbers, strings, and references (pointers) to objects. An instance of a net is generated by 'new' command like in Java. Let us generate an instance of Net a and execute it. Then transition t^1 in Net a fires, an instance of Net b is generated, and the reference to the instance is bound to variable x. Two instances run concurrently. After that, transition t^2 in Net a and transition t^3 in Net b fire synchronously, where inscriptions 'x:get(y)' and ':get(y)' define a synchronous channel. By this firing, a copy of the number object '1' in Net b is sent to Net a, and is bound to variable y. Finally, the rightmost places in both instances have the number object '1'. The transition t^2 in an instance of Net a can fire only when the net instance x has an enabled transition with a label that can be unified with ':get(y)'. In other words, firing of t^2 is suspended until the net instance x has an enabled transition with an appropriate label. This enabled us to introduce control over multiple processes. The number after '@' specifies delay time. If the delay is assigned to an input arc of a transitions, then tokens in the input place of the arc is available after the specified time, and if the delay is assigned to an output arc, then the tokens are put on the output place after the specified time.

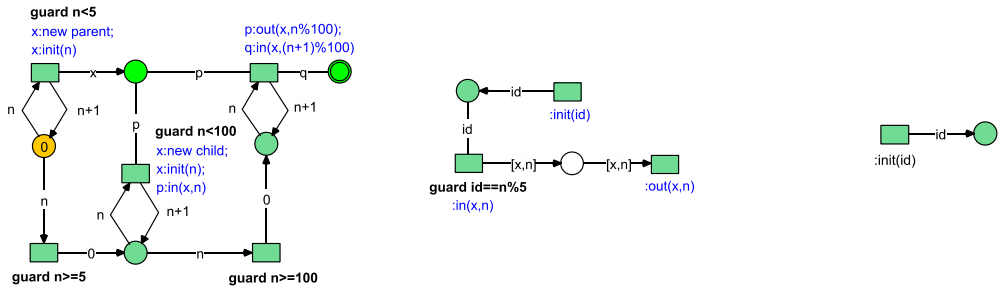


(a) Net a (b) Net b

Fig. 4: An example of RENEW

In this dissertation, we model the properties of RENEW: since all Petri nets are treated as objects in RENEW, instances can be replicated from the base Petri net and placed at any places. Instances themselves act as tokens and can invoke functions from external Petri nets using methods. Therefore, large-scale traffic flow simulation is performed by creating an original Petri net that serves as an agent and generating a large number of replicas of the original Petri net. Since instances are themselves Petri nets, further instances can be input within instances. Therefore, it is possible to create a hierarchical model and run it concurrently.

An example of instance replication and hierarchical petri nets in RENEW is shown in Fig. 5. There is a super-parent Petri net that serves as a field, within which 5 parent-instances are placed as submodels, and 20 child-instances are placed in each of the parent-instances. Each child-instance is assigned a unique ID by the method ‘:init(n)’ when it is created. The parent-instances are connected to each other, and the child-instances are passed to and from each other by the methods ‘:in(x,n)’ and ‘:out(x,n)’.



(a) Net super-parent (b) Net parent (c) Net child

Fig. 5: Sample Petri net of instance creation by RENEW. Petri net super-parent counts up token ‘0’ in the orange place and creates 5 instances parent. It then creates 100 instance children and populates the parent-instance with 20 each. Each child is passed back and forth between parents using the methods ‘:in(x,n)’ and ‘:out(x,n)’.

This mechanism is commonly used in the aircraft passenger boarding model in Chapter 3, the airport surface traffic flow model in Chapter 5, and the airspace traffic flow model in Chapter 7. For example, in the aircraft passenger boarding model, 120 aircraft passenger instances and aircraft seat instances are replicated and arranged to reproduce the movement of passengers in the aircraft cabin. In the airport surface traffic flow model

and the airspace traffic flow model, aircraft instances are replicated and moved around the airport/airspace network configured by RENEW. The airport surface and airspace network is a collection of queue instances, and the aircraft instances move between each queue to reproduce the airport surface movement and flight within the airspace.

2.4 Process Mining Technique

Process Mining is a method/technique for discovering, verifying, and improving the process model by analyzing event logs output from the target system. It is mainly used in order to improve the non-efficient procedures or detects bottlenecks in the business workflow [9]. According to the Process Mining Manifesto [24], its usage type can be divided into three parts: Discovery, Conformance checking and Enhancement. We focused on especially the conformance checking in this dissertation and tried to apply this technique to inspection for discrete-event simulation model. Conformance checking of discrete-event models is covered in Chapter 4.

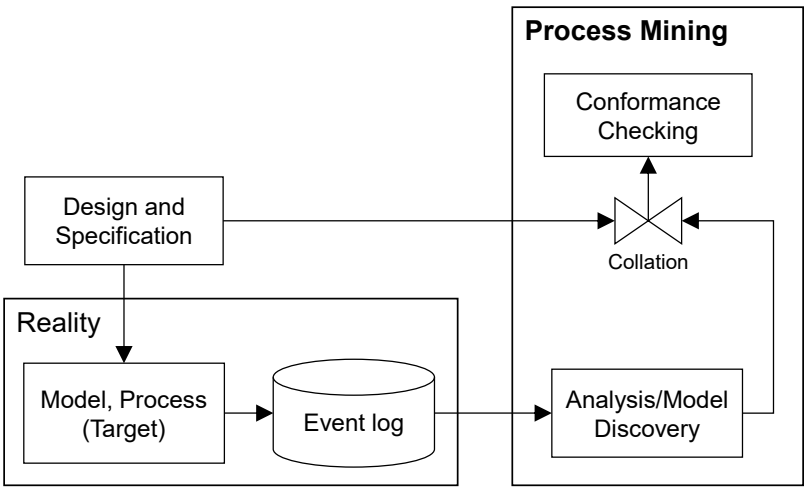


Fig. 6: Process Mining conceptual diagram

In the conformance checking, we can acquire the target model's conformance and reproducibility of the specification by comparing the design and event log. In addition, process mining is further used as an information extraction tool for modeling in this dissertation.Process mining can be used for data cleansing to remove undesired noise

from raw traffic flow data and create clean data for modeling, and for efficiently extracting statistical information. As shown in Fig. 6, the event log output by the model based on the design/specification is analyzed by process mining to discover the model. The analysis results and discovered model are compared with the original design to perform conformance checks.

In this dissertation, we use ProM (Process mining workbench) [25] which has been developed in Einthoven University. The ProM is an open-source software package derived from the early stage of Process Mining research, and it is extensible through several plug-ins that offer a variety of analysis algorithms. ProM is used as a specification-conformance check tool in Chapter 4, as well as for noise removal and statistical information extraction in the creation of the airport surface traffic flow model in Chapter 5 and the airspace traffic flow model in Chapter 7.

Chapter 3

Discrete Event Modeling with Object Petri Nets

In this chapter, we establish a methodology for constructing discrete event models and apply it as a component of the mesoscopic modeling process that will be implemented in subsequent chapters. Here we attempt to model human behavior flow as a basic traffic flow. The flow of passengers in an aircraft, which is an enclosed space, is modeled as a small-scale traffic flow. Since the stagnation and queuing that occur in human flows are the same as the mechanisms that cause traffic congestion in vehicular and other traffic flows, the human flow model created here can be applied to the modeling in the later stages of this dissertation. In the mesoscopic model developed in this dissertation, traffic flow is abstracted by sampling discrete values, and each moving object (agent) operates sequentially and in parallel. Each moving object also operates procedurally according to predefined rules. Based on the characteristics of traffic flow and its discretization, we present a method for building a hierarchical large-scale model by combining a large number of small submodels using object Petri nets.

3.1 Aircraft Boarding Optimization

At airports, it is common to see passengers queuing in front of boarding gates and onboard aircraft. This is due to the time required for processing (loading luggage and completing seating operations) relative to the influx of passengers. In other words, the low service rate relative to the arrival rate of traffic is a factor in queue formation, which can be thought of as a simple queueing model. As a countermeasure, airlines have introduced efficient admission control, known as the *Boarding Strategy*, to alleviate queues. Boarding strategies have been studied as optimization problems, and to date, various

types have been devised and implemented in practice by various airlines.

3.1.1 Related work

Many studies on the subject of aircraft boarding have taken the approach of reproducing passenger movements on a computer and comparing boarding strategies. Therefore, many research results were published in the late 1990s and 2000s, when computer simulation became easier. Marelli et al. (1998) analyzed turn times with a discrete event simulation (PEDS: Passenger Enplane/Deplane Simulation) developed by Boeing [26]. Turn time is the time between an aircraft entering and exiting an airport; the shorter the turn time, the greater the throughput of the airport. The PEDS simulation showed that the boarding time was about 22 minutes, which could be reduced to 17 minutes by using the Outside-in method, which is still one of the most efficient boarding strategies and is used by many airlines today. Some methods have been used to observe actual passenger movements and to obtain statistics; Landeghem et al. (2005) focused on passenger luggage as a factor that negatively affects boarding times. Passenger luggage stowage was analyzed at Brussels airport using video cameras [27]. The results were used in a study by Schultz et al. (2008). Schultz et al. performed simulations using the Asymmetric Simple Exclusion Process (ASEP) method [28]. After comparing various boarding strategies, they showed that the Outside-in method was the fastest and had the smallest variation in time required. This is consistent with the results of a study by Marelli et al. Nyquist and McFadden (2008) showed that among the components of turn time, boarding and disembarking the aircraft is one of the critical paths. However, long turn time is not necessarily caused by passengers boarding and disembarking, and streamlining passenger boarding does not necessarily lead directly to reduced turn times. They stated that shortening boarding time is effective only when turn times are shortened to obtain earlier departure slots [29]. Steiner et al. (2009), on the other hand, stated that shortening boarding time is critical for airlines and presented the idea of reducing boarding time by limiting luggage and making other efficiency improvements before boarding [30].

Many studies have compared boarding strategies and found the Outside-in method to be the best. Steffen (2008), however, found a more efficient sorting order using the Markov Chain Monte Carlo algorithm force-following search [31]. This is a reliable

method because the boarding time of passengers is almost always determined by the order in which they are lined up before entering the cabin. This method, called the *Steffen Method*, can reduce boarding time to one-fifth of the worst-case scenario, but this is only the case when the passengers are lined up ideally [32]. In reality, it is difficult to ensure that all passengers are lined up before boarding.

3.1.2 Key Factors of the Delay

In literature [32], the aisle interference and the seat interference are identified as important factors that cause an increase in the boarding time. Other factors caused by the uncertainty of human behavior such as taking a wrong seat and moving backward on the aisle could be ignored.

- **Aisle interference:** A passenger is stuck in the aisle for loading luggage, preventing another passenger from proceeding to a seat.
- **Seat interference:** A seated passenger hinders another passenger from taking a nearby seat. For example, when a passenger at the aisle-side seat is already seated and another passenger wants to take the window-side seat, then the passenger at the aisle-side seat has to stand up and move to the aisle. As a result, the aisle is blocked.

The above two kinds of interference do not occur if each passenger has enough space on the aisle for passing through other passengers. However, in the congested situation, interference occurs whenever a passenger load luggage or a seated passenger hinders another passenger from taking a seat. In Steffen's simulation, only the aisle interference is considered because the aisle interference is assumed to be the primal issue. It is also stated that considering other interference will make the result more accurate. In this dissertation, we study boarding strategies in congested situations such that the two kinds of interference are important.

3.1.3 Boarding strategy

We often observe congestion of passengers at the boarding gate or in the cabin, even though every passenger is assigned his/her seat and can surly takes it. To relieve the

congestion, many airlines introduce the entrance control called the boarding strategy. In literature, a boarding strategy is represented by the order of passengers or groups of passengers when they enter the aircraft. Fig. 7 shows the boarding strategies proposed so far. Each grid represents a seat, and the numbers of the seats indicate the order of entry.

- **Random:** Passengers board in the random order. This strategy is used for small aircrafts.
- **Outside-in:** Passengers are divided into three groups and board in an order with window-side seats first, the middle seats second, and aisle-side seats last. This strategy is used in several airlines.
- **Back-to-front:** Passengers board in the back to front order.
- **Block back-to-front:** This is a modification of the back-to-front strategy to fit the real situation. Passengers are divided into several groups and board in a back to front order. This is the most widely used strategy in airlines.
- **Reverse pyramid:** This is a hybrid between the back-to-front and the outside-in. Passengers board in a V-like manner.
- **Steffen method:** Passengers are lined up in a prescribed order: from back to front, from window-side to aisle-side, with the adjacent passengers in the line taking every other seat.

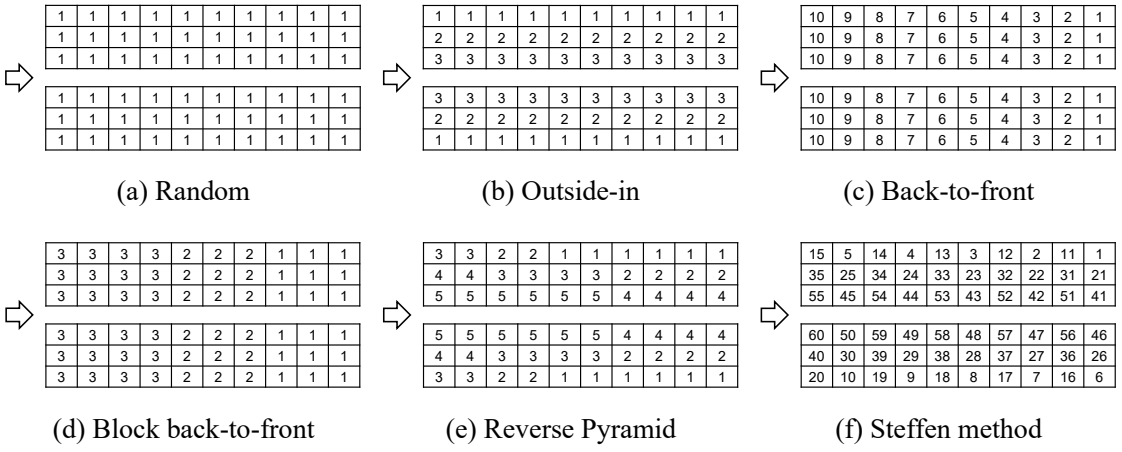


Fig. 7: Boarding strategies: Three seats are arranged in both sides of the aisle. The number indicated by each seat represents the order of entry. The left side indicated by an arrow is the front of the cabin.

3.2 Modeling Aircraft Boarding

When passengers aboard an aircraft, due to the structure of passengers moving in single row along a straight aisle, if the passenger in front stops, all the passengers following must also stop. Therefore, a queue is formed with the same structure as traffic congestion and stagnation in a traffic flow. The mechanism of queue formation can be similarly considered when scaling up to congestion of ground-moving aircraft at airports, in-flight aircraft in congested airspace, etc. In this section, we define seat configurations and passenger behavior patterns in aircraft cabins.

3.2.1 Passenger model design

Each passenger behaves as follows: (i) passing through the gate, (ii) walking on the boarding bridge, (iii) entering the cabin, (iv) walking on the aisle, (v) loading luggage, (vi) taking a seat, and (vii) temporary moving to the aisle for another passenger to take seats (if necessary). Real-life passengers have more complex movements, but if we simplify to the major movements that affect boarding time, we can narrow it down to the above behaviors. From the time they enter the cabin until they are fully seated, they can be in four different states.

- **Walking (WA state):** The passenger enters the cabin through the boarding gate and walks down the hallway until he/she reaches his/her seat. The passenger is constantly moving forward toward his/her seat as long as the area in front of him/her is clear. If the passengers in front of him or her are stopped, he or she stops with some distance between them.
- **Ready to get the seat (GS state):** When a passenger arrives in front of his/her seat, he or she stops to be seated. This state is called the “ready to get the seat” or simply “seat-ready” state, and if there is no obstruction to his/her seat, he/she is seated immediately. No more than two persons in the same row may enter the seat-ready state at the same time.
- **Position change (PC state):** Even if a passenger is ready to get seat, a position change operation will occur if another passenger ahead of him or her is obstructing

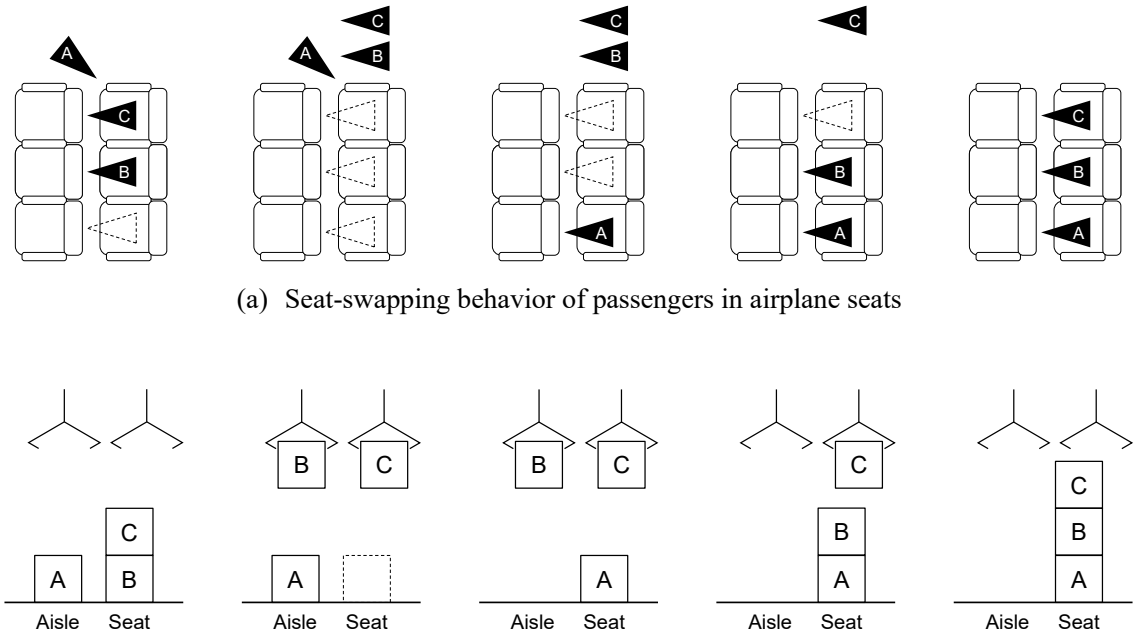
him or her. When a window-side seat passenger arrives in front of his or her seat, if a passenger is seated ahead of him/her in the aisle-side seat, the aisle passenger must exit once into the corridor and swap positions.

- **Seated (SD state):** When a seat is available, if there is no obstruction, the passenger immediately transitions to the seated state. However, passengers in the aisle-side seats may leave their seats to swap their position even after being seated once. Eventually, all passengers are seated.

All passengers are assumed to behave asynchronously and concurrently. We do not allow passengers to get ahead of other passengers on the aisle. In the real situation, some of the passengers form a group such as a family and friends, but we assume all passengers act independently as assumed in Steffen's simulation model. Also, we deal with the case that each flight is full. Although the main reason passengers stop is time required to load luggage [33] [34], since the proportion of luggage varies with destination, season, and customer layer, the time required to load luggage is assumed here to follow a normal distribution, and the emphasis is on reproducing the occurrence of traffic congestion.

The position-change behavior of passengers can be considered similar to the Blocks world which is a planning problem in artificial intelligence [35]. Fig. 8 shows one case of position change in a three-seated coach and its contrast with the Blocks world. The passenger seated ahead in the aircraft seat corresponds to the first stacked block in the Blocks world. In the Blocks world, an arm lifts up an obstructing block in order to move it out of the way of an earlier stack of blocks. This action is the same as the aisle-side seat passenger who is seated ahead of the other passengers going out into the aisle to let the window-side seat passenger who is coming later pass. For example, consider the case where passenger C in the aisle-side seat and passenger B in the middle seat are already seated ahead, and passenger A in the window-side seat comes later. In Block world, there are two arms, and in the initial state, Block B is on top of Block C. In order for passenger A to sit on the window-side seat, passenger B and passenger C need to leave the seat once, let passenger A through to the seat, and then be seated again. In the case of Block world, once B and C are lifted by the arm to place block A at the bottom, then blocks B and C are stacked on top of it. Blocks world can be described as a planning problem for the optimal procedure from the initial state to the goal state. Although there are many possible

event sequences to reach the goal state, in the aircraft boarding model, each passenger is assumed to take the shortest procedure.



(a) Seat-swapping behavior of passengers in airplane seats

(b) Blocks world represents similar planning problems with sets of blocks, table, and arms

Fig. 8: Contrast an example of position-change operation in a three-seated couch with Blocks world in artificial intelligence. In the aircraft boarding model, position-change must be done sequentially as in Blocks world, and each procedure cannot be skipped.

3.2.2 Seat model design

Seat layout of typical aircraft consists of two or three interconnected seats, which are arranged symmetrically across the aisle. Once passengers enter their own seats, they do not enter the other rows, so they do not interfere with each other across rows. Since all seat configurations are uniform, we consider two and three-seated couches to be treated as modules, which are connected by the aisles. Passengers move through the aisle to the row where their seat module is located, and once in front of their seat, they enter the module and do not interfere with the outside. Treating the seat couches as modules has the advantage of simplifying the structure of the model. Each passenger can only enter the seat module in which his/her seat is located and is not allowed to enter any other module.

In the modularized two-seated couch (two-seat module), if the two passengers are assigned a window-side seat and an aisle-side seat respectively, the state transitions for each passenger diverge in two ways, as shown in Fig. 9. If the window-side seat passenger is ahead, there is no position-change operation, and if the aisle-side seat passenger is ahead, a position-change operation occurs. The module has two destination places, and the two passengers are in the seated state (SD) when they reach their respective destination places. Once both passengers are seated, the seat module has no further action.

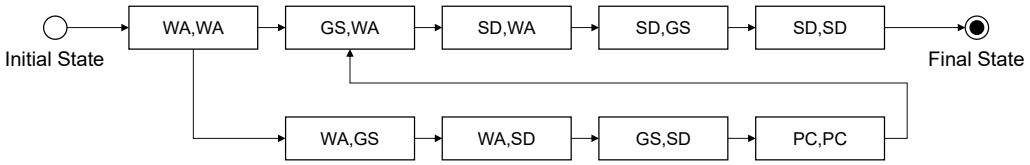


Fig. 9: State transition diagram of the two-seat module where the states of two passengers are indicated in each rectangle. Each passenger takes one of four states: WA (walking), GS (ready to get the seat), SD (seated), PC (position change). Two elements in each state represent window-side, aisle-side in order.

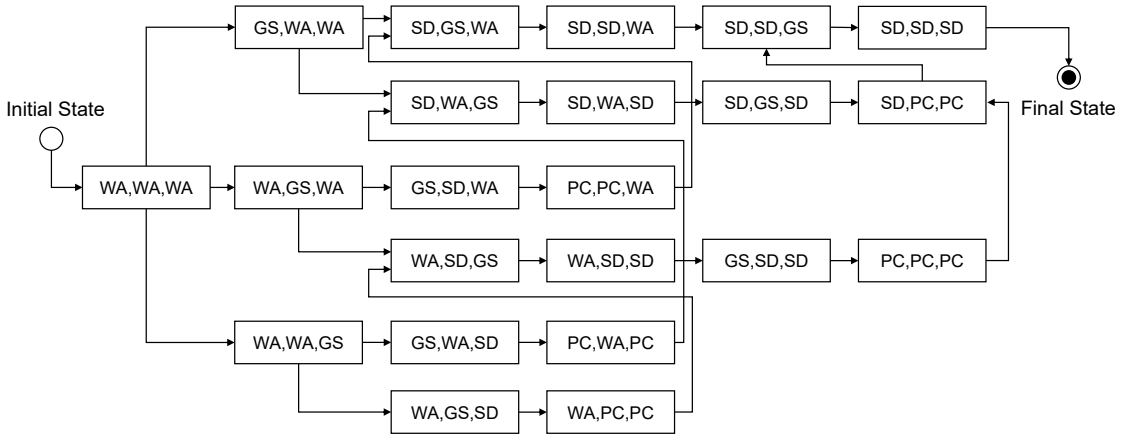


Fig. 10: State transition diagram of the three-seat module where the states of three passengers are indicated in each rectangle. Each passenger takes one of four states: WA (walking), GS (ready to get the seat), SD (seated), PC (position change). Three elements in each state represent window-side, middle, aisle-side in order.

In the modularized three-seated couch (three-seat module), the passenger state transitions (Fig. 10) are more complex and divergent than in the two-seat module. This is because the passenger state transitions can be divided into cases where there is no position-change operation, cases where it occurs once, and cases where it occurs twice,

depending on the order in which the three passengers arrive. The module has three destination places as two-seat module, and the passenger are in seated state when they reach their respective destination places; once all three passengers are seated, the seating module has no further action.

Within each module, as described in 3.2.1, each passenger shall take the shortest steps to get himself/herself seated. When three passengers take seats here, there are $3! = 6$ possible transition patterns from the initial state to the final states. Instances of these three-seat modules are connected with the aisle model and run concurrently.

3.2.3 Cabin model design

The target aircraft has one aisle in the center and 3 + 3 seats are arranged in both sides of the aisle (Fig. 11). The number of rows is 20 or 12. These numbers are the same as those used in the computer simulation [33] and the real experiments [32], respectively. There is no first-class cabin and no priority seating. Such a type of aircrafts is mainly used for domestic flights and the turnaround time is usually strictly limited.

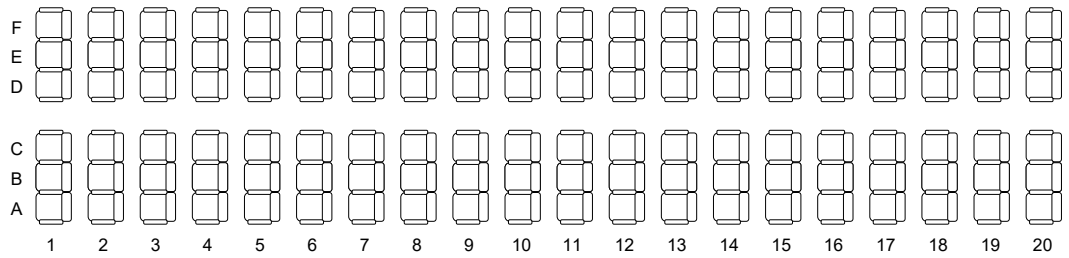


Fig. 11: Seat layout in the cabin. The 3 + 3 seats are arranged in both sides of the aisle. Passengers walk through the center aisle to their seats If the passenger ahead of them is already seated, they perform position-change operation.

3.3 Discrete-Event Simulation

In this section, we model the aircraft cabin and the passenger movement within it, and establish the base methodology for more complex traffic flow modeling. The modeling takes advantage of the characteristics of object Petri nets, and seats and passengers are replicated as instances to reduce the volume of workload. The method developed here will serve as the basis for the later stages of modeling.

3.3.1 Object Petri nets as modeling tool

Since actions by passengers occur sequentially and concurrently, and the behavior of moving objects is identical, modeling in an object-oriented manner is suitable. As mentioned in Chapter 2, we here use object Petri nets tool RENEW. Here we consider a Petri net that replicates a large number of passenger instances to reach their own seat place. In 2.3, we described the replication of moving object instances in the sample model configured in RENEW. In the sample model, the super-parent Petri net replicated and deployed the parent Petri net as instances. In addition, child-instances replicated in large numbers from the child Petri net were operated concurrently between the among parent Petri net. The same concept is used in the aircraft passenger boarding model. As in the super-parent model above, the aircraft cabin Petri net is constructed as a field in which passenger instances move around. The cabin Petri net consists of duplicated instances of the three-seat module, which are connected to form the seating arrangement in the cabin. Although each instance is simple, when a large number of instances are run simultaneously, the model as a whole exhibits complex behavior.

The ideas of efficiency by instance replication and hierarchy by submodels implemented in the aircraft boarding model are also applied to more complex traffic flow models in the later chapters. In this chapter, we show that human flow can be described as a discrete event system by using object Petri nets without missing information such as dwell time and individual travel times.

3.3.2 Aircraft boarding Petri net model

The aircraft boarding model is a hierarchical structure that replicates instances of seats and passengers by object Petri nets, which are then connected. The three-seat module shown in 3.2.2 is configured in Petri nets, and seat instances for each row are replicated based on it. The generated seats are connected to the aisle in the cabin Petri net. Fig. 12 shows the conceptual diagram of the cabin Petri net. There is an aisle in the center of the cabin, and each row has seat instances on each side. Passenger instances with various parameters and methods are also replicated from the passenger Petri net and move around the cabin Petri net. When a passenger instance is generated, it is given a unique ID, its own row and seat number, and baggage loading time. Each parameter has a method for

invocation, and when the method is executed from an external Petri-net, the parameter is returned to the caller. The passenger instance is given a unique ID, its own row and seat number, and baggage storage time when it is created. When a method of the passenger instance is called by an external Petri net, it returns parameters to the caller. Each passenger instance is aligned according to the boarding strategy, and then put in sequentially starting from the first row of the cabin. Passengers check the number of their row against their current location in the aisle. If the current point is in its own row, it advances into the seat instance; otherwise, it advances to the next row and repeats the same process. As passengers enter their seats, they perform the actions described in 3.2.2 within each seat instance, and the seat instances stop operating when all passengers reached their destination place, which means that all passengers are seated. When all the seat instances connected to the cabin Petri net are stopped, all passengers in the aircraft are considered seated and the simulation is complete.

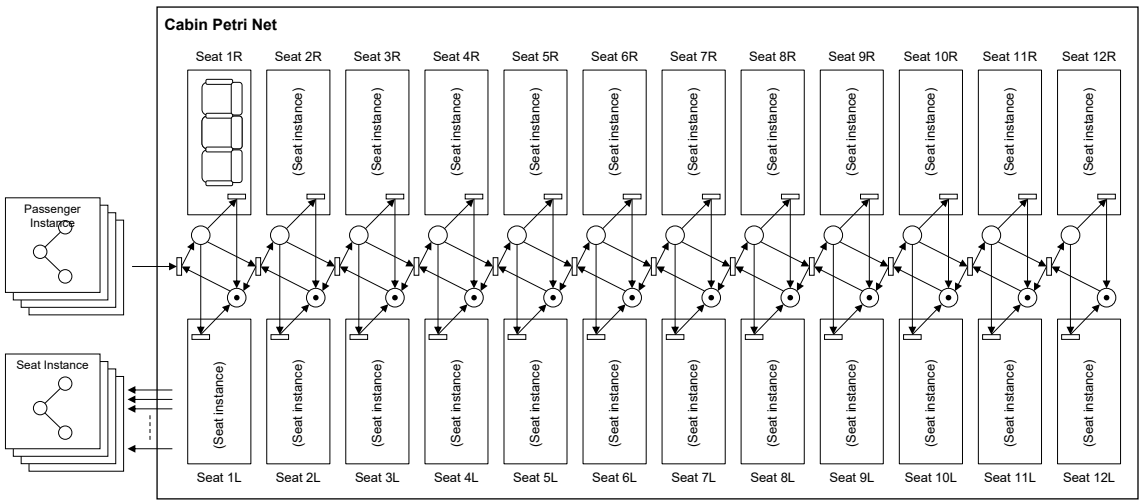


Fig. 12: Conceptual diagram of 12-row cabin Petri net configuration. It is a network of many seat instances connected together, and passenger instances move around on it. The passenger instances themselves are also Petri nets, and each passenger acts as an agent. A token, which is a competing resource, is placed at each connection between an aisle and a seat instance, and when a passenger enters a seat or a back row, he/she consumes this token. While one passenger is consuming a token, other passengers are not allowed to overtake.

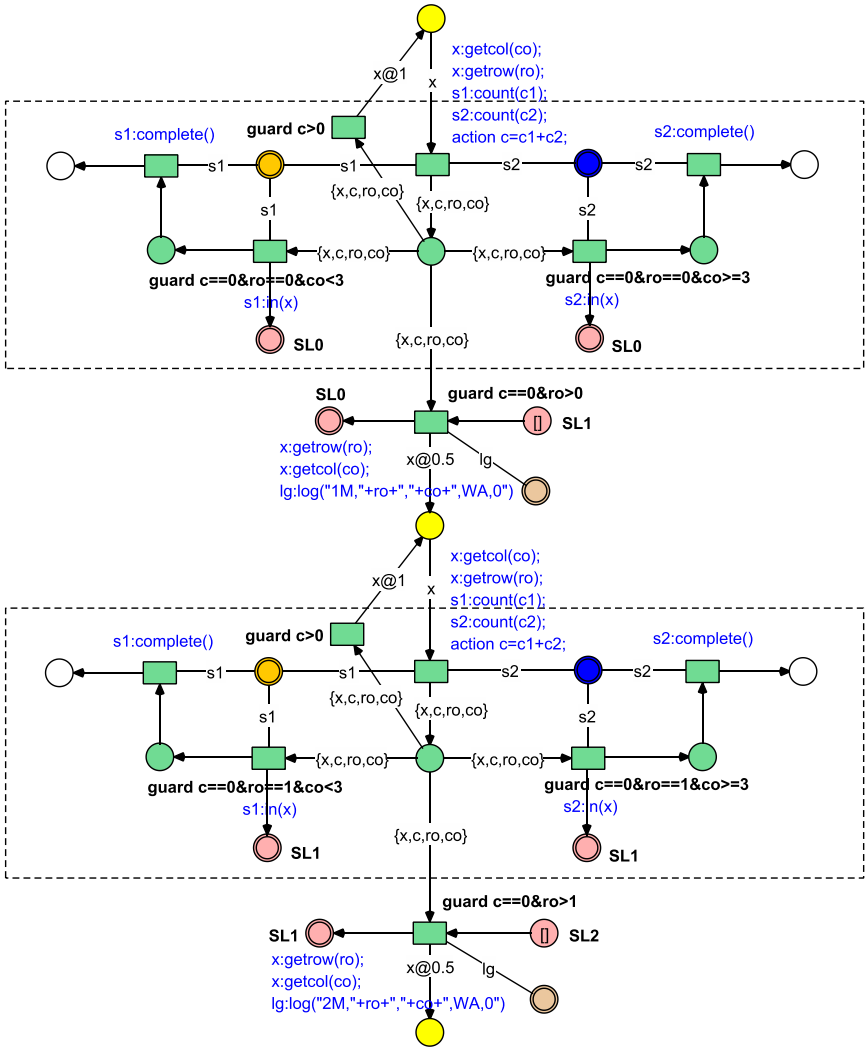


Fig. 13: Actual cabin petri nets with RENEW. Only two rows are excerpted in this figure. A passenger instance provided from the upper place gets its own row and seat number with the methods 'getrow()' and 'getcol()', then checks whether the seat instances located on the left and right at the current location correspond to that row and seat number.

3.4 Model Evaluation

The accuracy of the created model is evaluated by checking the reproducibility of events. Here, the reproducibility of events is confirmed by comparing the throughput of each boarding strategy in the real data and in the simulation results. By checking the relative throughput, it can be confirmed that the efficiency of the boarding strategy is reproduced in the model.

3.4.1 Evaluation strategy

The performance of a boarding strategy is evaluated by the total boarding time. This is the elapsed time between the time when the first passenger enters the gate and the time when all passengers take their seats. Given a boarding strategy, the boarding time can be estimated by the discrete-event simulation [4][5]. Let M be the total number of seats in the aircraft. As shown in Fig. 1, each strategy is represented by a matrix each component of which has a natural number from $I_k := \{1, \dots, k\}, k \leq M$ such that every number of I_k is assigned to at least one component of the matrix. We assume passengers having the same number board randomly. Given such a matrix, the boarding time is measured by the discrete-event simulation. Since the behavior of passengers may have randomness, the deviation of boarding times should be taken into consideration.

Now the boarding optimization problem is formulated as the problem of finding a boarding order that gives the shortest boarding time. As mentioned above, Steffen solves this problem using the Markov Chain Monte Carlo algorithm combined with computer simulation in [31]. The obtained solution is shown in Fig. 7(f). Due to the largeness of the search space, the obtained solution may be local optimum. The solution gives one-fifth of the worst total boarding time. Based on the simulation results, Steffen and Hotchkiss conducted experiments in a mock Boeing 757 and observed a significant reduction in the boarding time [32]. Milne and Kelly propose an improvement of the Steffen method [36]. Their method first assigns passengers to seats so that their luggage is distributed evenly throughout the plane, and after that load passengers according to the Steffen method. In this method, passengers are not allowed to choose seats. The improvement is at most 3.0% compared to the Steffen method. Therefore, we will not use information on the size of luggage in the proposed strategy.

3.4.2 Verification of model behavior

Regarding the logical correctness of the simulation model, we validate the model by the process mining technique [9] [25]. Whether the behavior of the simulation model satisfies the given specification or not is checked by verifying event logs generated by the simulator. The specifications for the passenger model and the cabin model are represented by temporal logic formulas and the model checking technique is applied to event logs.

Through this process, we found several bugs and fixed them [37].

We here validate the quantitative correctness of the simulation model, i.e., whether the boarding times measured by the simulator are similar to those by existing results or not. We compare the proposed simulation model with Steffen's model [31] and the results of his experiments using a real aircraft [32]. The total boarding time is measured by the number of steps (counts) in the simulation and the actual time in the real experiments. Since details of the count are not mentioned in the dissertation, we here focus on the ratio to the random strategy.

The simulation is conducted in the following way. Once all the passengers are on their seats, the simulation is restarted. We execute the simulation up to 50 000 counts. Tab. 1 shows the results. It is observed that the ratios by the simulation models (Steffen's model, the proposed model with 20 rows, and the proposed model with 12 rows) and Steffen's experiments are similar. There is one discrepancy in the results by the simulation and the real experiments. In the real experiments, the back-to-front strategy outperforms the block strategy, whereas the back-to-front strategy gives the worst ratio in the three simulation models. We guess that overtake between passengers in the real experiment occurs and this may increase the performance of the back-to-front strategy.

3.4.3 Simulation Results

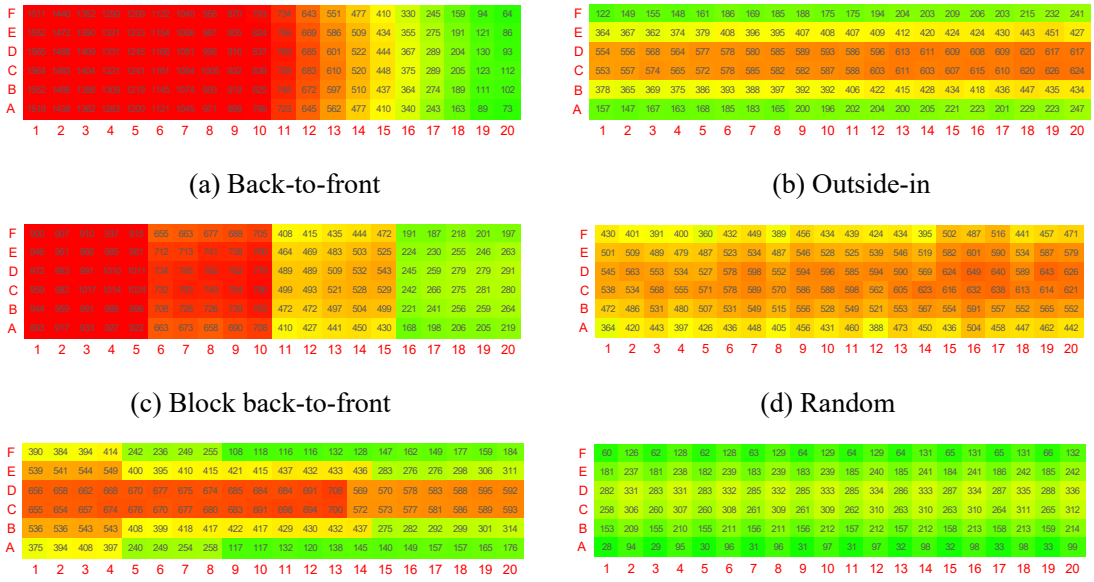
Using the results of the simulation, we analyze existing strategies to know why the Steffen method outperforms other strategies. The simulation results are also depicted in Fig. 14 as heat maps, where the number indicated on each seat position is the average counts necessary for taking the seat. Tab. 1 shows the comparison results of existing strategies. Tab. 2 shows the deviation of the total boarding time by the proposed simulator.

- **Back-to-front:** This strategy gives the worst boarding time. The boarding time increases as the seat number decreases. This is because each passenger can take the seat after all the passengers having seats with larger numbers are seated.
- **Outside-in:** This strategy gives the second-best total boarding time. The deviation in the total boarding time is small.
- **Block back-to-front:** The performance of this strategy approaches to the back-to-

front strategy if the number of blocks increases, and approaches to the random strategy if the number of blocks decreases. The total boarding time is between those of the two strategies.

- **Random strategy:** This strategy shows similar performance as the outside-in strategy. Since this is the strategy without any control, a burden to passengers is the lowest among all strategies.
- **Reverse pyramid:** Although the burden of this strategy is larger than that of the outside-in strategy, the improvement is small. The boarding time of aisle side passengers from No. 1 to No. 13 is relatively higher. During the simulation, congestion occurs on the aisle for these passengers. This is because the space on the aisle available to passengers becomes smaller as the boarding process proceeds. Reverse Pyramid is listed only in the Tab. 2 (not in Tab. 1), since the method was not tested in Steffen's simulation [31].
- **Steffen method:** This strategy gives the smallest boarding time. Compared to the random strategy, the boarding time is reduced to 56 % in the 20-row model, and 71% in the 12-row model. This result suggests that the performance of this strategy increases as the aisle becomes longer. In this method, the order of boarding is a total order and a unique number is assigned to individual passengers. As described in the introduction, this strategy is hard to be implemented in the real situation.

The heat map in Fig. 14 shows that each boarding strategy has a considerable difference in throughput. In particular, the Back-to-Front method used by many major airlines is inefficient, and special methods such as Reverse Pyramid, which is said to be more efficient than existing methods, do not differ that much from the existing Outside-in method. Next, we compare the results with those of previous studies: Tab. 1 shows the simulation results by Steffen (2008) [31] and the experimental data using the mockups by Steffen and Hotchkiss (2012) [32]. Since the simulation has 20 rows and the mockup used in the experiment has 12 rows, we created 20- and 12-row simulation models for comparison here, respectively.



(a) Back-to-front (b) Outside-in

(c) Block back-to-front (d) Random

(e) Reverse Pyramid (f) Steffen Method

Fig. 14: Heat map showing the boarding time of each seat.

Tab. 1: Comparison of existing strategies

	Steffen (2008)		Proposed		Steffen (2012)		Proposed	
Method	Mean [b]	Ratio [c]	Mean [b]	Ratio [c]	Mean [b]	Ratio [c]	Mean [b]	Ratio [c]
Back-to-Front	6276	2.21	3701	1.95	6:11	1.31	794	1.73
Block [a]	4727	1.66	2390	1.26	6:54	1.46	591	1.29
Outside-in	2750	0.97	1740	0.92	4:13	0.89	439	0.96
Steffen	1312	0.46	974	0.51	3:36	0.76	312	0.68
Random	2846	1.00	1897	1.00	4:44	1.00	459	1.00
	(20 rows)		(20 rows [d])		(12 rows)		(12 rows [d])	

a) In the block strategy, blocks of passengers enter from back to front. This type of block strategy is adopted in Steffen's simulation and our simulation for the 20-row model. However, in the Steffen's experiment for the 12-row aircraft mock, passengers are divided into three groups, Front, Center, and Back, and Back group boards first, Front group second, and Center group last. Therefore, the same block strategy is adopted in our simulation for the 12-row model.

b) The unit for the column "Mean" is the number of steps (counts) in the simulation and the actual time in the real experiments.

c) The mean value (average boarding time) is used for computing the ratio to Random strategy.

d) In the 20-row model, the luggage loading time is assumed to be a uniform distribution between 0 and 100. In the 12-row model, we generate the luggage loading time by the probability distribution measured in real situation [38].

Tab. 2: Comparison of deviation

Method	Max [b] (Count)	Min [b] (Count)	Mean [b] (Count)	Median [b] (Count)	σ [b] (Count)	Ratio [c]
Back-to-Front	1452	1171	1308.4	1318	65.5	2.03
Block [a]	1024	734	862.9	865	55.1	1.34
Outside-in	768	527	626.9	628	44.6	0.97
Steffen	430	305	365.6	364	25.8	0.57
Reverse Pyramid	750	538	631.9	631	45.5	0.98
Random	734	541	646.1	652	41.7	1.00

In the simulation shown in this table, we have used 20-row model and generated the luggage loading time by the probability distribution measured in real situation [38].

a) In Block strategy, blocks of passengers enter from back to front as in Steffen's simulation.

b) The unit for each column is the number of steps (counts) in the simulation.

c) The mean value (average boarding time) is used for computing the ratio to Random strategy.

The difference between the simulation comparison (20-row model) and the experimental data comparison (12-row model) is the number of rows and luggage loading time; since Steffen's (2008) [31] model uses a uniform distribution for luggage loading time, the 20-row model we created also generates luggage loading time from a uniform distribution. On the other hand, since the experimental data involve actual human luggage loading actions, the distribution according to Schulz (2018) [38] is used in the 12-row model.

Since simulations are recorded in log files on a step-by-step basis, the time units are different from the experimental data, respectively. Therefore, we focus here on the relative throughput of each boarding strategy ("Ratio" column). Here, the throughput of the Random method is used as a reference and other boarding strategies are compared; in Tab. 1, the worst value in the demonstration is the Block method, but the simulation model created in this study shows that the Back-to-Front method has the worst value. This may be due in part to the fact that the model assumes the worst-case scenario. In the worst-case scenario, the following passengers must always stop when a seat or aisle interference occurs. However, in the demonstration experiment, it is assumed that there were a certain number of passengers who slipped through behind even while the passenger in front was loading luggage. Other results are in good conformance with both simulation and demonstration results, indicating that the model reproducibility is good.

3.5 Summary

This chapter presents the modeling and simulation results of aircraft passengers using object Petri nets. object Petri nets have the advantage of being able to replicate instances of the base Petri net, allowing a large number of passenger and seat instances to be generated and run concurrently as in an agent simulation. The model developed in this chapter is generally conformance with the simulations and demonstrations in previous studies. Furthermore, the model can be considered to be working correctly because it reliably reproduces seat interference and aisle interference, which are the main causes of queue formation.

The object Petri net methodology developed here is used in the later stages of complex modeling, such as airport surface and airspace traffic flow, to simulate the creation of large numbers of instances, each operating independently.

Chapter 4

Process Mining Approach for the Conformance Checking of Discrete-Event Simulation Model

As described in Chapter 1, mesoscopic modeling of traffic flows is a technique for abstracting large volumes of continuous-value data into discrete-event models. Discrete-event simulation is an effective method to reproduce the target system behavior and to investigate its mechanism. In order to grant the validity to the simulation output, the model has to show the compliance with specification and design. However, the test process to prove the validity would impose heavy workload on developers. In this chapter, we introduce a new approach to check the specification-conformance of the simulation model by using Process Mining technique.

4.1 The correctness of discrete event systems

Discrete-event simulation is an effective method to reproduce the time series of events and to investigate its mechanism. It is used for a variety of purposes, such as discovery of the optimal control, or evaluation of system performance. In some cases, however, the validity of simulation model would be treated as prerequisite, and the validity itself is not considered. If the simulation model did not work correctly, the result would naturally be incorrect. Many simulation models are often so unmeasurable and complex that ordinary method for scale like exhaustive search would not be effective. Finding out an abnormality that is deviation from the specification in the numerous paths can impose heavy workload on developers.

The purpose of this study is to confirm that the specification is precisely reflected

in the actual implementation. We attempt to examine the conceptual simulation model by comparing it with the specification. Process mining is generally used for analysis of real systems. Since the method uses only the event log, it can be also applied to the model behavior analysis. This technique enables developers to effectively inspect the system property such as safety, liveness, and fairness [9]. In this chapter, we present a new approach to check the specification-conformance of the simulation model by using Process Mining and attempt to alleviate the developers' workload in the test phase.

4.1.1 Related work

Balci (2004) argued that the modeling and simulation (M&S) validity would be one of the indicators affecting its quality and introduced techniques to assess the quality of large-scale complex M&S [39]. In his study, he also argued that we are unable to claim the level of accuracy for M&S with 100% confidence due to many causes including M&S complexity, reliance on human judgment, qualitative measurements, lack of data, and lack of exhaustive testing. The validation and verification (V&V) for M&S is therefore equal to "confidence building" activity. It is important to specify the set of acceptability criteria for each M&S, and V&V eventually would depend on whether an appraiser can accept the test result. Although a confident level, for example, "95% confident that the M&S satisfies the acceptable criteria", should be provided, it is very difficult to quantify the level. Law (2009) [40] introduced some techniques to build valid and credible M&S with claiming that it is difficult to apply the formal V&V method for various M&S. In his study, the 7-step approach is suggested. The result validation method for comparison of the model and system output data is introduced as its fifth step. The validity of M&S simply means how close the model is to the target actual system. A complex M&S can only be an approximation to the actual system, no matter how much time and cost is spent on model building. Law presented practical techniques to assess the approximation for developing valid and credible M&S. Sargent (2011) [41] also stated there is not any oracle that enables us to find the M&S valid. In his study, four practical approaches, namely, conceptual model validity, model verification, operational validity, and data validity are discussed. He claims that there is no absolute technique and method for each M&S project. Judging the validity of model is affected by an individual perspective. However, it takes

cost and time until sufficient confidence is obtained that a model can be considered valid for its intended application. The relation between model confidence and cost along with value of the model to a user are trade-off. M&S credibility depends on the confidence required in order to use a model and the output data from that model. Raunak and Olsen (2014) [42] followed the argument that simulation models are no different than software that are generally termed as "non-testable" due to the absence of a test oracle, and proposed quantification of V&V activities as an answer to the question "how much validation is adequate?" However, there has been little research to develop V&V related adequacy criteria for this type of software. They mentioned the fifth step result validation of the Law's 7-step approach [40] which Sargent also mentioned as operational validity [41]. For the case of multiple elements to be validated in a M&S and multiple validation techniques, they proposed the method to compute validation coverage v_c $0 \leq v_c \leq 1$ based on whether a specific validation technique for each validatable element has been successfully applied or not by assigning relative weights to validation techniques and elements. If v_c meets the predefined threshold, then enough validation has been performed on the simulation model.

As mentioned above, there is no absolute methodology to decide the validity of M&S, and as Raunak and Olsen [42] claimed, M&S is very similar to the non-testable software type that have non-deterministic or probabilistic element. For such type of software, the statistical approaches on event occurrence can be useful. Our proposed approach can be effective for the non-testable M&S application.

4.2 Process Mining Technique

As mentioned in Chapter 1 and 2, we use Process Mining technique for conformance checking of discrete event systems and information extraction and behavior verification for mesoscopic modeling in this dissertation. In this chapter, we demonstrate the model conformance checking methodology by performing process mining on the event logs from the aircraft passenger boarding model created in Chapter 3. This section presents the properties of models that can be checked by process mining and an overview of the check method, and trace analysis is performed using a simple sample model.

4.2.1 Properties of systems checked by process mining

Process mining is used to check from the event logs whether the properties expected by the designer are exhibited in the model. In this dissertation, the items to be checked using process mining are classified into the following three categories:

(1) **Safety Property:** Every event recorded in the log should be part of the predefined state transition. If any unexpected events or state transitions are observed, it means the target system does not satisfy the specification. In the safety check, we confirm that there are no undesired incidents.
(2) **Liveness Property:** We confirm that events to be executed should eventually be executed. In other words, every event that eventually designed to happen should be observed in each case in the event logs.
(3) **Fairness Property:** If any deviation or unbalanced occurrence probability are observed, the target system may potentially contain some defects. Process Mining can inspect the probability of occurrence of the specific state and event in each trace to detect any unbalanced parts.

4.2.2 Analysis method

The target discrete-event simulation model is developed based on certain specification and design. Any event occurs in this model will be recorded in the event logs. The event log formatted in CSV is converted in XES (eXtensible Event Stream) through plug-in. The log file should include the occurring event, time of occurrence, executor, and event lifecycle, and if necessary, information to supplement the event (e.g., time required between events, event sequence, etc.).

As shown in Fig. 15, we will introduce a test method consist of the two main parts: Event analysis and Trace analysis. Combining these means, we attempt to inspect the properties in 4.2.1. In the Event analysis, we examine the probability of occurrence and co-occurrence, and then confirm the fairness and the safety of each event. On the other hand, in the Trace analysis, we confirm that all the recorded traces are either within the expectation of design and observed uniformly in the event log.

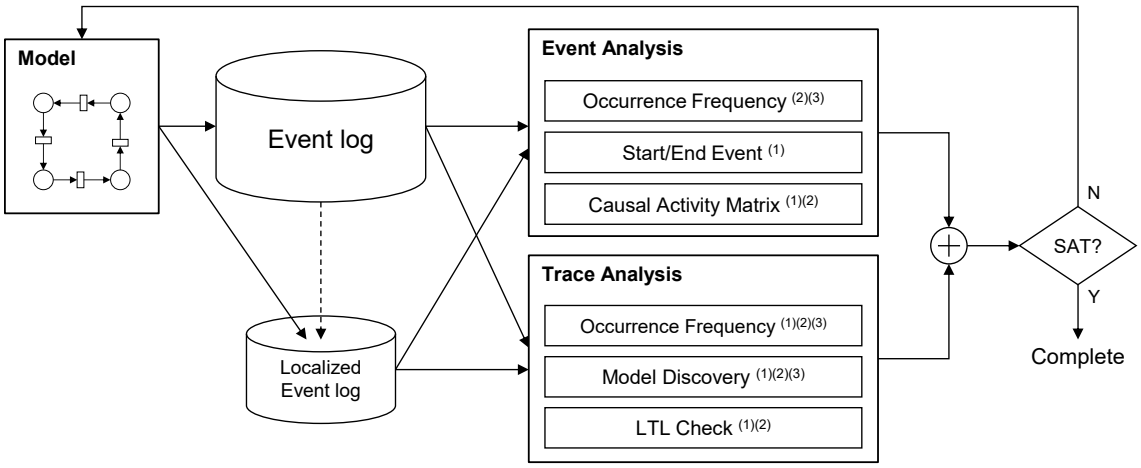
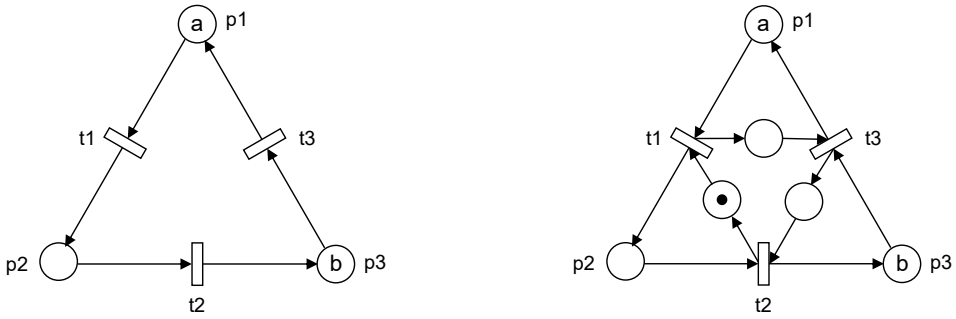


Fig. 15: Conceptual diagram of analysis through Process Mining. The event log output from the model is analyzed to check whether the model is operating correctly. Although the raw event log can be analyzed directly, the analysis can be simplified by an event log that localizes only some of the system's behavior. The superscript numbers in each analysis item correspond to each property mentioned in 4.2.1.

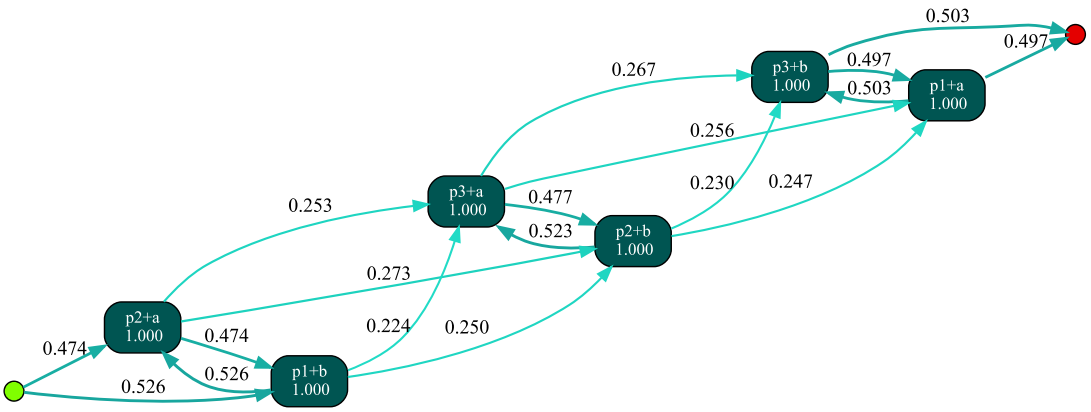
4.3 Process mining on discrete event model

As the test model to show the whole picture of the inspection by Process Mining, we firstly test a simple Petri net shown in Fig. 16. Petri net N_1 has four places forming circle connected with four transitions. In the initial marking, the two tokens a and b are placed in places $p1$ and $p3$, respectively. In the net N_1, each token can move regardless of the other token. Thus, there are multiple patterns of state transitions in N_1. On the other hand, Petri net N_2 has four places and transitions as well as N_1, but N_2 has a shared resource as a token located on the place connected with each transition. Token a and b cannot move simultaneously because of the restriction. In this case, since a is the first to consume the colorless token that serves as a resource, a fires the transition first, and then b and a fire alternately. Therefore, there is only one firing sequence.



Petri net N_1 Petri net N_2

Fig. 16: Sample Petri nets for test. As a sample of the behavior check by process mining, we have two simple Petri net models: N_1 has two colored tokens a and b looping over three places; N_2 has the same looping structure, but a and b alternate with the colorless token as a competing resource. transitions are fired in turn.



(a) Discovered model from N_1

(b) Discovered model from N_2

Fig. 17: Discovered models from event log by Inductive Visual Miner. Since N_1 has no control and only two tokens loop, the transition order of a and b in the discovered model is undefined. On the other hand, N_2 has a restriction on the firing sequence of a and b by the colorless token, so that they always fire alternately. Therefore, the discovered model also shows a straight sequence.

The firing transition records its name along with consume/produce token name into the event log. Fig. 17 shows the discovered models derived from their event logs, mined with ProM and Inductive Visual Miner. In N_2, tokens compete for a single resource, so

the discovered model from N_2 is only one straight way. On the other hand, the discovered model from N_1 shows a branched mesh state transition because N_1 allows each token moves freely. According to these discovered models, Petri Nets N_1 and N_2 are operating as designed and there are no unintended state transitions. If unintended state transitions are included in the reproduced model, then there is something wrong with the implementation of Petri nets, and the model needs to be modified, focusing on the nodes where unnecessary transitions were identified.

4.4 Process mining for aircraft boarding model

Next, we apply Process Mining to the conformance checking for the aircraft boarding model created in Chapter 3. Passengers move around on the Cabin Petri net where three-seat modules are located on both sides across the aisle. Passengers walk through the aisle to look for their seats. If there are no obstacles to seating, the passenger immediately takes a seat, and if the preceding passenger is already seated on the aisle side, the position change operation is performed. The simulation is completed when all of passengers take their seats. We check that passenger behavior in the simulation complies with the above rules and does not deviate from the design.

4.4.1 Event Analysis

Event analysis is an effective way to demonstrate that the target model behaved desirably according to the model design as the specification in Chapter 3. ProM and its plug-ins can provide the frequency of occurrence of each event and the results of analysis of start and end events. As shown in Tab. 3, all the case started from WA state and ended in SD state. This means that for all 3511 cases conducted in the aircraft passenger boarding simulation, all started with the passenger in walking state and ended with seated state.

Furthermore, the frequency of occurrence is also important in the analysis: in Tab. 3, WA state is 72.86%, indicating that most of the states recorded during the simulation are walking. This result is reasonable, as walking is the most common state while passengers are actually in the cabin. For example, the model created in Chapter 3 simulates a 20-row aircraft cabin. If a passenger has seat row i, WA state is recorded

every time the passenger passes through the aisle until row $i-1$, and the transition to GS state finally occurs when the passenger reaches row i. In other words, when no position change occurs, WA is one time, GS is one time, and SD is i times. Therefore, the number of WA states that appear in the log will be large. Since all passenger instances are generated in WA state, all simulations start with WA of one of the passengers. If it starts on any other event, it means that there is a bug in the passenger Petri net. The end event is more important than the start event. The termination condition of a simulation is when all passengers are in SD state. Therefore, the fact that all 3511 simulations ended with SD means that all passengers were successfully seated.

Tab. 3: Event analysis result. The number of start and end events represents the number of cases and is the same as the number of simulations. This table shows 3511 simulations, all of which started with WA event and ended with SD event.

	WA	GS	SD	PC
Count	4845180	796965	632570	375645
(%)	72.86%	11.98%	9.51%	5.65%
Start Event	3511	0	0	0
(%)	100.0%	0%	0%	0%
End Event	0	0	3511	0
(%)	0%	0%	100.0%	0%

The causal activity matrix enables to examine the causal relationship of each event and confirm that the state transitions are taking place as designed. Tab. 4 shows the Causal activity matrix created by extracting the logs of window seat passengers and aisle-side seat passengers from the event logs and the causal relationship between the events that appeared in each localized log. According to the definition of Causal activity matrix [43], a value close to 1.0 indicates that a causal relationship definitely exists between two events; a value close to -1.0 indicates that there is no relationship; a value close to 0 indicates that the relationship is uncertain and there is no evidence for it. Let x and y be two events in the event log. If x directly precedes y in all state transitions, the from x to y relationship value is 1.0. In other words, since y can be regarded as never preceding x, the from y to x relationship value is -1.0. On the other hand, if the $x \rightarrow y$ state transition does not appear in the event log, the value is 0 because there is no evidence

to negate the relationship but also no evidence to establish the relationship.

Comparing (a) window-side seats and (b) aisle-side seats in Tab. 4, we find a difference in the causal relationship between PC and SD states. This is because window-side seat passengers do not change their SD state once they are seated. On the other hand, aisle-side seat passengers have to leave their seats and enter PC state again due to the position change operation even if they once entered SD state. The value is set to 1.0 because SD→PC state transition appeared in the event log, which provided evidence of the relationship between PC and SD.

Tab. 4: Causal activity matrix. The directly follows relations between events are analyzed by ProM and Discover Matrix plug-in. Here, only the passenger logs for window-side and aisle-side seats are extracted and then analyzed, respectively. Items with differences between (a) and (b) are underlined.

(a) Window-side seat passengers

	GS	PC	SD	WA
GS	0.0	-1.0	1.0	-1.0
PC	-1.0	0.0	*0.0*	0.0
SD	-1.0	*0.0*	0.0	0.0
WA	1.0	0.0	0.0	1.0

(b) Aisle-side seat passengers

	GS	PC	SD	WA
GS	0.0	-1.0	1.0	-1.0
PC	-1.0	0.0	*-1.0*	0.0
SD	-1.0	*1.0*	0.0	0.0
WA	1.0	0.0	0.0	1.0

As described above, event analysis enables confirmation that there is no unintended occurrence of events by examining the frequency of occurrence of events and their relationships. If an imbalance in the frequency of events or unintended state transitions (causal relationships among events) are observed, then there are some potential errors in the model, and the errors need to be corrected. By referring to the results of process mining, abnormalities can be identified quickly.

4.4.2 Trace Analysis

Trace analysis offers us the detailed information of state transition with visualized concrete image. ProM and its plug-in can visualize the model output with mathematical modeling languages (e.g., Petri Net, Workflow-Net) [9]. The visualized graph could virtually be utilized as a kind of demonstration due to its image size. We can visually recognize the conformance of state transition only in case the trace divergence is scalable.

In other words, if the target system has a certain level of complexity, the divergence of traces would explosively increase, and it would be extremely difficult to detect undesirable traces by checking with our own eyes. Fig. 18 shows a visualized image of the event log output from the sample model by ProM. Although the sample model is not so large, even such scale can have quite large state divergence more than we could visually recognize, and the ProM also output the quite complex visualized image. We here attempt to localize the range of analysis for the large system, and finally confirm the conformance by stacking these small results.

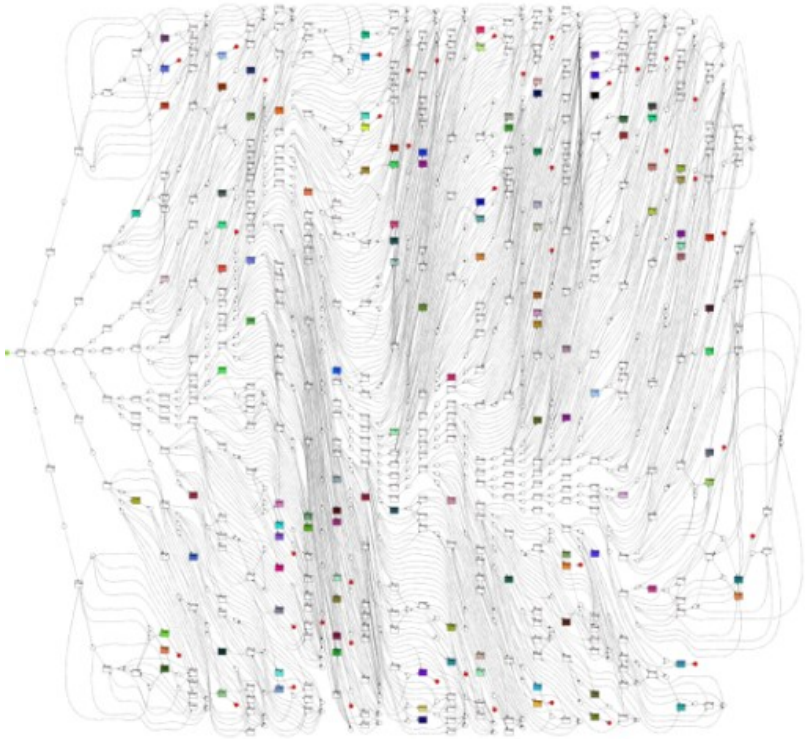


Fig. 18: Explosively complex discovered model. Since the movements of all passengers in the event log, the discovered model directly from the raw data may not be useful for analysis. In order to reduce the discovered model to a level that can be visually confirmed by humans, it is necessary to localize the event log.

In the early stage of model development, the model should be designed to be an aggregation of modules that are functionally separated. Modules should also be independent of each other, and their independences contribute to reduction of the inspection volume. As mentioned in Chapter 3, the aircraft passenger boarding model is

designed so that each seat instance is completely independent and does not interfere with each other. Also, all seat instances are replicated from the same Petri net. Therefore, by checking the seat module alone and showing that the behavior is as designed, we can guarantee that the behavior is correct after all passengers have entered the seat instance.

Focusing on the seat instance, we can ensure that at least the minimum component of the model works as intended. In this case, if the test of the seat module is firstly completed, we can omit the test items for the other seats. Since the other seats are just duplicate of the module, the passenger's behavior inside is assured. Fig. 19 shows the analysis result of independent test of the module by ProM. There are only six traces, and the proportion of their appearances is approximately equal. The behavior of the passengers in the module is determined by the order in which they enter the room: there are six combinations of three passengers entering the room, which is exactly the same as the number of branches in Fig. 19, indicating that the state transitions are as intended. If the appearance rate is biased here, or if an unexpected number of branches is observed, it suggests that there is some error in the module.

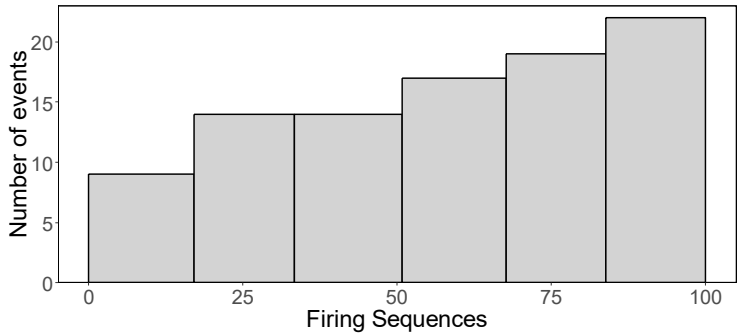


Fig. 19: Firing sequences of the three-seat module. The x-axis shows the proportion of occurrences of each trace in the simulation, and the y-axis shows the number of events that make up the firing sequence for that trace. since the firing sequence changes depending on the order in which the three passengers arrive, a total of six different traces are identified. Each trace appears with approximately the same frequency.

Fig. 20 shows the discovered model of the three-seat module. Compared to Fig. 18, the size of the model is much smaller, and the state transitions can be visually confirmed with a model of this size. Fig. 21 is the discovered model of the three-seat module limited to aisle passengers only. In the discovered model, after repeating WA state, it transitions

to GS state and then to SD state; if a position change still occurs after the passenger enters SD state, it transitions to PC state and then again from GS state to SD state. Fig. 21 indicates that the state transitions are as designed and that there is no deviation from the design.

In this way, instead of discovering the model directly from the raw event log, the size of the log is reduced to a level where the state transitions can be visually confirmed, and it is confirmed that there are no abnormalities or deviations in the behavior.

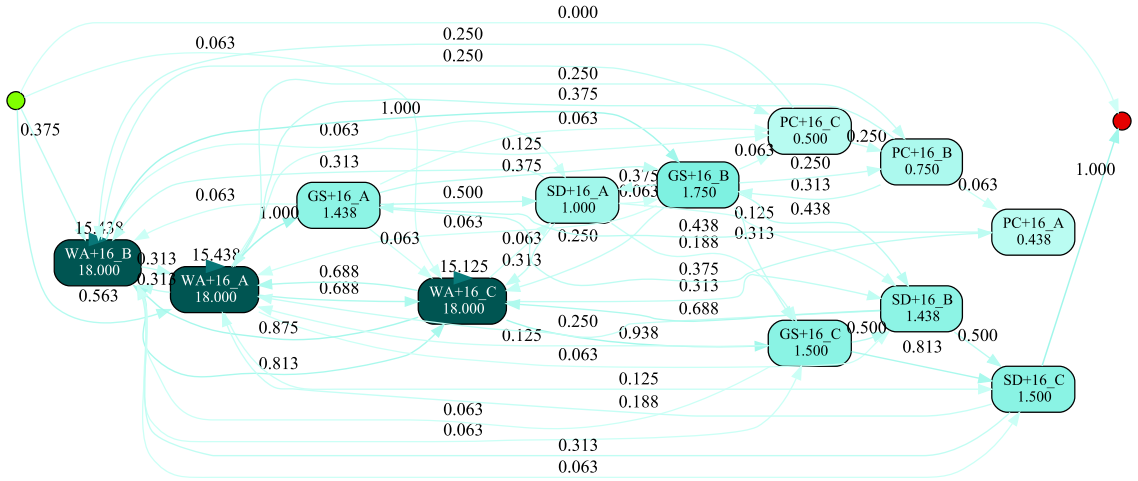


Fig. 20: Discovered model of the three-seat module. The numbers at each edge indicate the relative occurrence; the model size is much smaller than in Fig. 18 because the logs are restricted to the three-seat module.

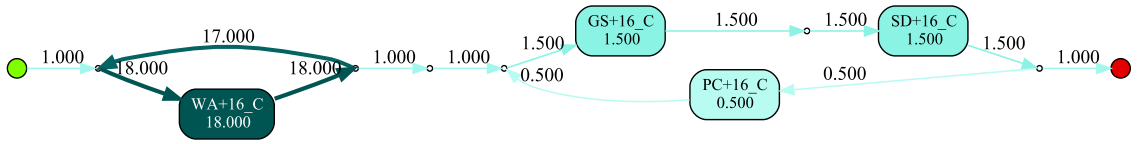


Fig. 21: Discovered model of an aisle-side seat passenger. Only the logs of aisle-side seat passengers are further extracted from the data of the three-seat module to reduce the model size. Localizing the logs to this level further simplifies the state transitions and makes them easier to check.

4.4.3 Small-scale connection check of modules

Next, we conduct an additional trace analysis by connecting several three-seat modules in series. Since the full cabin model consists of 40 modules and 120 passenger instances, the amount of analysis becomes large when the entire model is checked. In order to check

the joints between modules, we consider the seat arrangement with three rows of three seats on either side of an aisle. The first step is to perform the trace analysis focusing on the movement of each passenger. Fig. 22 shows the inspection result of the downsized event log by extracting each passenger's behavior. Interestingly the result was quite clarified thanks to the size reduction. Each passenger's behaviors are at most a few ways. A passenger walks through the aisle until reaching the seat, and then he is ready to get a seat. If a passenger is in a window-side seat (seat A or F in Fig. 22), the pattern can be divided into two cases: with or without a preceding passenger. If there is no preceding passenger, the passenger in the window seat is seated and no position changing occurs. On the other hand, if the middle or aisle-side seats are seated ahead of the others, a position-change operation will occur. This means that the behavior pattern is only two ways.

Let us also consider the case where a passenger is seated in an aisle-side seat (seat C or D in Fig. 22). If an aisle-side seat passenger is seated ahead of the other passengers, but a middle or window-side seat passenger arrives later, he/she has to get up from their seat. The pattern can be divided into three cases: no position change occurs, one position change occurs (one of the window-side or center-side seats is seated ahead of the other, but the other passenger arrives later), and two position changes occur (there is no preceding passenger, and the window-side and middle seat passengers arrive later and change their positions, respectively). Fig. 22 shows that all traces have desired results in both occurrence rate and firing sequence. If this analysis reveals an anomaly in the frequency of occurrence of some traces, or if the firing sequence is not the expected number of events, it means that the Petri net contains some errors. In this way, instead of analyzing the entire event log directly, the behavior of the model can be verified by isolating and analyzing the necessary parts.

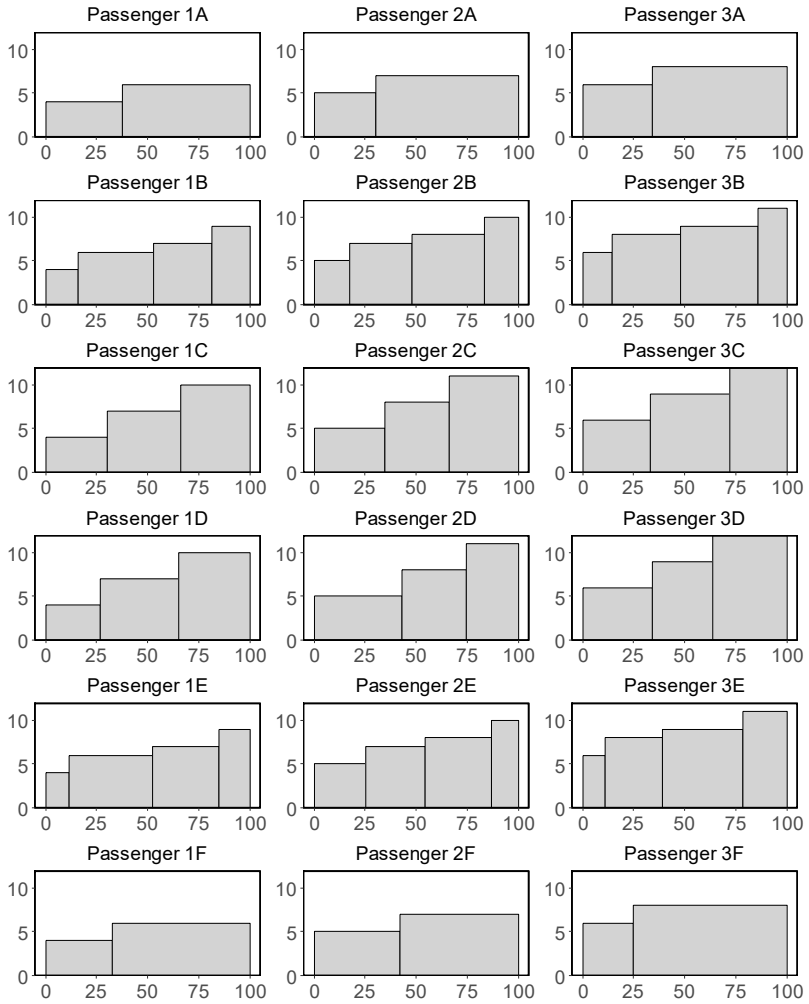


Fig. 22: Occurrence ratio of passengers' behavior (3-seat×2-column×3-row dimension). Fig. 19 shows the trace of the entire three-seat module, but here the trace analysis results for each passenger are shown. For example, for passenger 1A in the window-side seat, approximately 30% of the firing sequences with 4 events and 70% of the firing sequences with 6 events appear. The difference is the presence or absence of the position-change operation. The ratio of the number of events for passenger 1A in the window-side seat is the same as the ratio of events for aisle-side or middle seat, which is a 1:2 relationship.

4.4.4 Behavior conformance by LTL Checker

As mentioned in 4.2.1, we confirm that the simulation does not go into any unexpected state and all the traces are included in the predefined sets. However, although we know that passengers are safely seated, that they have not entered an abnormal state, and that no abnormal firing sequences have appeared, we do not know whether the rules that the designer wants them to comply are being complied.

For example, in the aircraft passenger boarding model, the simulation is completed when all passenger instances are seated (in SD state). If there is an occurrence of an overtaking action or neglect of a position change operation in the middle of the simulation, it is difficult to ascertain them only by event analysis or model discovery from traces. Although each instance in the model performs only simple actions, they are executed concurrently in large numbers. Rule compliance checks should be performed to ensure that passengers are not overtaking or that no passengers are "cheating" in the position-change operation. In this dissertation, we use Linear Temporal Logic (LTL) to describe the rules we want the model to comply and check whether they are actually complied in the simulation.

ProM offers LTL Checker plug-in [44] to check such pre-defined properties and detect errors in characteristic of the model considering the real world. LTL [45] is a modal temporal logic and used as a mean to describe the required specification in the model checking. LTL can offer satisfaction of condition described in LTL formula in the specific trace where the state will change as time flows. LTL formula is built up from atomic proposition $p_1, p_2, \cdots, p_n$, logical operators and temporal operators. Logical operators are usual truth functional connectives in logic. Temporal operators are also connectives that enable to describe properties of state transition.

$$\phi ::= true|false|p_1 \wedge p_2|p_1 \vee p_2|\neg p_1|p_1 \rightarrow \phi_1|\mathrm{G}\phi_1|\mathrm{F}\phi_1|\mathrm{N}\phi_1|\phi_1 \mathrm{U} \phi_2$$

Let ϕ be a set of propositions, then $p_1, p_2, \cdots, p_n$ are LTL expressions if $\{p_1, p_2, \cdots, p_n\} \in \phi$. When p_1 and p_2 are LTL expressions, $\neg p_1$, $p_1 \wedge p_2$, $p_1 \vee p_2$, $\neg p_1$, $p_1 \rightarrow p_2$, $\mathrm{G}p_1$, $\mathrm{F}\phi_1$, $\mathrm{N}\phi_1$, $p_1 \mathrm{U} p_2$ are also LTL expressions. An explanation of the temporal modal operators is given in Tab. 5.

Tab. 5: Temporal modal operators in ProM LTL Checker

Operator	Symbol on ProM	Meaning	Description
G	[]	Globally	For proposition p_1, $\mathrm{G}p_1$ implies that p_1 always holds.
F	<>	Future	For proposition p_1, $\mathrm{F}p_1$ implies that p_1 will eventually hold at some future time.
N	_O	Next	For proposition p_1, $\mathrm{N}p_1$ implies that p_1 has to hold at the next state.
U	U	Until	For propositions p_1 and p_2, $p_1 \mathrm{U} p_2$ implies that p_1 holds until p_2 becomes true at some future time.

The LTL Checker plug-in of ProM offers preset LTL formula editable on GUI and import function for an external LTL file. Instructions for using the LTL checker can be found in the reference manual [46]. We use LTL Checker for mainly the following purposes:

- **Validity check for event log:** The event logs output by the simulation model are not simply for simulation purposes only, but can be used for various purposes, such as identifying congestion points and improving traffic flow by extracting statistical information. Therefore, at least in the event log, we need to ensure that the model complied with the rules desired by the designer. The LTL check allows the designer to confirm both properties: that what should not happen by design does not actually happen (safety) and that what the designer wants to happen eventually happen (liveness).
- **Bug detection:** If a bug is latent in the model and traces containing a rule violation exist in the event log, LTL check can detect them in a short time. However, since error traces may appear with extremely low frequency, a fairness check (checking the frequency of occurrence) is essential in trace analysis as a preliminary step to LTL check.

In the case of the aircraft passenger boarding model, we can verify that each instance is not behaving improperly in the model by inspecting two rules: (i) there are no unexpected state transitions in the passenger alone, and (ii) there are no improper overtaking behaviors for all combinations of two passenger instances. The model can be checked to ensure that each instance is not behaving improperly in the model. For rule (i), we confirm that all passenger instances can take only four states (WA, GS, PC and SD), from the results of Tab. 3, and the causal activity matrix (Tab. 4) shows that the state transitions are normal as designed. However, when associating the occurrence events with the executors and distinguishing each event one by one, the causal activity matrix cannot be applied due to its huge size.

Therefore, we use the LTL formula to describe the rules to be complied and to check to ensure that there are no violations in each trace. Let us consider a k-row passenger pair k_i and k_j $(1 \leq \{i, j\} \leq 20)$ in the aircraft passenger boarding model: if k_i

becomes GS state, k_j is not allowed to interrupt and enter GS state at the same time. By describing this rule in LTL and checking compliance for all passenger pairs, we are able to confirm that there are no inappropriate interruptions at the entrance of a seat. In the rule (ii), by checking that no overtaking behavior has occurred in passenger pair (k_i, k_j), we can confirm that no irregular overtaking has occurred during the simulation time. Note that the number of LTL formula can be reduced by targeting only those k_j that are in WA state between k_i entering the aircraft Petri net and becoming SD state, but in this case, LTL checks are performed for all passenger combinations.

- **Two passengers are in the same row of seats:** If k-th row ($k \in 1, ... ,20$) passenger (k_i) passed the first row then another passenger (k_j) followed, which means passenger (k_i) passes firstly the row (0) and goes into WA state, the preceding passenger (k_i) will also pass row (k) and goes into GS state, then the following passenger (k_j) will pass row (k) and goes into GS state. Since the ProM LTL checker inspects each trace, the safety property can be checked by indicating the absence of error traces. The detection of error traces can be expressed by F (Future) operator, this rule is therefore expressed in the LTL Checker format as:
  $F\Big((person = k_i \wedge activity = WA) \wedge F(person = k_j \wedge activity = WA)\Big) \rightarrow$
  $F\Big((person = k_i \wedge activity = GS) \wedge F(person = k_j \wedge activity = GS)\Big)$.

- **Two passengers are in the different row of seats:** If two passengers are in different seating rows, the order in which they enter the aircraft should be considered. Here we consider the case where the passenger in the front row is ahead and the passenger in the back row enters later. If the passenger ($k + l$) passed the first row and goes into WA state, then another passenger (k) passed the first row and goes into WA state, the preceding passenger ($k + l$) will pass row (k) and thereafter the following passenger (k) will pass row (k). In this case, unlike the case where passengers are in the same row, passenger ($k + l$) does not enter GS state when it passes through row (k). Therefore, it is not known which passenger ($k + l$) or passenger (k) would enter GS state first. Therefore, the LTL Checker format is used here, and the firing transition is specified by event. Let t_k denote the transition in front of the seating row, the LTL formula can be described as follows:

$$F\left((person = (k+l)_i \wedge activity = WA) \wedge F(person = k_j \wedge activity = WA)\right)$$
$$\rightarrow F\left((person = (k+l)_i \wedge event = t_k) \wedge F(person = k_j \wedge event = t_k)\right)$$

Here is an example of a rule violation in a trace and its detection by the LTL formula. Case 1 and Case 2 in Fig. 23 show the state transitions without position change operation. In Case 1, k_j is seated after k_i is seated, and they do not enter GS state at the same time. On the other hand, in Case 2, k_j transitions from WA state to GS state before k_i transitions from GS state to SD state, resulting in both being in GS state. Therefore, Case 2 is detected as a rule violation. Cases 3 and 4 show the state transitions with position change operation. In Case 3, initially k_j in SD state leaves the seat again and enters PC state at the same time that k_i arrives and enters GS state. After k_i enters SD state, k_j also transitions to GS and SD states. On the other hand, in Case 4, k_j transitions from PC state to GS state immediately after k_j leaves the seat to change their positions, resulting in k_i and k_j entering GS state at the same time. Therefore, Case 4 is detected as the rule violation.

Especially in Case 4, there is no abnormality in the behavior except that two persons entered GS state at the same time, and it is difficult to detect a rule violation by looking only at events and traces. k_j's movement alone shows that he left SD state for position change operation, and then transitioned PC→GS→SD. This is normal behavior when viewed as a stand-alone passenger Petri net. In other words, this violation cannot be detected by looking only at the movement of individual passengers. It is also difficult to detect the rule violations in trace analysis or model discovery. Compliance with the rules can be checked by describing each rule in LTL formula.

The models created in this dissertation run sequentially and concurrently by object Petri nets. Furthermore, since the model is composed of many subnets combined, it is difficult to check for the presence of rule violations in the huge event logs. Therefore, we focus on passenger pairs only and describe rules in LTL. By performing LTL checks on all pairs of passengers, rule compliance can be ensured for all behaviors performed on the model.

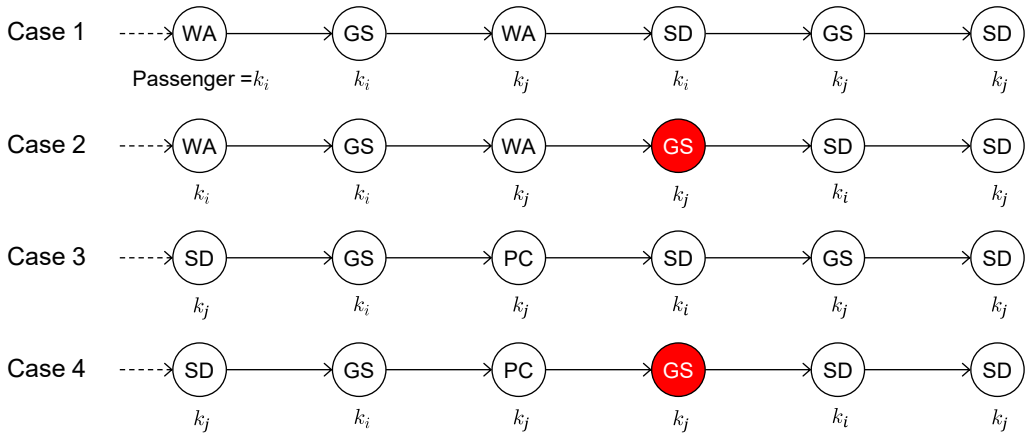


Fig. 23: Sample of rule violations detection in tracing. Two passengers in k-th row are extracted from the event log and arranged in order of execution. Rule violation occurs in the red circle in Case 2 and Case 4.

4.5 Summary

While correct model behavior is a prerequisite for simulation, it is difficult to build the model that has a mechanism for verifying its own correctness. It is necessary to incorporate verification methods in the design phase. If the correctness of the behavior can be confirmed from the event log, which is the output of the model, the time and effort required to build a mechanism for inspection can be greatly reduced. Although a simple comparison of event log information and actual data may confirm the reproducibility of the model, it is difficult to accurately determine the degree to which it reproduces reality. Even if the reproducibility is high, the model is inadequate if it contains errors.

In this chapter, we introduced a novel test method by Process Mining using only event log. This method offers several means to confirm the correctness of the target model from many aspects. Since the event log is finite, Process Mining cannot prove the thorough and permanent correctness of the model. However, we can acquire at least the validated logs that are applicable for subsequent analysis or statistics. This method will exert its strength in case of the black box model. Especially it can be used for debugging in the poor environment where no debuggers or similar functions are provided. Furthermore, since this method does not impose the systems to append the extra mechanism for test, it is expected to considerably reduce the developers' workload.

Chapter 5

A Method for Mesoscopic Modeling of Large Volume Traffic Flow Applying Process Mining Techniques

In Chapter 3, we developed a modeling methodology that considers traffic flow as a discrete event system, and in Chapter 4, we presented a conformance check method to ensure that the created model is working properly. The modeling framework developed in this dissertation analyzes traffic flow data, which is the source of modeling, and graphs it as a discrete event system. Model creation from these traffic flow data can be used for various purposes but handling large-volume traffic flow data requires huge computing power and a great deal of work. To mitigate this problem, we propose mesoscopic models in which continuous values are replaced with statistical information derived from reduced data by discretization while retaining the model abstraction level that allows for bottleneck verification and identification of stagnation. In this chapter, a novel model creation method that reduces the workload by applying process mining techniques are presented. Using airport traffic flow data as an example, we create an actual model and show that process mining techniques are quite useful in the modeling process.

5.1 Using Process Mining for Mesoscopic Modeling

Highly accurate, high-frequency, large-capacity traffic flow data are available to the public. As traffic flow data has become more accurate and frequent, handling it directly requires huge storage and computing power. In addition, if the data granularity is too high, then it becomes difficult to see the events that should be checked in the model [8]. Therefore, depending on the purpose of modeling, data abstraction is necessary. Here, we

consider mesoscopic modeling approach that replaces continuous values with statistical information obtained by discretizing them. By this approach, it is possible to reduce the data size while retaining the model abstraction level that allows further analysis.

There are two challenges in creating a mesoscopic model. The first is that it requires time and effort to discretize the traffic flow and extract features. Secondly, the previous model would no longer be useful in case of the environmental change that affects the traffic flow. Each time the environment changes, the same amount of effort is required to rebuild the model from initial data acquisition. The model needs to be maintained by persons who have the necessary knowledge and skills. Therefore, we attempt to solve the above problems by applying process mining technology to the model creation. We have already used process mining techniques for conformance checking in Chapter 4, and these techniques are also useful in extracting the information needed for mesoscopic modeling.

By applying this technique to mesoscopic modeling of traffic flows, we propose an efficient method for identifying traffic patterns, removing noise, and extracting statistical information. Furthermore, by creating a mesoscopic model from actual airport surface traffic flow data [47], we show that the model created by process mining is useful for traffic flow analysis. The mesoscopic modeling approach applying process mining techniques is outlined as follows. First, an abstracted route graph is defined to summarize traffic flow data. Next, based on the graph, continuous data is converted into discrete events, and an event log is generated. Finally, process mining is performed using the event logs to extract statistical information needed for modeling. In this chapter, we actually create a model of the airport surface traffic flow and evaluate the accuracy of the model.

5.2 Modeling Overview

5.2.1 Application of Process Mining to Traffic Flow Data

Process mining is a technique for discovering bottlenecks and inefficient procedures inherent in a system through the analysis of recorded events and traces based on the event logs output from the system [9]. It is mainly used in business process analysis to analyze workflows [48] and improve inefficient procedures, and it can also be used for checking

whether the system behaves as expected and conforms to specifications [8], but there are currently almost no examples of this being used for traffic flow analysis. This is because process mining is suitable for events of a nature that can record events in a discrete manner. However, there are some studies similar to traffic flow data; traffic forecasting between network nodes [49] and TCP/IP logs analyzed by process mining and used for protocol mining [50]. Process mining was also applied to rail traffic [51]. The study applied protocol mining to log files obtained from the operating system to measure the operation performance, and also attempts to re-adjust the block time and track section of each vehicle. The powerful analysis techniques described above can be used in traffic flow analysis by abstracting the data to the scope of process mining.

5.2.2 Process Flow of Mesoscopic Modeling

Process mining requires a time-based event log in which each event that occurred in the system is recorded. The event log represents the behavior of the system itself, but if each event in the traffic flow data is recorded as an event, the amount of analysis will be enormous and a state explosion will occur, so it needs to be discretized to some extent. Therefore, we consider extracting the major nodes in the traffic flow and recording their passage time as an event. The depth of mesoscopic depends on how detailed the nodes are set. In other words, if the number of nodes is small, the model will approach macroscopic and the amount of data will be small, and the traffic flow reproduced in the model will be rough. On the other hand, if the number of nodes is large, the model approaches microscopic and the amount of data increases, and the model can reproduce detailed traffic flow.

Fig. 24 shows the process flow of the mesoscopic modeling with process mining technique. The first step is to extract the major points in the traffic flow and to create an abstracted route graph with these points as nodes. Next, the elapsed time spent passing each node is extracted from the raw traffic flow data and recorded in an event log. The event logs are then analyzed by process mining to obtain statistical information, and a final mesoscopic model based on the abstracted route graph and the statistical information is created.

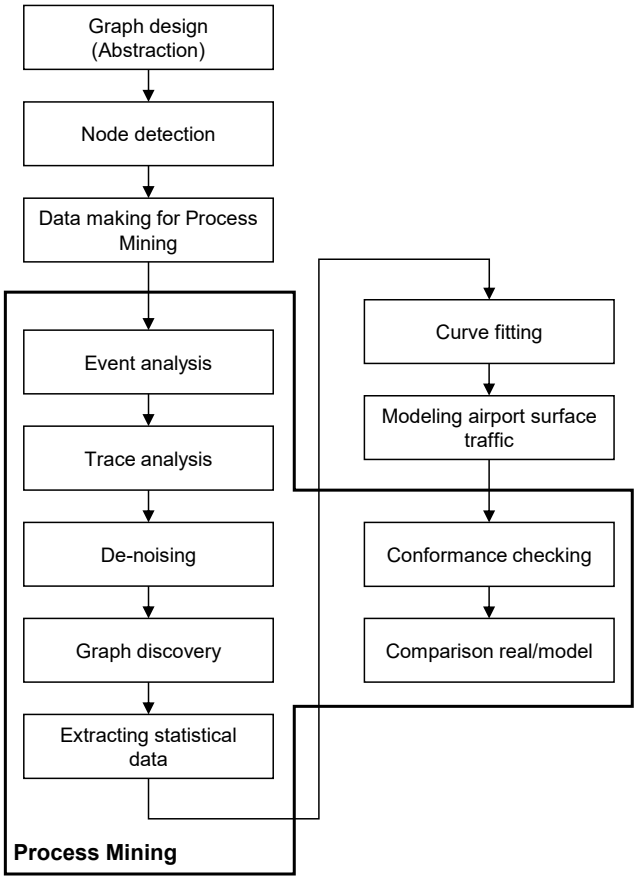


Fig. 24: Process flow of mesoscopic modeling with process mining: The statistical data necessary for mesoscopic modeling is acquired by process mining.

Process mining is used as a support tool to streamline to make statistical data from continuous values. The process flow is basically an analysis of discretized event logs, which overlaps with conformance checking in Chapter 4, but the purpose of execution is different. The purpose of using process mining technique in the mesoscopic modeling is to extract needed information and data cleansing to eliminate undesired noise in traffic flow data.

5.3 Mesoscopic Model of Airport Surface Traffic Flow

Here we use airport surface traffic flow data as an example of the mesoscopic modeling applying process mining. Traffic flow on the airport surface is limited to a small area, and moreover, it is two-dimensional data with no concept of altitude. Aircraft traffic is fixed

to taxiways and runways only, and no other routes are available. Unlike vehicles and pedestrians, airport surface traffic is restricted by severe rules, which means that moving aircrafts are almost never subject to irregular behavior. Therefore, since it is a simple traffic flow, it is easy to demonstrate the effectiveness of process mining in the mesoscopic modeling and is suitable as an example. As the airport surface traffic flow data, we use CARATS Open data [47]. CARATS Open data consists of flight trajectory data of all regular flights in Fukuoka FIR (Flight Information Region) and airport surface traffic data of some major airport in Japan. The sources of the data are radar data and flight plans. For each flight, the following information is recorded in 10-second cycles for airway data and approximately 1-second for airport surface data: timestamp, flight number (unique identification of the flight), latitude, longitude, altitude, and aircraft model (e.g., B772, B738, A320). The data contains flight trajectory data in one week of every odd month (2012-2016); every month (2017, 2018).

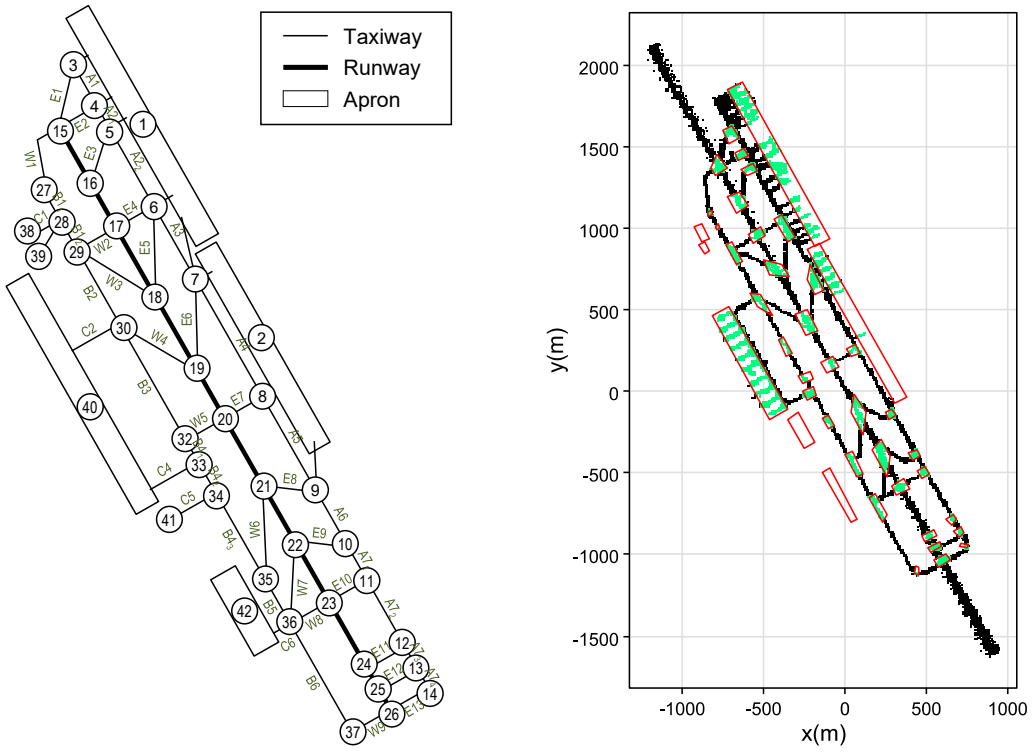
5.3.1 Graph design and node detection

The first step is to define an abstracted route graph as the framework of the mesoscopic model of the airport surface traffic flow. The degree of discretization of the mesoscopic model depends on how detailed the nodes of the graph are, but if the nodes are the junctions of taxiways and runways, and all the data between them are converted into statistical values, a model with sufficient granularity can be created to identify bottlenecks and calculate the amount of airport surface traffic [47]. Each taxiway and runway published in the AIP (Aeronautical Information Publication) is extracted and both ends of the taxiways and runways are set as nodes.

To create an event log for process mining, we extracted the aircrafts' passing information of each node in the defined graph and transformed the coordinate information recorded in the CARATS open data into a Cartesian coordinate system centered on the ARP (Airport reference point) [52]. In addition, we set a rectangle on each intersection of runway/taxiway as the checkpoint to detect crossing trajectories. By judging the inside/outside of each rectangle and trajectory, the timing of the aircraft's passage can be obtained as an event.

Fig. 25 shows the results of the inside/outside determination for all traffic flow data. If a node is judged to be inside the rectangle, it is considered to have passed the node and the time of passage is output to the event log, while all other data is discarded. When outputting the log, the edge (taxiway or runway) is determined from the two-point pairs (start and endpoints) of the nodes shown in

Fig. 25 (a) and recorded as an event. An edge containing multiple nodes is identified if the start and end points match any two points in the node list. In the case of CARATS open data, the amount of data was reduced by about a factor of 40 when the event logging was completed.



(a) Abstracted route graph (b) Node detection result

Fig. 25: Airport surface graph: The x-axis and y-axis represent the distance from ARP (Airport Reference Point) in meter. The graph in (a) consists of the main routes, runways and taxiways, and their intersections as nodes. The node detection result in (b) was obtained by setting rectangles on the airport surface according to the node coordinates of the abstracted route graph and performing an inside/outside decision on the raw data in each rectangle.

5.3.2 Process mining for mesoscopic modeling

Event Analysis: First, we split the event log into the arrival and departure aircrafts for event analysis, and the results were as shown in Tab. 6. In general, the arrival aircraft always starts from RWY (Runway) and stops at E/W-APRON (East/West Apron). Therefore, the start event should be RWY, and the end event should be only E/W-APRON, but according to Tab. 6, there are latent abnormal cases that start and end in other places. In the case of arrival aircraft, 99.81% of the start events and 97.90% of the end events are normal, and the rest are abnormal cases. In the case of departure aircraft, 97.22% of the start events and 99.74% of the end events are normal, and the rest are abnormal cases. The causes of this can be classified into three main categories: (i) errors during data recording, (ii) errors during event log generation, and (iii) actual irregular operations. During data recording, the blind area of radar and the data fluctuation may result in missing records and noise. As for errors in generating the event log, for instance, in the case of the node detection, there may be cases where edges cannot be detected due to misalignment of positional information, etc., in the process of extracting passing taxiways.

Tab. 6: Event analysis result. Grayed out events are those that are considered noise in the raw data.

Arr/Dep	Start/End	Events	Occurrence	Rate	Noise
Arrival	Start events	RWY	2658	99.81%	
		Other	5	0.19%	x
	End events	E-APRON	2052	77.06%	
		W-APRON	555	20.84%	
		Other	56	2.10%	x
Departure	Start events	E-APRON	2028	76.27%	
		W-APRON	557	20.95%	
		Other	74	2.78%	x
	End events	RWY	2652	99.74%	
		Other	7	0.26%	x

An example of an irregular operation would be, for example, when a departing aircraft breaks down during the taxiing and returns to the spot without taking off. Or, if

the airport layout is such that there are two terminals across the runway, it may involve a runway crossing to only move between spots. In such a case, the aircraft would depart from the apron and reach the runway, but would not terminate there, but would continue moving and stop at the opposite apron.

Trace Analysis: The next step is to perform trace analysis using Explore Event Log, a standard ProM plug-in. Tab. 7 shows the top 10 cases of traces and a part of the abnormal traces for departure aircraft. The majority of the traces that appear start at E/W-APRON and end at RWY. However, as mentioned in the event analysis, some of the traces are recorded in the event log in an incomplete state. As we can see from the abnormal case, for instance, in the trace <E2, RWY, END>, the departure aircraft that should have started at E or W-APRON actually started at taxiway E2. This could be due to missing data or noise during event log generation. Trace <E4, W2, B12, B11, W1, RWY, END> also starts from taxiway E4 instead of the apron. This may be due to missing data at the start. In addition, as an actual case of irregular operation, in the traces <W-APRON, RWY, E8, A5, A4, A3, E-APRON>, the aircraft left the west apron and entered the runway once, but it left the runway then stopped at the east apron.

Tab. 7: Trace analysis result

(a) Departure trace (Top 10 occurrence)

#	Trace	Occurrence	Rate
1	E-APRON, E1, RWY, END	485	18.24%
2	W-APRON, B2, B12, B11, W1, RWY, END	362	13.61%
3	E-APRON, A21, E2, RWY, END	333	12.52%
4	E-APRON, A22, A21, E2, RWY, END	177	6.66%
5	W-APRON, B42, B43, B5, B6, W9, RWY, END	117	4.40%
6	E-APRON, A21, A1, E1, RWY, END	94	3.54%
7	E-APRON, A1, A21, A22, A3, A4, A5, A6, A71, A72, A73, A74, E13, RWY, END	93	3.50%
8	E-APRON, E4, RWY, END	92	3.46%
9	E-APRON, A3, A4, A5, A6, A71, A72, A73, A74, E13, RWY, END	90	3.38%
10	E-APRON, A3, E4, RWY, END	79	2.97%

(b) Departure trace (Abnormal case)

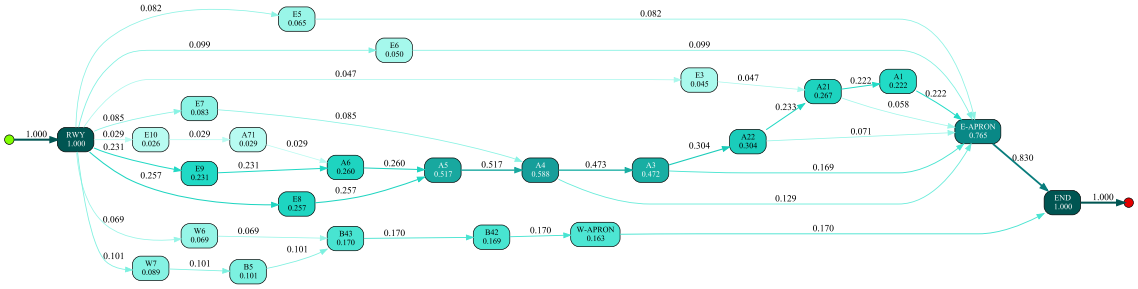
#	Trace	Occurrence	Rate
1	E-APRON, A4, E7, W5, B41, B42, B43, B5, B6, W9, RWY, END	1	0.04%
2	W-APRON, RWY, E8, A5, A4, A3, E-APRON	1	0.04%
3	E-APRON, A3, E4, W2, B12, B11, W1, RWY, END	1	0.04%
4	W-APRON, RWY, E8, A5, A4, A3, E-APRON	1	0.04%
5	E-APRON, A1, A21, E3, RWY, END	1	0.04%
6	E4, W2, B12, B11, W1, RWY, END	1	0.04%
7	A22, A3, A4, A5, A6, A71, A72, A73, A74, E13, RWY, END	1	0.04%
8	A3, A4, A5, A6, A71, E10, RWY, END	1	0.04%
9	E-APRON, E4, W2, B12, B11, W1, RWY, END	1	0.04%
10	E2, RWY, END	1	0.04%

While it is unclear what the actual circumstances were, we assume that some problem occurred on the taxiway, and the aircraft was forced to pass the runway again to avoid it, or that it was simply a spot swap. Since the number of these occurrences is small for the entire trace, they have little effect on the statistical information even if they are excluded. Therefore, they can be regarded as unnecessary noise for model creation and can be excluded from the analysis.

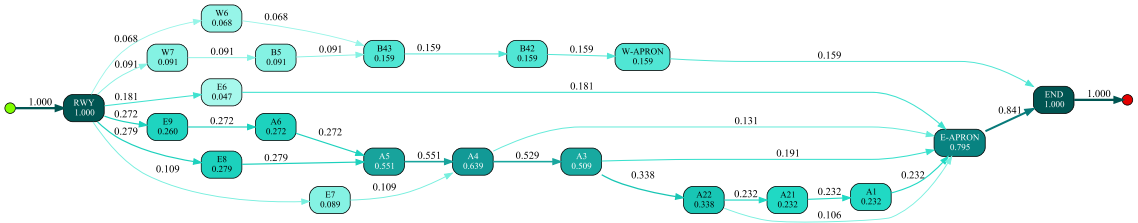
De-Noising: When performing process mining, assuring data quality is important and a challenge [48] [53]. It is necessary to eliminate errors as much as possible, but it takes a lot of time and effort to make judgments while checking each trajectory, and if an error is overlooked, the model will be created with potential errors remaining. In ProM, the standard plug-in Explore Event Log allows sorting in ascending order according to the frequency of occurrence of traces, and it is possible to regenerate the event log from which any trace is extracted. Traces with a low frequency of occurrence at a level that does not affect the statistics are eliminated here. So, traces with an occurrence rate of less than 1% were eliminated. The analyst should observe the pattern and occurrence rate of traces in the modeled events to determine the appropriate filter settings.

Graph Discovery: As mentioned in Chapter 4, process mining enables model discovery

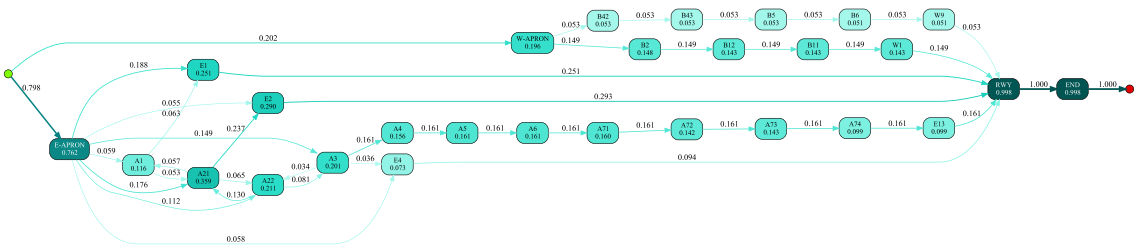
from event logs. It is possible to visually compare the event to be modeled with the discovered model, and if unintended state transitions are identified, the model can be quickly corrected. ProM's Inductive Visual Miner plug-in can be used to reproduce transition patterns from event logs as discovered model. The reproduced graphs can also be used as a reference for model creation. The graph of the arrival traffic from the data before de-noising is shown in Fig. 26 (a). The data is overfitted with unnecessary noise, resulting in a state explosion.



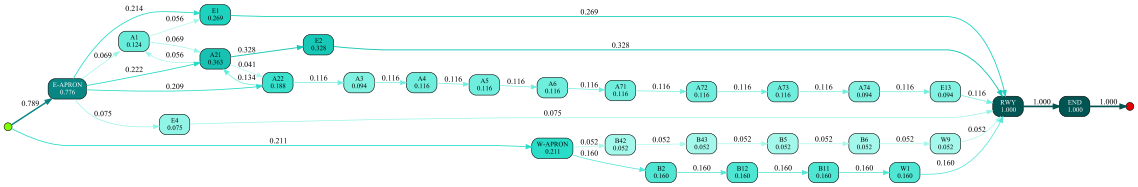
(a) Discovered graph of arrival traffic (full).



(b) Discovered graph of arrival traffic (de-noised).



(c) Discovered graph of departure traffic (full).



(d) Discovered graph of departure traffic (de-noised).

Fig. 26: Discovered models from event logs for each arrival/departure aircraft by Inductive Visual Miner. The discovery models directly from the raw event logs contain some incorrect transitions due to noise. On the other hand, in the de-noised discovered models, incorrect transitions are eliminated, and the overfitting is improved, making it easier to visually check.

On the other hand, Fig. 26 (b) shows the graph of the arrival traffic after de-noising, in which unnecessary transitions are eliminated and the number of branches is reduced to a level that can be visually confirmed. This is also true for the departing aircraft traffic in Fig. 26 (c) and (d).

When using the reproduced graph for modeling, the number of branches should be a readable level as shown in Fig. 26. If there are too many branches, it becomes difficult to confirm. In addition, the reproduced graph should be visually checked [37] because they are useful for checking whether there are any abnormal state transitions by analyzing the output log again by process mining after the model is created.

5.4 Airport Surface Traffic Flow Model

5.4.1 Probability Density Distribution of Taxi Time

The process so far has completed the extraction of traces (patterns) of airport surface traffic flow. Furthermore, with the probability density distribution of the time required for the transition between nodes (events), we can create a mesoscopic model of the airport traffic flow. The unimpeded taxi time of an aircraft on an airport surface traveling at normal speed without a single stop corresponds to the Erlang distribution [54] [55]. According to Khadilker (2013) [54], in airport surface traffic flow, the probability density distribution of travel time is asymmetric, which is most proximate to the Erlang distribution. On the other hand, Burgain had proposed a normal distribution [56], so

Khadilker calculated the Kullback-Leibler divergence [57] for the normal, lognormal, and Erlang distributions for confirmation. As a result, the Erlang distribution was consistently the smallest value in the airport surface traffic flow data. Therefore, in this dissertation as well, the Erlang distribution is used for the airport surface traffic flow model. Utilizing process mining, we obtain the probability density distribution of the taxi time required for each event and reproduce the transitions between each node in the graph.

Each node in the graph defined in 5.3.1 corresponds to a taxiway and a runway, and each time an aircraft enters a node, it is logged as an event. Therefore, by extracting the time required for the event through process mining, we can create the data necessary for curve fitting of the probability density distribution. ProM allows the insertion of elapsed time in all events by the "Add elapsed time in Trace as Attribute to all events" plug-in. The elapsed time inserted in each event becomes the sojourn-time on each taxiway and runway.

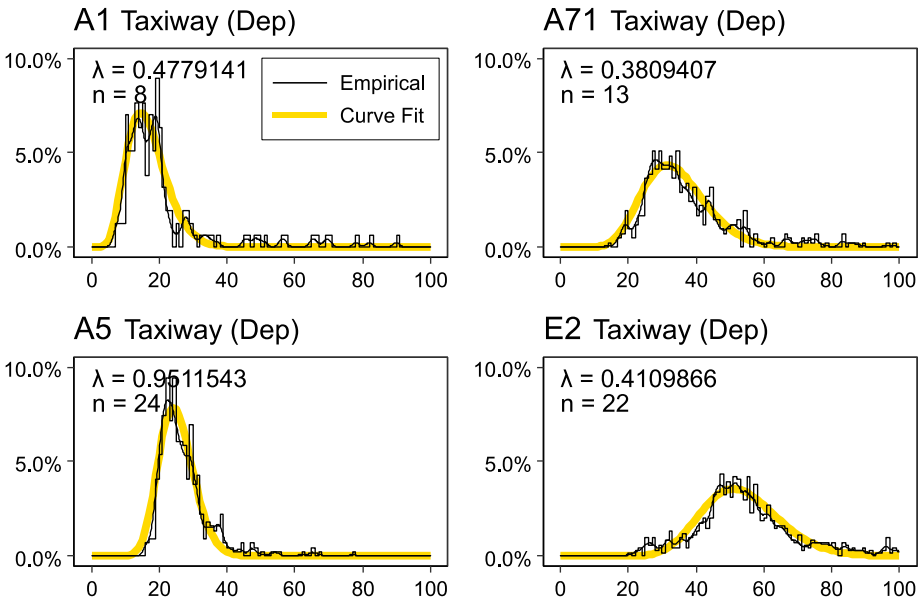


Fig. 27: Probability density of flight times taxi times on some typical links for departure traffic. The travel time between nodes can be calculated from Erlang distribution fitted to the real data. The λ and n in the upper left of each plot indicate Erlang distribution parameters.

Fig. 27 shows the probability density distribution of the sojourn-time for each event (runway/taxiway) obtained by exporting the log with the elapsed time inserted from ProM. The same process was performed for all the taxiways (after de-noising) that appeared in

the trace analysis, and further curve-fitted to the Erlang distribution to calculate the probability distribution parameters of the formula $f\ x\ = \frac{\lambda^n}{n-1\ !} x^{n-1} e^{-\lambda x}$. ProM and its plug-ins combination enable to reduce the effort to obtain these necessary data.

5.4.2 Creating Mesoscopic Model

Based on the data obtained so far, we create a mesoscopic model to reproduce the traffic flow of the airport surface. As in Chapter 2, object Petri nets RENEW is used as the modeling language. Since the mesoscopic model discretizes continuous data and replaces them with statistical information between nodes, we consider a stochastic Petri net that gives a probabilistic delay time for each node movement (firing of transitions in the Petri net). The delay time required for transition firing can be given as a function in RENEW, we used the delay time of transition firing to represent luggage loading time in the aircraft boarding model in Chapter 3. In the airport surface traffic flow model, we use the probability density distribution obtained in the previous section to generate the taxi time.

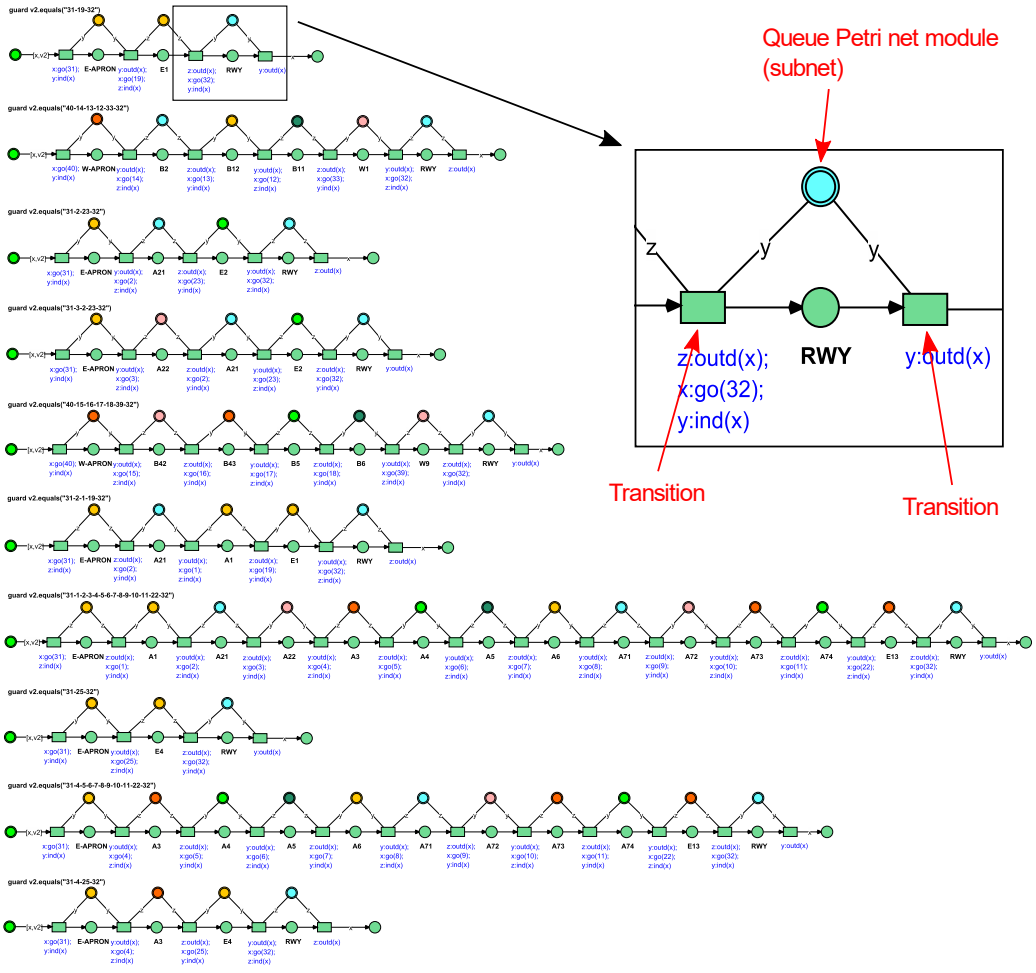


Fig. 28: Mesoscopic Petri net airport surface model with RENEW (excerpt): Each node has a built-in queue as a submodel, and inputs and outputs are performed simultaneously with the firing of neighboring transitions.

Fig. 28 shows a portion of the created model, which is a set of Petri nets based on the departure aircraft traces output by ProM. The paths taken by each aircraft are all detected in the trace analysis, and since the traces in the event log are de-noised, the airport surface traffic flow can be directly reproduced by the Petri net corresponding to the traces. The Petri net shown in Fig. 28 is a hierarchical net, where each trace node incorporates a queue instance as a submodel. Since RENEW can treat individual Petri net as object, it is possible to have all traces share the same queue. Thus, all aircrafts passing through each trace use a common taxiway and runway submodel (queue), which will reproduce the actual traffic flow competing for the same resources.

5.4.3 Model Evaluation

This section presents the result of accuracy evaluation by comparing the created model with actual data. As an example of model evaluation, we compare the traffic volume by time. The traffic volume on the airport surface is the number of aircraft present on the airport surface at a given moment [54] [58]. Here, the time axis is divided into slots of fixed time (60 seconds), and the number of aircraft on the airport surface is counted in each slot (Fig. 29).

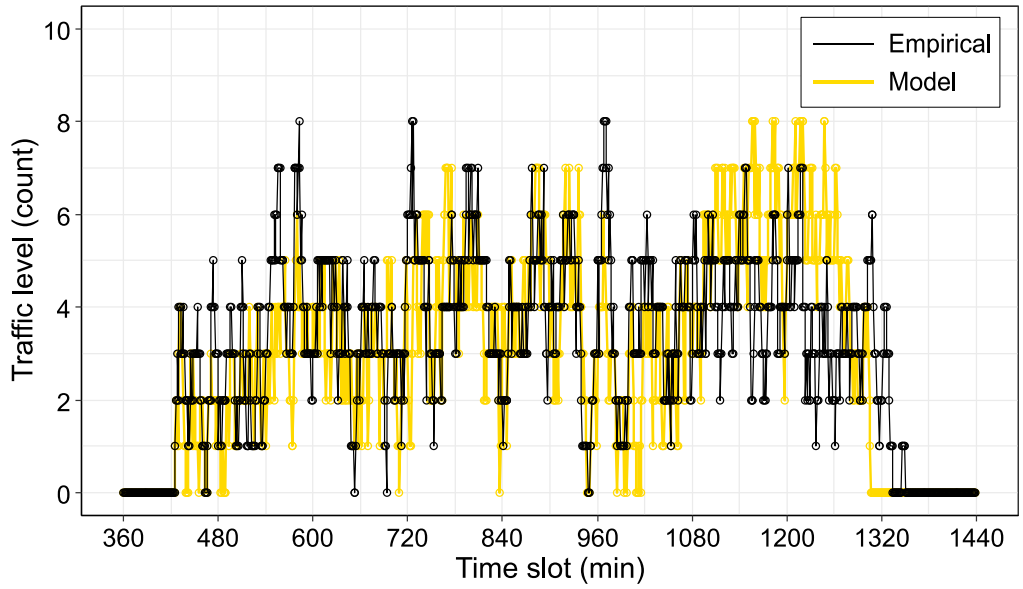


Fig. 29: Airport surface traffic level: This graph shows the 18-hours (operating hours at the target airport) period extracted from a single day, and the x-axis is represented in time slots of 60 seconds.

The traffic volume between the model and the actual data is generally identical, and it can also be confirmed that the traffic volume increases synchronously during the congestion period. The traffic volume on the model at around slot 1200 (corresponds to 20:00) is one or two aircraft higher than the actual data. This is probably due to the long congestion period caused by the evening rush from 18:00 to 19:00, which resulted in the accumulation of blockage and affected the slots behind. The occurrence of stagnation largely depends on the runway throughput. In the model, the runway occupancy time is fixed, but in actual airport operations, it is assumed that the runway occupancy time is adjusted during busy periods for efficiency. In other periods, the model follows the actual data well and reproduces the data with considerable accuracy.

In this chapter, we focus on the modeling methodology and showed the comparison of traffic volume as an example, but various other evaluations are applicable. For example, there is a method to aggregate the error in ground running time for each aircraft and score the error between the actual data and the simulation. The evaluation methods of the traffic flow model are dealt with in more detail in Chapters 6 and 7. The traffic volume comparisons made in this chapter are a simplified assessment, although they provide a rough confirmation of the reproducibility of the traffic flow data. In addition to traffic flow comparisons, Chapters 6 and 7 present several approaches to geometric accuracy checks and evaluations from an operational perspective.

Process mining is used to check the conformance of models to specifications in Chapter 4. It should be noted that what was done in this chapter using process mining is similar to what was presented in Chapter 3, but with different objectives as stated in 5.2.2. For example, in both Chapter 3 and this chapter, event analysis and trace analysis are performed. In Chapter 3, event analysis is performed to confirm that no unintended events are starting or ending, and trace analysis is performed to confirm that no unintended firing sequences, i.e., traces that do not appear in the actual data. However, the purpose of process mining in this chapter is to extract the information necessary for modeling, and is used as a support tool to improve work efficiency.

5.5 Summary

In this chapter, we have focused on the mesoscopic model and tried to improve the efficiency of its creation to solve the problem that the model creation is time-consuming, labor-intensive [8], and difficult to handle large volume traffic flow data. By abstracting traffic flow data to a level that does not compromise the necessary information, and further applying process mining techniques, we have presented a method for creating a mesoscopic model. Depending on the output required by the analyst, the continuous data can be abstracted to a certain extent to produce equivalent results. The model created in this chapter accurately reproduces traffic volume changes while downsizing the huge data to a few tenths of its original size.

It is important in mesoscopic modeling to determine the degree of abstraction.

Analysts/developers must be careful to what extent they abstract while maintaining micro-macro(meso) equivalence [8]. Lower levels of abstraction provide higher accuracy but require more modeling effort. This is true even if process mining is used. Conversely, if the level of abstraction is increased, the effort required for modeling is reduced, but necessary information may be missing. The selection of nodes to achieve an optimal abstraction level depends on the desired output and the characteristics of the traffic flow data. Therefore, the design of the graph and the determination of the abstraction level for mesoscopic modeling should be done in the preliminary stage of process mining. After the selection of the abstraction level, the model creation process in the latter stage can be made more efficient by the method presented in this chapter. If the framework of the existing model can be appropriated, it is easy to obtain statistical information by converting the acquired data into event logs by process mining. For example, the extraction of statistical information between nodes is time-consuming if the traffic flow data is processed as it is, but it can be completed in a very short time using the ProM method presented in this chapter. Therefore, when it is necessary to recreate the existing model multiple times, this method is particularly effective

In the next step, in order to handle more complex traffic flow data, we should consider extending the modeling method presented in this chapter from the airport surface to the airspace. In the case of the airport surface shown as an example in this chapter, the main points for selecting nodes, such as taxiways and runways, are clear and can be easily abstracted. However, since airspace traffic flows are more complex and ambiguous than airport surfaces, we need to consider a higher level of abstraction in the preliminary stages of mesoscopic modeling.

Chapter 6

A Framework for Extracting Abstracted Route Graphs Toward Air Traffic Flow Modeling

In this chapter, we deal with methods for creating abstracted route graphs for modeling highly ambiguous traffic flow data. While the airport surface traffic flows modeled in Chapter 5 were two-dimensional, the data targeted in this chapter are three-dimensional airspace data of greater complexity. Compared to road traffic and train traffic, modeling of air traffic flow is difficult because trajectory of each aircraft fluctuates in the three-dimensional space due to various factors such as weather and congestion level. By this reason, finding important routes on which aircrafts frequently pass is necessary for building air traffic flow models. We propose a framework for finding important routes in the form of graphs based on combination of various technologies such as space partition, trajectory clustering, and skeleton extraction.

6.1 Abstracted Route Graph for Mesoscopic Modeling

The mesoscopic model we construct in this dissertation is a network model that discretizes raw data, which are continuous values. Therefore, in the preliminary stage of model creation, it is necessary to create an abstracted route graph from the raw traffic flow data. The abstracted route graph is an important fundamental part of the model's performance and should accurately capture the characteristics of the traffic flow while greatly reducing the amount of data. If the abstracted route graph does not match the mainstream (major path) of traffic flow, the mesoscopic model will be significantly less accurate.

In the mesoscopic modeling of airport surface traffic flow in Chapter 5, the

abstracted route graph was composed of taxiways/runways and their intersections. Since an airport is a two-dimensional, restricted, boxy field, it is relatively easy to create an abstraction graph. However, if the traffic flow is extensive and has no fixed path, the complexity increases, and the abstracted route graph becomes significantly more difficult to create. To extend the applicability of mesoscopic modeling to all types of traffic flows, this chapter develops a framework of extracting abstracted route graphs using three-dimensional, unfixed pathless airspace data as an example.

6.2 Airspace Data Abstraction

By extracting flight routes from flight trajectory data, we can compose simulation models which contribute to improvement of flight operation, congestion control, and reduction of CO_2 emission. We first identify important points in the target space, e.g., crossings in a road network, and then creating graphs having such points as nodes and various traffic parameters assigned to edges to build mesoscopic models for traffic flow. The abstraction level of traffic flow determines accuracy of simulation results. Compared to road traffic and train traffic, modeling of air traffic flow is difficult because trajectory of each aircraft fluctuates in the three-dimensional space due to various factors such as weather and congestion level. In Chapter 5, we applied the mesoscopic modeling approach to traffic flow on an airport surface. An airport surface consists of several fixed points and lines such as aprons, taxiways and runways. This situation is similar to that in road networks. We extracted a graph having nodes representing crossings of taxiways and gave the probability distribution of velocity on each edge. The inputs of the model are departure flights from one of aprons and arrival flights to one of runways. To apply the mesoscopic modeling to traffic flow in the airspace, we need to extract routes in the airspace together with velocity distributions on the routes. Developing methods to do this is the main objective of this chapter. We propose a framework for finding important routes in the form of graphs, called abstracted route graphs, based on two approaches: (i) density-based clustering of trajectories and skeleton extraction, and (ii) space partition and trajectory clustering. The obtained abstracted route graphs are used for building mesoscopic models of air traffic flow. Through evaluating the accuracy of the abstracted route graphs, we

show that the proposed framework is appropriate for the mesoscopic modeling of the airspace.

The framework is outlined as follows. First, we explain existing technologies used in the proposed framework. Next, pre-processing of air traffic trajectory data is described. Based on the pre-processed airspace data prepared for the model, the two approaches mentioned above are explained. Then we show how to compose abstracted route graphs from the outputs of the two approaches. The obtained abstracted route graphs are evaluated from two different points of view in the last section.

6.3 Clustering Algorithm

The proposed framework is a combination of various technologies. We give brief introduction of them. The core technology is clustering of trajectories. It aggregates trajectories with high similarity into clusters and eliminates unimportant trajectories. As the similarity, various distance measures between two trajectories are proposed, e.g., Euclidean distance, distance between trajectories used in TRACLUS [59] and SSPD [60], distance defined on time series data used in DTW [61], and edit distance used in LCSS [62] and EDR [63]. There are variety of clustering algorithms. Due to the characteristics of flight trajectory data, we need to select an appropriate algorithm and a similarity measure. In, multiple distance measures are applied to various datasets but there are no significant differences in the obtained results.

For the abstracted route graph extraction, there are several methods for simplifying traffic trajectory data and extracting abstracted routes from it. In [64] [65], space partition and the genetic algorithm are applied to AIS (Automatic Identification System) data and patterns of ship trails are found. In [66] [67], main routes of aircrafts in the airspace are extracted using clustering of aircraft trajectories, where DBSCAN [68] based algorithm HDBSCAN is used as the clustering algorithm. In [66], superiority of trajectory clustering algorithm TRACLUS is suggested but this algorithm has not been applied to flight trajectory data.

Fig. 30 shows the proposed framework consisting of two approaches. In Method 1, firstly frequently used areas in the airspace are extracted by density-based clustering, and

next skeleton of routes are found. In Method 2, firstly the target airspace is divided by the density of traffic in each region, and then trajectory clustering is applied to find representative routes. Different from car traffic and train traffic, we need to consider the altitude in the airspace. This will be considered in the abstraction process of traffic flow.

- **Method 1:** Density-based Clustering and Skeleton Extraction (DC/SE)
- **Method 2:** Space Partition and Trajectory Clustering (SP/TC)

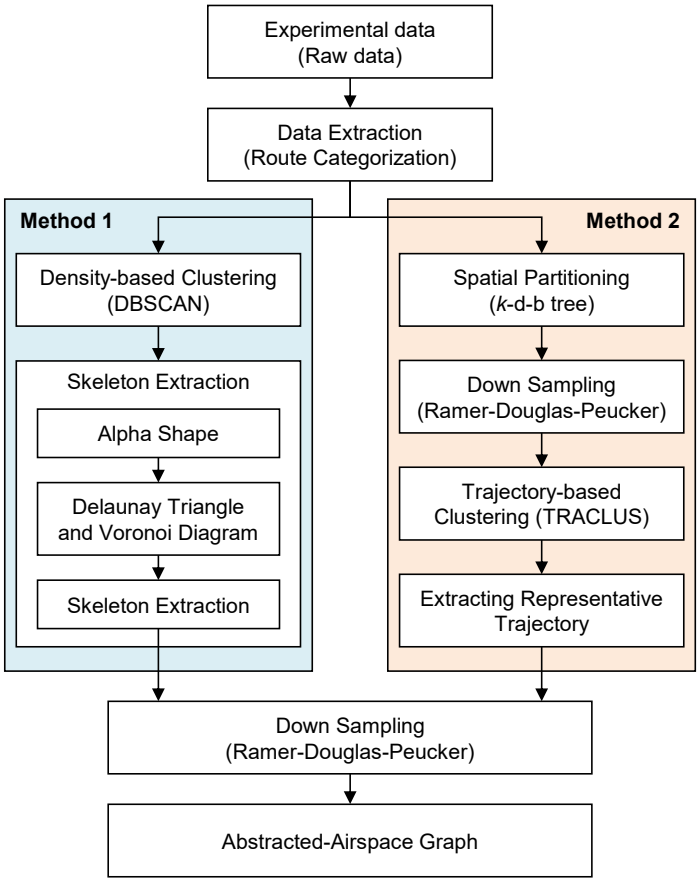


Fig. 30: Overview of framework. As a pre-processing step, the target airspace data is cut from the raw data, and the data is used to generate an abstracted route graph using Methods 1 and 2, respectively. For down sampling in the final stage, the same parameters are used for both methods.

6.4 Pre-processing of Flight Trajectory Data

As well as the airport surface traffic model in Chapter 5, we use CARATS Open data [47]. The density of traffic around airports is much larger than that of other areas. We focus on

the areas around an airport since trajectory endpoints are easy to identify in airport departure/arrival data and contribute to the acquisition of clean data sets. We here apply the proposed framework to the area around Tokyo International Airport. The pre-processing procedure is described as follows:

(1) Flight trajectory data in the approach control area of Tokyo International Airport is extracted from the CARATS data.

(2) Arrival flights and departure flights are separated, and other flights that pass through this area are eliminated from the data.

(3) The coordinate system is changed from WGS-84 to the plane rectangular [69].

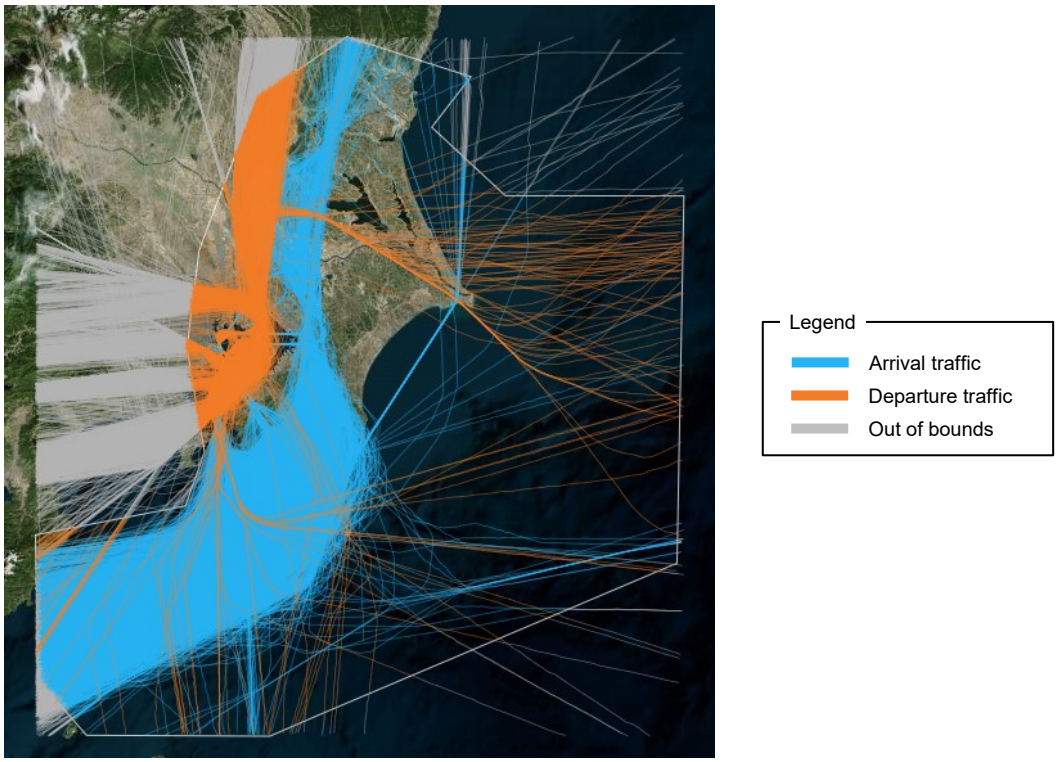


Fig. 31: Trajectories after pre-processing. The area surrounded by white lines is the approach control area of Tokyo International Airport. For each trajectory, a decision is made as to whether it is inside or outside this rectangle, and the data is further sorted by departing or arriving flights. Data near the boundary is used even if it is outside the boundary because it is necessary for space partition and clustering.

Fig. 31 shows trajectories after the pre-processing colored by arrival and departure flights. The approach control area is also indicated as a rectangle. Decision of

arrival/departure flights is made as follows: if the initial point of a flight is close to the airport and with low altitude, then the flight is judged as a departure one; if the final point of a flight is close to the airport and with low altitude, then the flight is judged as an arrival one; other flights are eliminated from the data. Since data near the boundaries of the approach control area is necessary for accurate spatial partition and clustering, data outside the boundaries are also partly used for processing.

6.5 Density-Based Clustering and Skeleton Extraction

Density-based Clustering and Skeleton Extraction (DC/SE) treats the raw traffic flow data as a set of points and extracts point sets in areas of concentrated air traffic as clusters by density-based clustering. Next, the centerlines are extracted from the contours of the clusters to generate an abstracted route graph. The process is similar to estimating the skeleton from the body of the obtained clusters.

6.5.1 Density-based clustering

Density-based clustering is an algorithm that starts with areas having points densely and expands them by merging neighbor points. DBSCAN is one of representative density-based clustering algorithms. We here use this algorithm. Since DBSCAN generates clusters according to the density of points, clusters are generated in frequently used areas and traffic in other areas is eliminated as noise. This works as filtering of data, i.e., only important areas that contain many flight trajectories remain. There are two parameters ε_d and $minPts$ in DBSCAN. A cluster is composed if more than or equal to $minPts$ points exist in a circle with radius ε_d. This procedure is applied to all points in the data and the obtained clusters are concatenated to make clusters larger. Points that do not belong to any clusters are eliminated as outliers.

Large radius ε_d results in clusters having unnecessary points. The makes performance of filtering worse. Large $minPts$ works negatively for the generation of clusters. Small $minPts$ enlarges the area of each cluster and this makes the post processing for finding main routes difficult. Since the value of $minPts$ is sensitive to the amount of data, it is necessary to adjust it even for the same kind of data source. Fig.

32 (a) shows the result of DBSCAN. We can observe that only frequently used routes remain, and other routes are eliminated.

6.5.2 Skeleton extraction

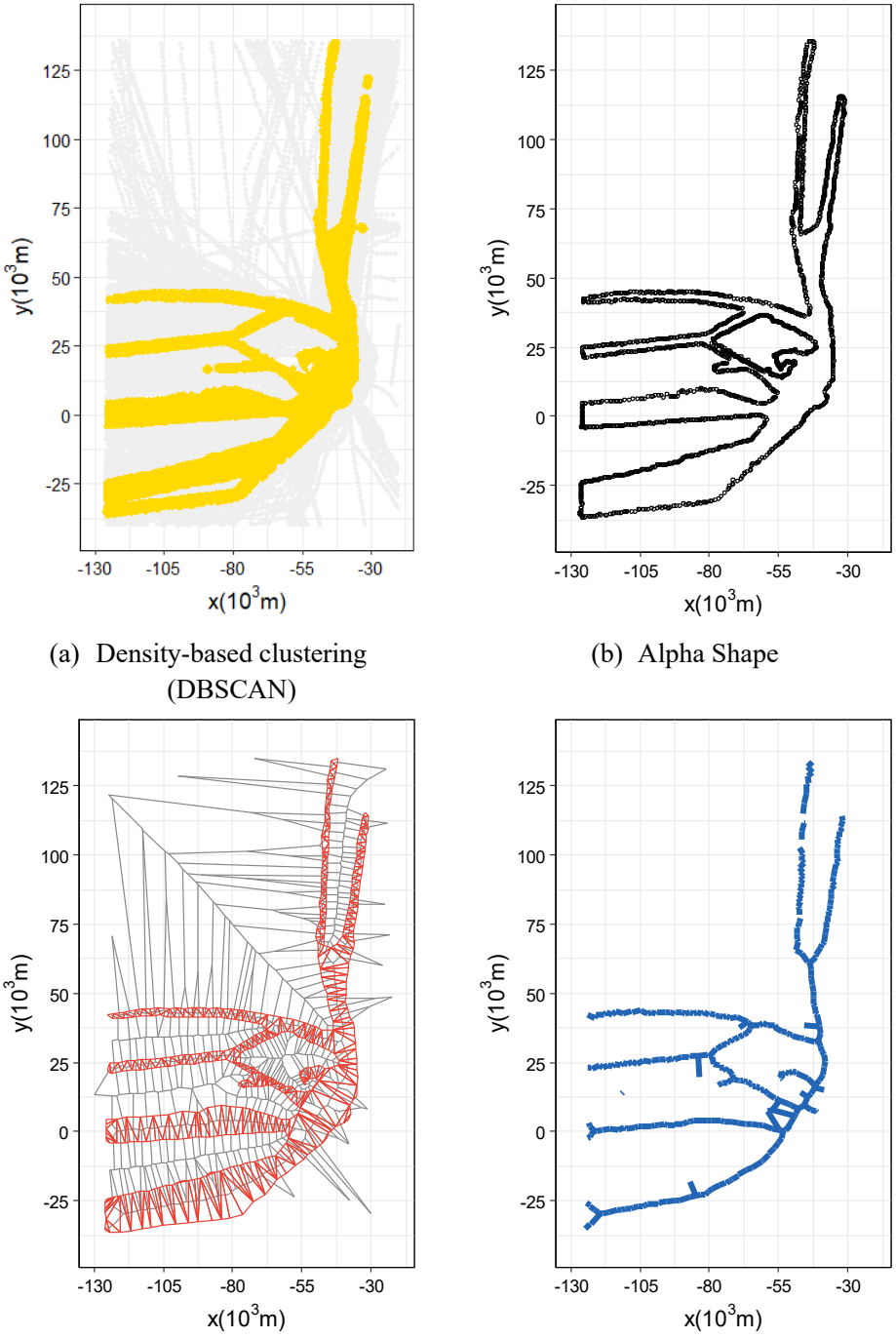
To obtain main routes in airspace, we extract skeletons from each cluster. We here adopt the following approach: first we compute Voronoi diagrams from envelops of the clusters, and then skeletons are obtained by connecting edges of Voronoi diagrams. The details are described below.

- **Alpha shape:** Alpha shapes are extension of convex hulls. We use them for computing contours of the clusters obtained by DBSCAN. The alpha shape method has one parameter α that controls the accuracy of approximation for a given set of points. By the value of α, the alpha shape becomes a convex hull ($\alpha = 0$), and can have holes inside of it. Fig. 32 (b) shows a result by the alpha shape method. The the obtained alpha shape indicates not only external borders of the cluster but also an inner hole. Such a hole is important to get accurate skeletons. A single value of α to all points may not bring a desirable result. Therefore, we use a different value of α for the hole areas. In the current implementation, we manually choose an appropriate value of α according to the density of points in each region.

- **Delaunay Diagrams and Voronoi Diagrams:** Contours of clusters are obtained as alpha shapes. Next, we extract the skeleton of clusters using Delaunay diagrams and Voronoi diagrams. Given a set of seed points on a plane, a Voronoi diagram is a partition of the plane to regions called Voronoi regions such that all points of each region have the same closest seed point. In the case of the Euclidian plane, each border of regions consists of bisections of the line segment between two seed points. Delaunay diagrams are the dual of Voronoi diagrams and indicate the adjacent relation between two Voronoi regions. A Delaunay diagram is obtained as follows: we first put a point in each Voronoi region and draw a line between two points if the corresponding Voronoi regions are adjacent. Usually, a Delaunay diagram consists of triangles called Delaunay triangles. Fig. 32 (c) shows the Delaunay diagram and the

Voronoi diagram obtained from the alpha shape. Since borders of each Voronoi region is a part of bisections of two points, we can find the center line of the alpha shape as follows: we extract edges of Voronoi regions such that both end points are included in a Delaunay triangle. The edges are shown in Fig. 32 (d). These edges indicate the skeleton of the alpha shape.

There are other methods to extract skeletons from polygons. In the straight skeleton method [70], a given polygon is shrank into inside uniformly and finally the center line of the polygon is obtained. This method is usually applied to predicting the roof shape of buildings from satellite images. It can be used for skeleton extraction of traffic flow.



(a) Density-based clustering (DBSCAN)

(b) Alpha Shape

(c) Delaunay/Voronoi Diagram

(d) Skeleton Extraction

Fig. 32: Extraction of main routes by DC/SE. Once areas of traffic concentration are extracted by density-based clustering, their contours are obtained by Alpha shape. Furthermore, the internal skeleton is extracted from the contour data, and finally only the main paths are output.

6.6 Space Partition and Trajectory Clustering

In Space Partition and Trajectory Clustering (SP/TC), the airspace is first divided by importance to traffic flow through space partition, and trajectories are further segmented within the finely divided space. Each segment is clustered by trajectory clustering process. This is a technique that aggregates similar trajectories together and replaces them with a representative trajectory. The trajectories frequented by aircraft are clustered, resulting in a set of important trajectories for the airspace.

6.6.1 Space partition

In the space partition process, a given two-dimensional space is firstly divided into regions by grid lines having variable intervals and then a tree structure that represents the relation between grid regions is computed. By eliminating unimportant regions from the space, we can reduce computational cost for manipulating the partition. This technique is often used for acceleration of drawing in computer graphics.

We use the space partition technique to divide the space into regions having variable granularity determined by the amount of traffic. As indicated in Fig. 31. trajectories of aircrafts do not necessarily exist uniformly in the airspace. In the proposed space partition, high traffic areas are divided by high-resolution grids and other areas are by low-resolution grids. This approach contributes to reducing computational cost in the post processes. As the tree structure, Dobrkovic [64] proposes quad trees. However, Filipiak [65] shows that k-d-b trees (k-dimensional binary trees) [71] have less branches at leaf nodes than quad trees. So, we adopt k-d-b trees here.

(a) shows the result of the space partition together with density of traffic. It is observed that the size of each region becomes smaller adaptively to the density of traffic. We give the size of leaf nodes k as a parameter of the k-d-b tree partition. We determine the grid interval according to the number of points contained in the region. As a result, congested areas have deeper subtrees than other areas. k determines the number of points in a single leaf node. It also determines the number of partitions in space and the depth of the k-d-b tree.

6.6.2 Trajectory clustering

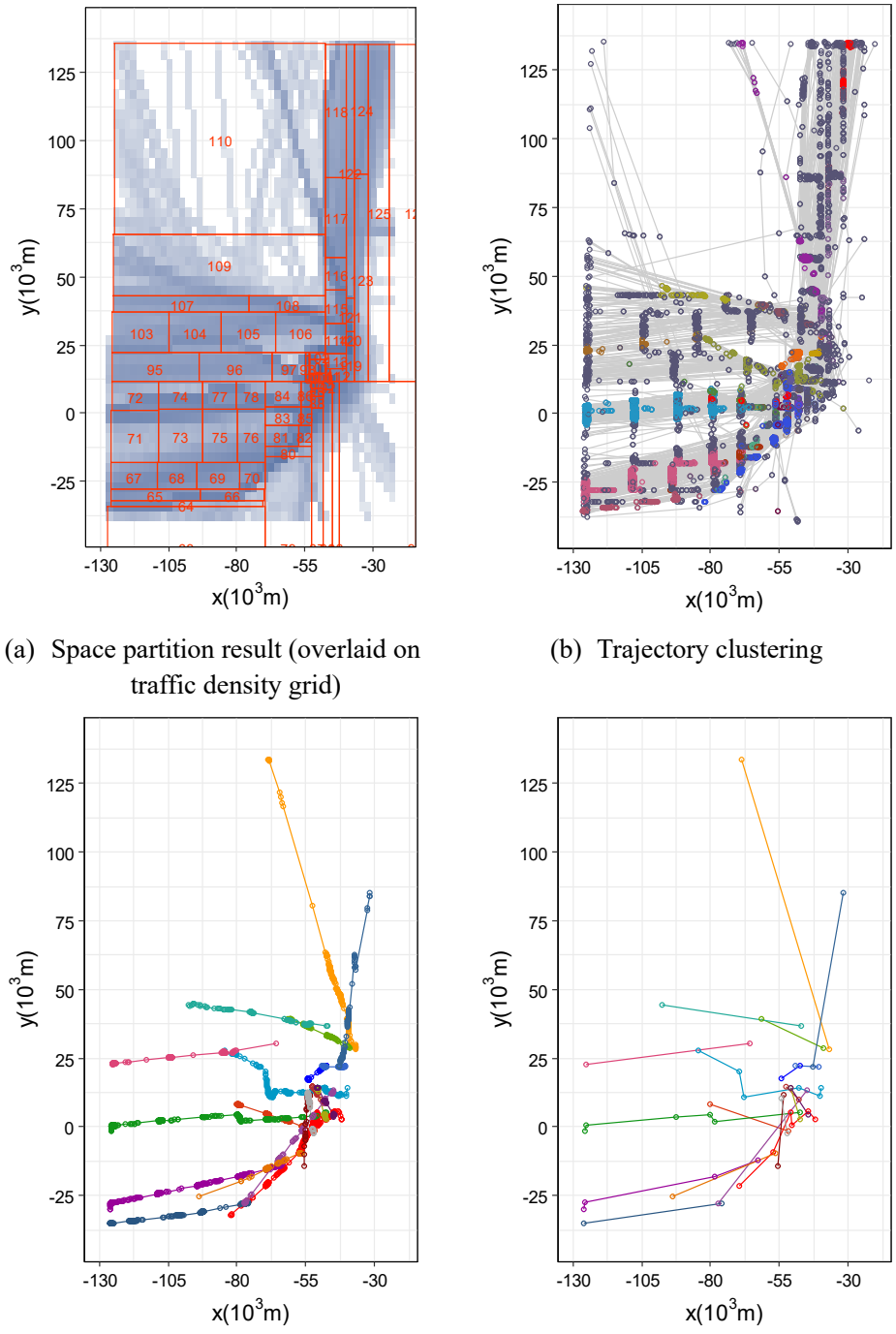
In the trajectory clustering, clusters are created from fragments of flight tracks. This is different from DBSCAN that considers only individual points. One of disadvantages in the point-based clustering is that we cannot distinguish routes on the same trajectory but in opposite direction since the direction vector of each track is not considered. This is possible in trajectory-based clustering.

- **Trajectory clustering algorithm:** We use here TRACLUS [59] as the clustering algorithm. In this algorithm, trajectories are divided into multiple segments first and clusters are created within each segment. Typical trajectories around an airport start with a common area (airport terminal area) and then branch. For such trajectories, we can obtain a cluster corresponding to the common area. Clustering in TRACLUS is similar to that in DBSCAN. If there are more than or equal to $minLns$ segments within radius ε_t of segment L, then a new cluster having segment L as its core is created. As long as the density of segments around the cluster exceeds the threshold $minLns$, the cluster is enlarged. In the original TRACLUS, trajectories are simplified and segmented by the MDL method [72] in the pre-processing stage of clustering. After this pre-processing, however, the length of each segment depends on the shape of tracks and independent of the importance of each region. We determine the length of each segment according to the size of leaf regions, i.e., segments become smaller in important regions. The maximum segment length of each trajectory depends on the importance of the airspace, and furthermore, the alignment of segment lengths around the perimeter has the advantage of reducing ε_t and facilitating cluster formation.

  The result by TRACLUS is shown in Fig. 33 (b). Since the clustering result drastically changes for the values of parameters, we need to adjust them carefully. Similarly to DBSCAN, $minLns$ should be determined based on the amount of trajectory data, We here use $\varepsilon_t = 463m\ (0.25NM)$ derived from precision of the aircraft navigation system, and $minLns = 30$. End points of each segment is colored by the cluster number that contains it. It is observed that routes from the airport are clearly separated. Fig. 33 (c) shows the result of this step.

- **Extraction of representative tracks:** Main routes are obtained by connecting representative tracks extracted from the result of trajectory clustering. We here use TRACLUS algorithm [59] for computing main routes. In this algorithm, end points of each segment are found by scanning the plane. Next, the average coordinates of the segment that crosses the scan line are obtained. To do this, we first compute the average vector $\vec{V}$ of all vectors $\overrightarrow{v_1}, \overrightarrow{v_2}, \cdots, \overrightarrow{v_n}$ each of which represents a segment in the target cluster. Then we rotate the plane so that $\vec{V}$ becomes parallel to the x-axis and the scanning is done along the direction of $\vec{V}$.

- **Down sampling:** The obtained tracks may have too many points. To simplify them, we apply the polyline simplification technique. In this technique, a point is removed when its deviation from the route is less than a given threshold value. We here use Ramer-Douglas-Peucker (RDP) algorithm [73] [74]. RDP algorithm is originally a technique for cartographic generalization of drawing maps and is used for simplifying the map according to its scale. RDP algorithm has one parameter ε_r. If the deviation of a point is smaller than ε_r, then the point is removed. We use $\varepsilon_r = 926m\ (0.5NM)$ here. The result of simplification is shown in Fig. 33 (d). Note that RDP algorithm is also applied to outputs of TRACLUS, skeleton graphs, and k-d-b trees.



(a) Space partition result (overlaid on traffic density grid)

(b) Trajectory clustering

(c) Representative routes

(d) Down sampling result (polyline simplification)

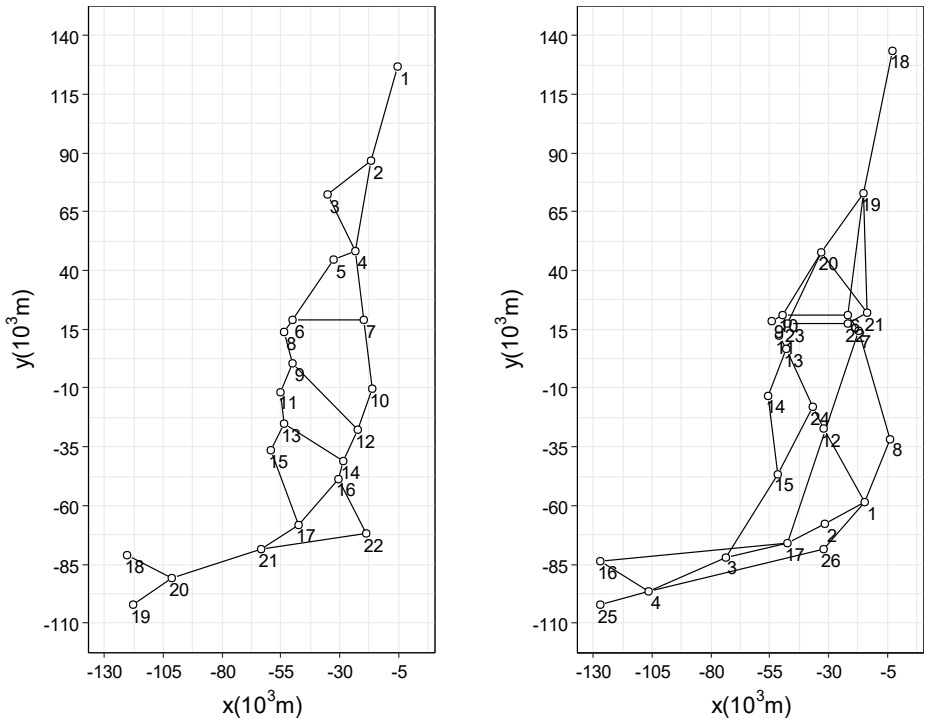
Fig. 33: Extraction of main routes by SP/TC. Space partition changes the fineness of the space according to the importance of the airspace to the traffic flow. The trajectories are segmented in the divided airspace, and each segment is clustered by trajectory clustering. After replacing each cluster with a representative trajectory, nodes comprising the polyline of the representative trajectory that change little are eliminated, leaving only important nodes.

6.7 Evaluation of Abstracted Route Graphs

DC/SE and SP/TC resulted in a set of segment data/representative trajectories, which are the components of the main route in the airspace. In this section, we create abstracted route graphs by concatenating these components, and evaluate them. An abstracted route graph can be evaluated based on how accurately it reproduces the features of the airspace data. We here evaluate abstracted route graphs by two approaches: a geometric approach and an operational approach.

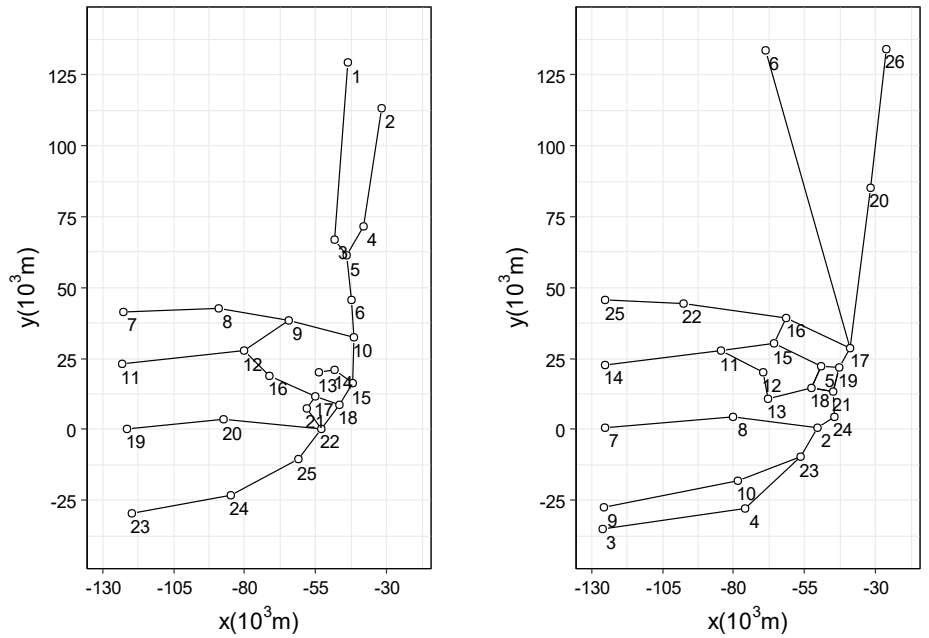
6.7.1 Composing abstracted route graphs

We obtain abstracted route graphs from results by DC/SE and SP/TC, respectively. For the output of DC/SE, clusters are concatenated to obtain graphs. For the output of SP/TC, we first remove unnecessary edges that remain after the processing based on Delaunay/Voronoi diagrams and apply RDP algorithm. The obtained abstracted route graphs are shown in Fig. 34. It is observed that the two methods give similar graph structures but details such as the number of branches at each node are slightly different. In Fig. 34, there is no distinction of node number in both DC/SE and SP/TC for departure and arrival routes, so the constituent node is set as $U = \{1^a, 2^a, \cdots, m^a, 1^d, 2^d, \cdots, n^d\}$ in this case. The superscripts a and d of each node denote arrival and departure, respectively. m denotes the number of component nodes in the arrival path graph, and n denotes the number of component nodes in the departure path graph.



(a) DC/SE arrival route

(b) SP/TC arrival route



(c) DC/SE departure route

(d) SP/TC departure route

Fig. 34: Obtained abstracted route graphs. The representative trajectories/segment data obtained from DC/SE and SP/TC are concatenated to generate abstracted route graphs. Although both DC/SE and SP/TC provide good visual abstraction compared to the actual traffic in Fig. 31, we further evaluate numerically to see how the abstracted route graphs obtained from the two methods differ.

6.7.2 Evaluation by geometrical approach

We evaluate accuracy of the obtained abstract route graphs by the distance between the abstracted route graph and actual trajectories. We compute the mean value and the deviation of the distance. Fig. 35 shows how to evaluate the distance between the abstracted route graph and a trajectory. We first compute the distance between each edge of abstracted route graph and each point on the trajectory (Δd), then the error is evaluated by the mean error μ and the standard deviation σ of the set of Δd's. The results are shown in Tab. 8. SP/TC gives the smaller mean error and standard deviation than DC/SE. From this result, we conclude that the abstracted route graph by SP/TC represents frequently used routes more accurately than that by DC/SE.

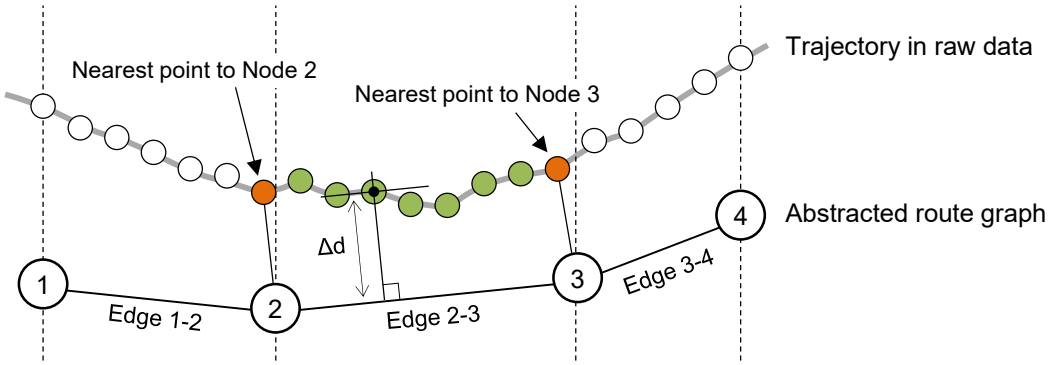


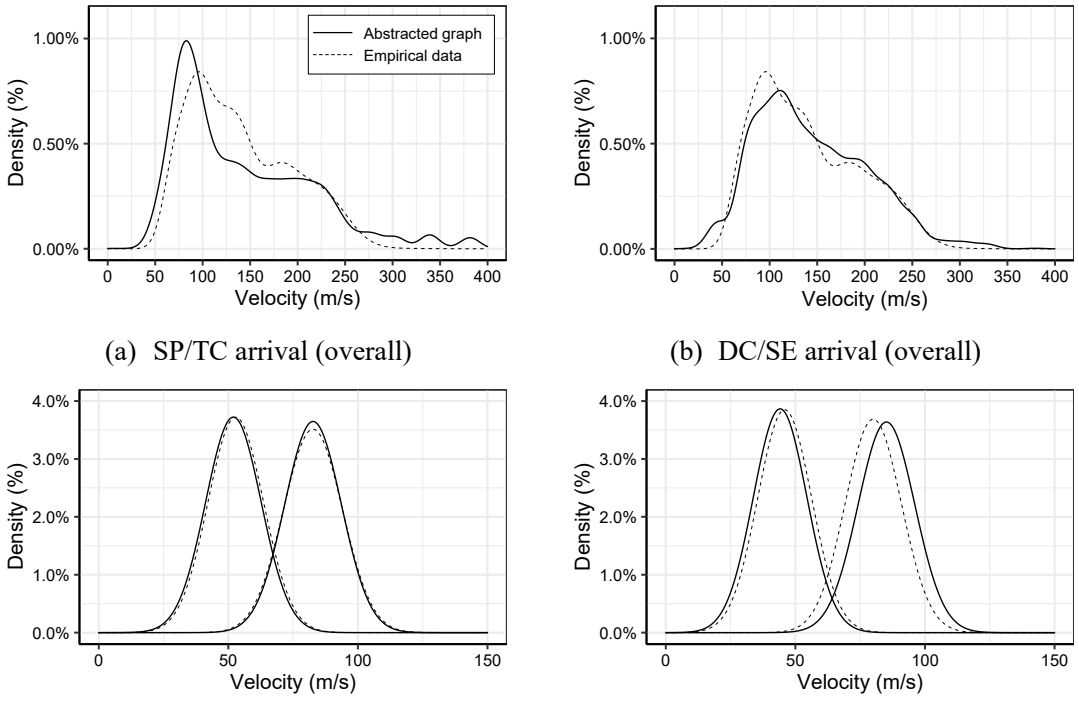
Fig. 35: Geometric relationship between real trajectories and graphs. In each trajectory, the nearest neighbor points of the two nodes constituting an edge are calculated; for points within the area bounded by the two nearest neighbors, the shortest distance to the edge is calculated.

Tab. 8: Results of geometric comparison of abstracted route graphs and actual trajectories

Route	Method	Mean μ (m)	Std. dev. σ (m)
Departure	DC/SE	15628.95	25210.33
	SP/TC	9304.77	18861.86
Arrival	DC/SE	14150.57	21555.69
	SP/TC	6993.16	14309.45

6.7.3 Evaluation by operational approach

From the operational approach, we compare the distributions of velocity between the actual data and the abstracted route graph. Since the abstracted route graph has no timing information, the velocity on the graph is computed as follows. For each flight trajectory, we find the nearest point to each node of the abstracted route graph (see Fig. 35) and regard that the aircraft passes though the node at the same time as the time of the nearest point. The velocity on each edge is computed from the length of the edge and the transit time between the end points of the edge. In the proposed mesoscopic models described in the next subsection, all traffic flow is assumed to be on the abstract flow graph. By this comparison on the velocity, we can evaluate whether the abstracted route graph is appropriate for predicting future traffic flow or not.



(a) SP/TC arrival (overall)

(b) DC/SE arrival (overall)

(c) SP/TC arrival (6^a-10^a/22^a-23^a/6^a-13^a)

(d) DC/SE arrival (7^a-6^a, divided into two parts with 65 m/s)

Fig. 36: Evaluation of velocity. It shows the results of a comparison between the velocity distributions created directly from real data and the velocity distributions generated from an abstraction graph. The smaller the difference between the actual velocity distribution and the velocity distribution in the graph, the more accurately the abstracted route graph reproduces the behavior of the aircraft. Therefore, if nodes can be placed in the area where more aircraft pass, the error will be smaller.

Fig. 36 shows the distributions of velocity on the abstracted route graph and the empirical data, where histograms are flattened with band width 10 m/second. We can observe that both SP/TC and DC/SE give overall distributions close to that of the empirical data. Next, we focus on a specific route corresponding to edge 7^a-6^a in the arrival graph by DC/SE. For this route, the graph by SP/TC has three edges 6^a-10^a, 22^a-23^a, and 6^a-13^a. This is due to characteristics of the two methods. In SP/TC, high-density areas are divided into small regions that will belong to different clusters. In DC/SE, however, clusters with density more than certain level are concatenated and as a result high density areas tend to have large clusters. We can observe both velocity distributions are similar.

6.7.4 Application to mesoscopic model

We briefly describe how to construct mesoscopic model from the abstracted route graph. The detail of model construction is presented in the next chapter. Using the abstracted route graph, a mesoscopic model used for predicting future traffic demand is constructed as follows. The basic structure of the model is similar to that for airport surface in Chapter 5. The probability distribution of velocity is assigned to each edge. Input of the model is the location and the time of an aircraft that arrives at the target area or departs from the target airport. Each input aircraft is put on a nearest node of the abstracted route graph and moves to neighbor node according to the distribution of velocity. Therefore, at each time step, estimated number of aircrafts assigned to each node is computed. In this modeling, information on individual aircraft, i.e., location at each time step, is aggregated as the expected number of aircrafts at each node.

6.8 Summary

Unlike airport surface traffic flow, in the preliminary stages of modeling, it is necessary to first extract the characteristics of the traffic flow and generate an abstracted route graph. The mesoscopic modeling method with process mining presented in the previous chapter cannot be applied directly without the abstracted route graph. Therefore, in this chapter, we have shown a framework for extracting route graphs that represent characteristics of

air traffic flow. Since fluctuation exists in flight trajectories, representative routes are extracted. The proposed framework consists of two methods, DC/SE and SP/TC. In DC/SE, first density-based clustering is applied and next a geometrically-defined skeleton is extracted from each cluster. In SP/TC, first the airspace is divided into grid regions according to importance of each region and next trajectory clustering is applied. The abstracted route graphs are obtained by both methods. The obtained route graphs by the two methods have different characteristics. Separate routes in the graph by SP/TC may be merged in the graph by DC/SE, i.e., the SP/TC gives more detailed graphs. We need to choose the method according to usage of the model.

Through the evaluation on geometrical and operational points of view, we observe SP/TC gives a more appropriate graph for the modeling, but the result may change by parameter setting. Currently, we need to find manually optimal values of parameters which affects characteristics of obtained graphs in a trial-and-error manner. We need to choose appropriate values for the purpose of modeling such as models having precise location, models with precise transit time, etc. Systematic methods to find optimal values of parameters for specific usage are remains. Also, the evaluation here is only the comparison of the abstracted route graph and the raw data. We further create a mesoscopic model based on this abstraction graph. What is important to the analyst is how well the model reproduces the original traffic flow. An abstracted route graph may perform well in the evaluation of geometric and operational approaches, but it is not a good modeling framework if it fails to reproduce the original traffic flow in the later stages of the modeling.

Chapter 7

Mesoscopic Modeling, Simulation and Performance Evaluation for Air Traffic Flow in the Airspace

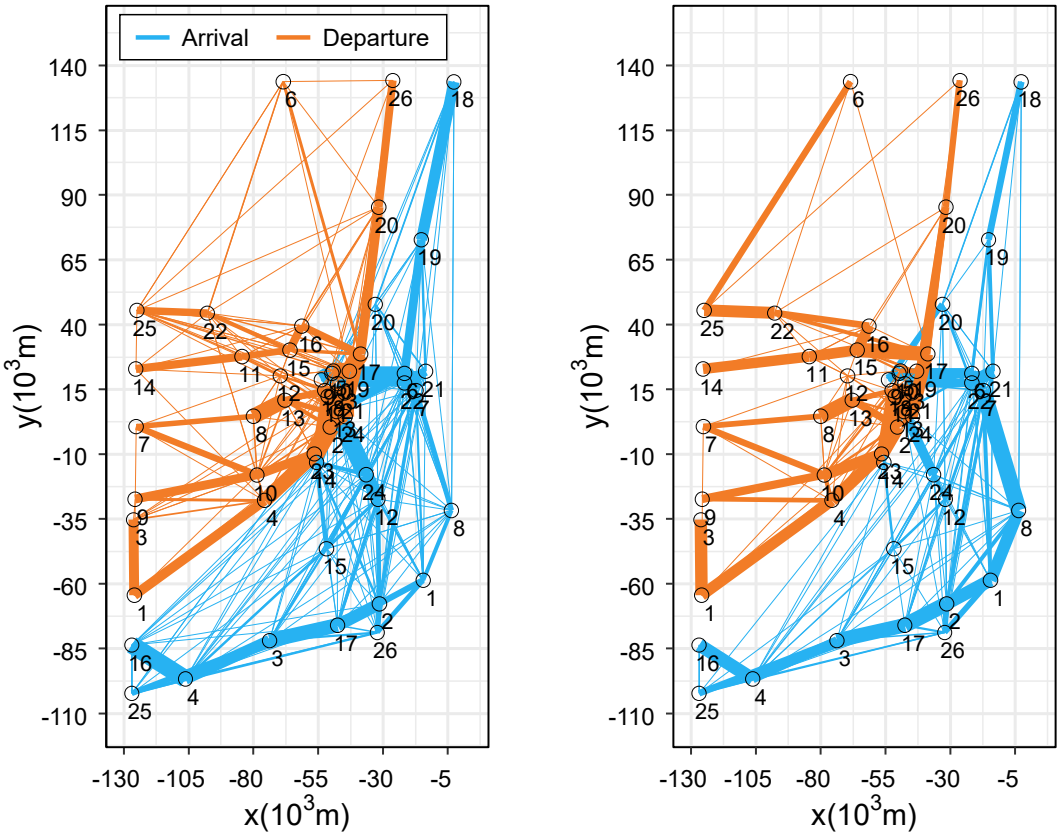
In this chapter, the mesoscopic modeling is performed using object Petri nets based on the airspace abstracted route graph created in Chapter 6. Although the abstracted graphing framework in Chapter 6 enabled us to obtain the mainstream (major path) of traffic flow in the target airspace, modeling airspace is different from airport surfaces, which are narrow in scope, and cannot be achieved simply by forming a simple network model. Here we show how to identify route information, extract statistical information, and efficiently generate models with a large number of nodes and edges. We also evaluate that the generated models precisely reflect the characteristics of traffic flow and have sufficient accuracy while significantly reducing the amount of data.

7.1 Traffic Volume Analysis

In this section, we analyze the movement information of each aircraft between nodes by comparing the abstracted route graph with the raw traffic flow data. At the stage of creating the graph in Chapter 6, representative trajectories by trajectory clustering were combined to form edges between nodes, but in actual operations, there are transitions between nodes other than representative routes, albeit at low frequency, due to shortcuts and detours. Here, we acquire transition information between these nodes by using process mining.

7.1.1 Inter-node transition information

Three-dimensional flight paths are highly ambiguous compared to airport surface traffic flow. As can be seen from Fig. 31 in the previous chapter, especially on arrival traffic, frequent route deviations and shortcuts are observed. First, we extract the passage information of each node by fitting the raw data to the abstracted route graph, then record them in the event log. Next, we obtain transition information between two nodes as bigram by process mining.



(a) Overall graph (b) 1-day graph

Fig. 37: Departure/Arrival inter-node transitions based on the abstracted route graphs. Line thickness of each edge represents the proportion of traffic volume in each segment to the total.

Fig. 37 (a) shows the results of extracting transitions between two nodes for all data under analysis. Fig. 37 (b) shows the results of the same analysis for arbitrarily chosen date to be used as a sample for airspace modeling. Although the major traffic flows are consistent with the edges of the abstracted route graph, transitions with no connections

are observed in the stage abstraction graph created in Chapter 6. In particular, the arrival traffic graphs are complicated by the mesh of approach routes from the south.

In Fig. 37, the major traffic flows can be easily identified because the line thickness of each edge denotes the frequency of occurrence. For arrival routes, the pairs of the adjacent node u_{i-1} and u_i with the high probability of occurrence $P(u_i|u_{i-1})$ are $(18^a, 19^a)$, $(19^a, 6^a)$, $(19^a, 21^a)$, $(16^a, 4^a)$, $(4^a, 3^a)$, $(3^a, 17^a)$, $(17^a, 2^a)$, $(2^a, 1^a)$ from the top. On the other hand, divergences from the middle of the route are also frequent. This is due to the fact the interventions of air traffic control (ATC) frequently cause the arrival aircrafts to take shortcuts. The arrival traffic trace σ_a with the high probability of occurrence are $\langle 18^a, 19^a, 6^a, 13^a, 11^a \rangle^{0.103}$, $\langle 18^a, 19^a, 21^a, 22^a, 23^a \rangle^{0.030}$, $\langle 18^a, 20^a, 21^a, 22^a, 23^a \rangle^{0.022}$, $\langle 15^a, 24^a, 13^a, 11^a \rangle^{0.018}$, $\langle 18^a, 21^a, 22^a, 23^a \rangle^{0.017}$, $\langle 12^a, 24^a, 13^a, 11^a \rangle^{0.013}$, $\langle 14^a, 13^a, 11^a \rangle^{0.011}$, $\langle 15^a, 12^a, 24^a, 13^a, 11^a \rangle^{0.010}$ and traces lower than these eight traces had a probability of occurrence of less than 0.01. Also in the departure traffic trace σ_d, the top traces in order of probability of occurrence are $\langle 5^d, 19^d, 17^d, 20^d, 26^d \rangle^{0.049}$, $\langle 18^d, 2^d, 23^d, 4^d, 1^d, 3^d \rangle^{0.033}$, $\langle 18^d, 13^d, 8^d \rangle^{0.029}$, $\langle 18^d, 2^d, 13^d, 8^d \rangle^{0.028}$, $\langle 18^d, 2^d, 23^d, 10^d, 9^d \rangle^{0.024}$, and the top 24 traces had a probability of occurrence of 0.01 or greater. The superscripts denote the probability of occurrence. In prior literature [9], the superscripts denote occurrences of a case, which varies with the size of the raw data, so we use the probability of occurrence.

The results are well characterized by the arrival and departure routes. The arrival traffic show characteristics such that the main paths exist at the north entry and south entry around adjacent node pairs $(18^a, 19^a)$, $(19^a, 6^a)$, $(19^a, 21^a)$, $(16^a, 4^a)$, $(4^a, 3^a)$, $(3^a, 17^a)$, $(17^a, 2^a)$, $(2^a, 1^a)$ from which the traffic branches off in detail. Thus, there are a small number of high-frequency traces and a large number of low-frequency traces with similar probabilities of occurrence. Departure routes, on the other hand, have many traces with medium probability of occurrence, and are structured in such a way that they branch in a well-balanced manner. This is because the departure routes at Tokyo International Airport are arranged in such a way that they are dispersed according to the direction of the destination.

The total number of pairs of the adjacent node u_{i-1} and u_i for one day to be sample-modeled was 239 (136 pairs for arrival and 103 pairs for departure). In the airport surface model, traces with low frequency of occurrence were removed as noise, but in the

airspace model, probability density distribution parameters were obtained for all patterns in order to model more rigorously.

7.1.2 Probability Density Distribution of Flight Time

In the airspace model, we extract the probability density distribution of flight time by curve fitting. In the airport surface model, we adopted the Erlang family because the probability density distribution of ground travel time showed asymmetry. However, from the two-node transition data obtained in 7.1.1 showed that flight time on each edge varied little and was symmetrically distributed around the mean value on many edges. Therefore, since the Gaussian family is more suitable for airspace data, we adopt the Gaussian distribution for curve fitting.

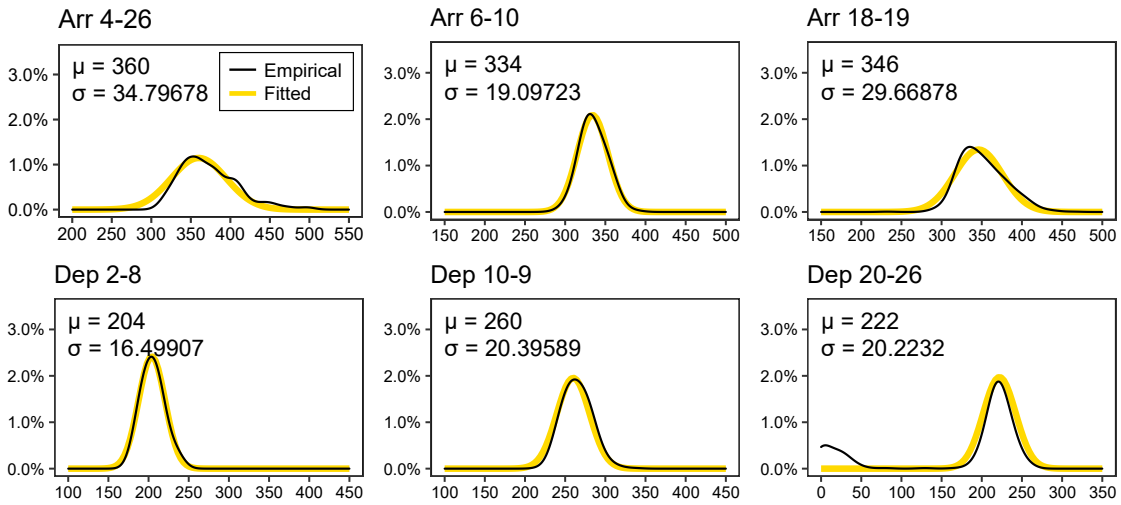


Fig. 38: Probability density of flight times on some typical edges. The travel time between nodes can be calculated from a Gaussian distribution fitted to the real data. The μ and σ in the upper left of each plot indicate Gaussian distribution parameters.

Fig. 38 shows an excerpt of the probability density distribution of flight time and the results of curve fitting on typical edges. It can be seen that the data fit the Gaussian distribution well. We conducted curve fitting on all two-node transition data obtained from the trace analysis, and extract Gaussian distribution parameters. The obtained parameters allow us to reproduce aircraft movement on the model without having real data.

7.2 Airspace Modeling with Object Petri Net

Based on the abstracted route graph and the probability density distribution parameters of inter-node flight time, we create an airspace model using the object Petri nets modeling method presented thus far. We basically follow the method of creating the airport surface model presented in Chapter 5, but since the airspace model has more components and complexity compared to the airport surface model, the aircraft Petri nets and the node Petri nets should be redesigned.

7.2.1 Model design

In the airspace model, as the aircraft passenger model and the airport surface model, we adopt the hierarchical design. The top layer is the airspace Petri net, on which the aircraft Petri nets move around. While the airport surface model had a small network size because it consists of only taxiways and runways, the airspace model, as shown in 7.1.1, had a total of 52 nodes (26 nodes for departure and 26 nodes for arrival), and the number of patterns of two-node transition was 239 even in one day of log data to be used for the sample model. Therefore, directly reproducing the abstracted route graph as a network would require an enormous amount of work. This is a problem faced not only in airspace models, but also when modeling traffic flows that have a vast area of coverage, such as ship data, or traffic flows that are highly random and do not have fixed paths, such as human or animal walking paths.

Here, we take advantage of the characteristics of object Petri nets to prepare a common place "float" that connects all instances of node Petri nets. Aircraft Petri nets are assembled in the float place, and by letting the aircraft itself determine which node to go next, there is no need to construct the abstracted route graph as the complex network diagram entirely in Petri nets.

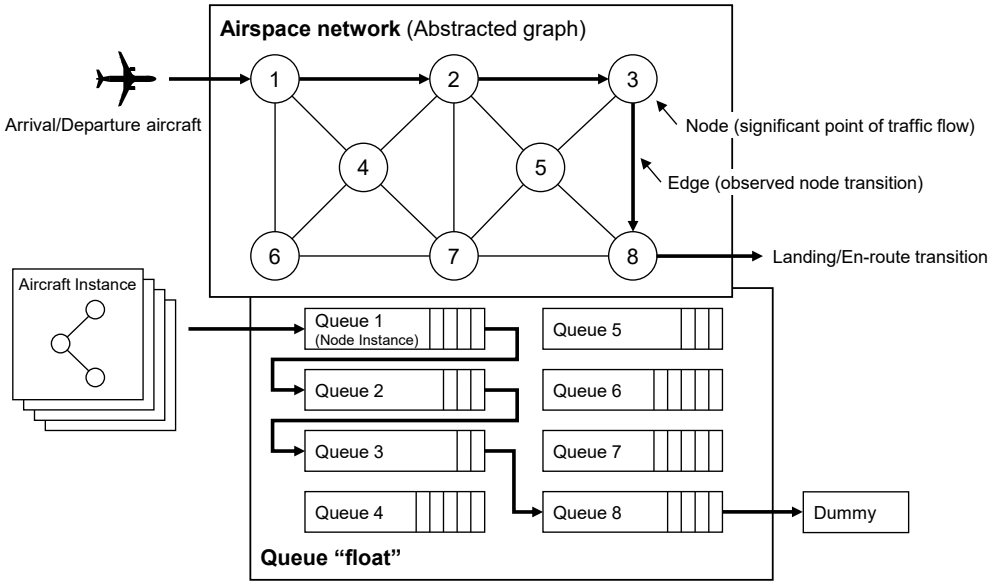


Fig. 39: Airspace model design. Each node belongs to the common place "float". The aircraft determines for itself which node to proceed to next. The aircraft in the figure transitions in the order of node 1→2→3→8 in the network. In the actual Petri net, all nodes are located at the common node "float", and the aircraft instance swims through the float in the order of node 1→2→3→8.

Fig. 39 shows the conceptual design of the airspace model. It has a hierarchical structure, with the upper layer being the airspace network created based on the abstracted route graph. However, the actual Petri net do not have a concrete network form, but a set of node instances on "float" place. Each node instance is assigned a node ID when it is created, and the aircraft instance swims around in the "float" in search of this node ID, which results in the same movement as transitioning through the airspace network. In the airport surface model, when a node transitions, a delay time (taxi time) generated from the Erlang distribution is assigned to the transition, but in the airspace model, the delay time (flight time) cannot be generated by a single node alone because the probability density distribution parameters change depending on the last node through which the aircraft has passed. Therefore, in the airspace model created here, the delay time (flight time) is generated by the aircraft instance itself. Since the aircraft instance has routing information, it knows the previous node u_{i-1} and the current node u_i. The aircraft instance is designed to impose a delay time on itself by Gaussian distribution parameters based on its information. Based on this design, the airspace Petri net generates node instances and aircraft instances at the start of the simulation, and connects all node instances to the "float" place. All aircraft instances are also placed in the "float" place.

After the simulation starts, the aircraft repeatedly moves float⇔node, and when there are no more nodes to move next, it transitions to a dummy node and disappears. The airspace Petri net stops when the number of aircraft instances on the float reaches zero.

7.2.2 Airspace Petri net model

The airspace Petri net consists of a set of node instances, all of which are connected to a common place "float". When an aircraft instance enters the “float” place, it searches for the next node instance it should enter, and if it can enter, it fires immediately the entrance transition of the node instance. Once the aircraft instance enters a node, it obtains the probability density distribution parameters by the previous node u_{i-1} and the current node u_i, and calculates the delay time (flight time) required as a penalty. The penalty is added to the timer of the aircraft instance, and the timer counts down as the simulation proceeds. The node instances are composed of queues, similar to airport surface models, and operate as FIFOs. As the aircraft leaves the node, it checks its timer value and if the timer is greater than or equal to 1, it waits; if it is 0, the aircraft leaves the node and returns to the “float” place to search again for the next node to enter.

This behavior takes advantage of object Petri nets. In object Petri nets, instances can be replicated from the base Petri nets and each can have its own individual properties. Since the aircraft instance knows which node to go next from the route information, it can generate its own flight time required for transitions between nodes, and can reproduce the inter-node transition by probability density distribution by having it stay in the queue until the timer expires.

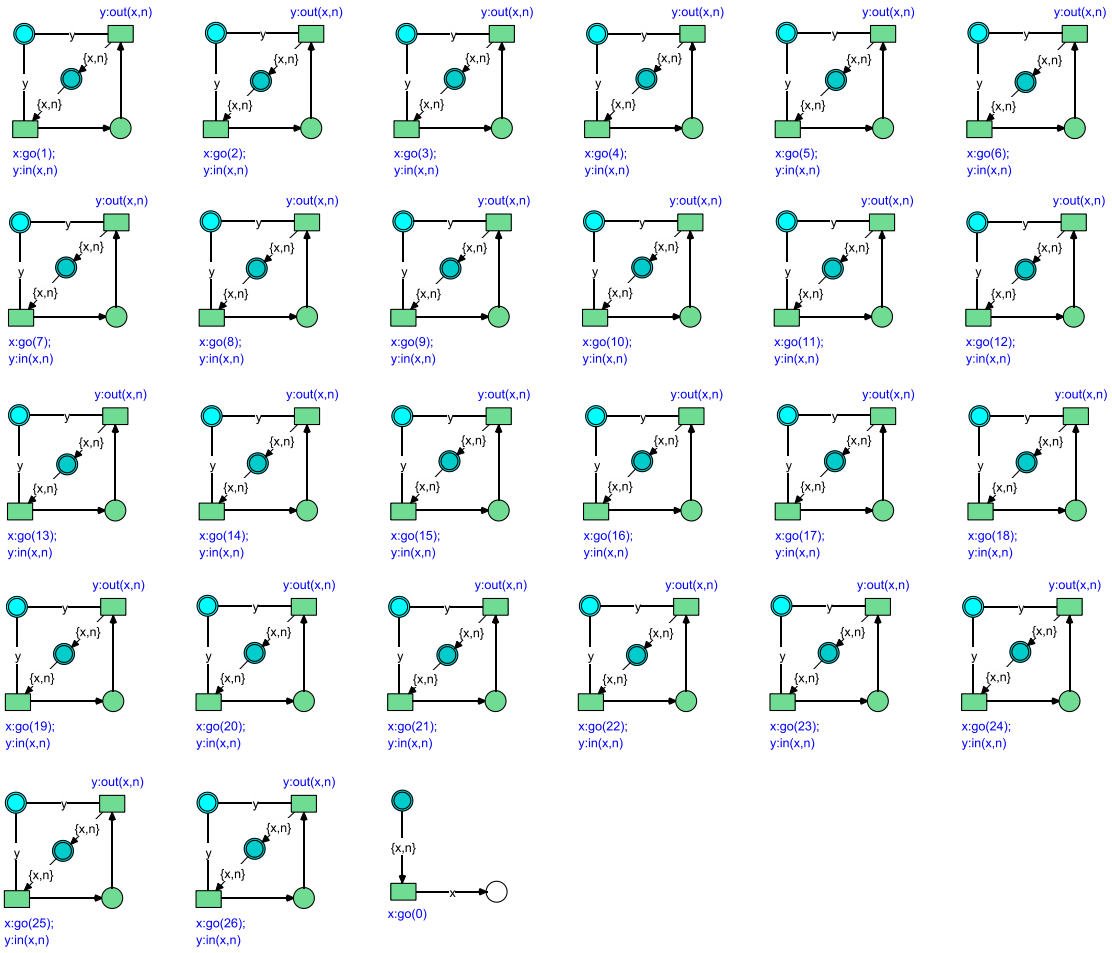


Fig. 40: Floating queues. Twenty-six node instances and a dummy node are located, and all connected to the common place “float” (dark cyan place). An aircraft instance in float cannot enter nodes unless it can simultaneously fire its own transition with method ‘:go($\boldsymbol{n}$)’ and the method ‘:in($\boldsymbol{x}$,$\boldsymbol{n}$)’ provided at the entrance of each node. Note that $\boldsymbol{n}$ denotes the node ID and $\boldsymbol{x}$ denotes the aircraft instance itself. When an aircraft exits the current node and transitions to the next node, it fires the method ‘:out($\boldsymbol{x}$,$\boldsymbol{n}$)’ at the exit and returns to float again.

Fig. 40 shows node instances of the actual airspace model for the arrival traffic. Although there appear to be only 26 independent node instances in a row, they are all connected to a common place “float”, and if the next node to proceed to cannot be identified, the aircraft stays at the “float”. For example, if a node that does not exist in the node set $U = \{1^a, 2^a, \cdots, m^a, 1^d, 2^d, \cdots, n^d\}$ is in the routing information of the aircraft instance, or if there is more than one way for the node to go next, the aircraft will not be able to fire the next transition and will remain in the float. If the aircraft instance in the float place continues to persist during the simulation process, then there is some error in the model. The cause may be an error in the routing information of the aircraft instance,

or a missing node instance that should be placed in the float. The simulation completes when the number of aircraft instances on the float in the airspace Petri net reaches zero; if there are any instances remaining in the float, the simulation is abnormal. In this case, since an abnormality is latent somewhere in the model, it is necessary to identify and correct the causative node or route information by checking method with process mining presented in Chapter 4.

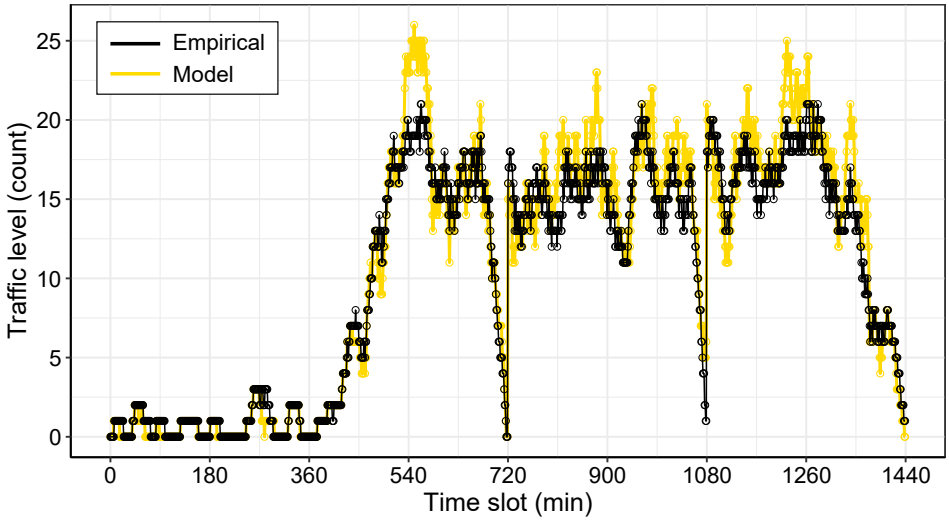
As the aircraft swims around the float and nodes, it records its transition information as an event log. This event log is used for conformance checking and model evaluation through process mining.

7.3 Model Accuracy Analysis and Evaluation

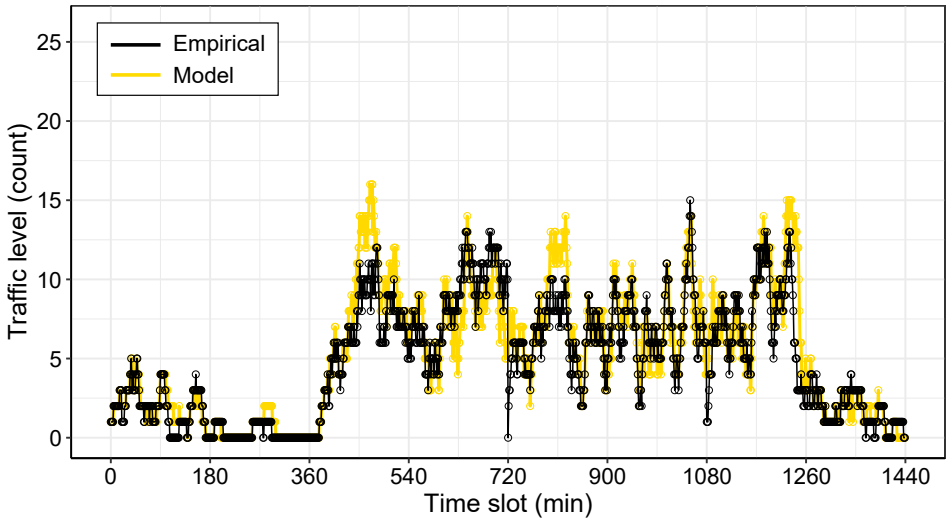
After the simulation is completed, we evaluate the model. As noted in 1.1, whether the model is giving the desired performance depends on its objective. Mesoscopic models have the advantage of being able to significantly downsize large amounts of raw data while still preserving the information needed for analysis. The information that is important in the airspace model is the flight time accuracy of each aircraft and the throughput of the airspace. Therefore, we evaluate the traffic volume in the target airspace and the flight time of each aircraft. Note that the evaluation is to be conducted after completing the conformance checking by process mining as presented in Chapter 4, confirming that the model is working as designed, and assuring that the event logs do not contain any incorrect information.

7.3.1 Airspace traffic volume validation

While traffic volume in the airport surface model was the number of aircraft present on the airport surface at a given moment [58] [54], in the airspace model, we count the number of aircrafts flying in the target airspace at a given moment. Here, the time axis is divided into slots of fixed time (60 seconds), and the number of in-flight aircrafts in the airspace is counted in each slot. Note that simulation results are divided into arrival and departure aircrafts and evaluated separately.



(a) Arrival traffic level



(b) Departure traffic level

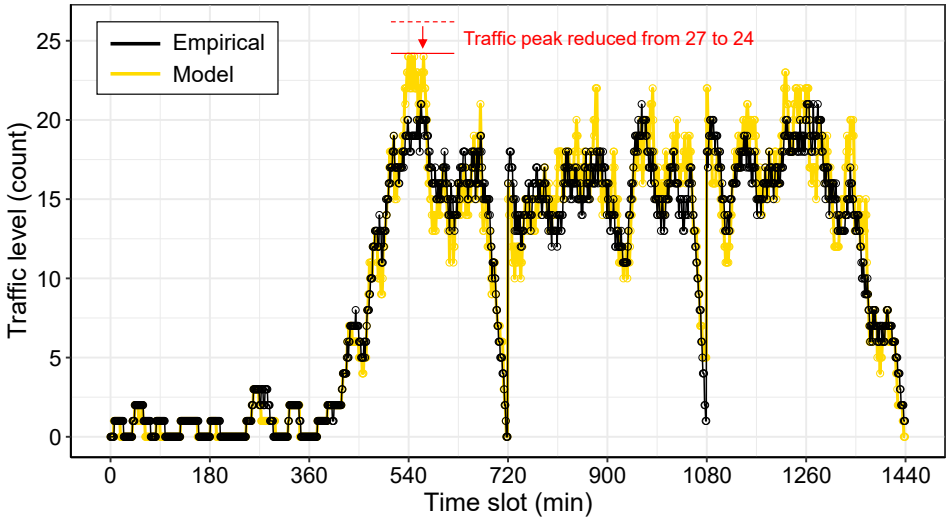
Fig. 41: Airspace traffic level. The size of the queue assigned to each node is set to 5. The x-axis is represented in time slots of 60 seconds (=one minute).

Fig. 41 shows the results of traffic volume comparisons for arrival and departure aircrafts. For both arrivals and departures, the model's simulation results show good tracking compared to the raw data, even though the selection of parameters for the abstracted route graph leaves room for optimization. In particular, in arrival traffic, there is a time period when traffic volume drops around time slot 720 and 1080 (correspond to 12:00 and 18:00), and the model is able to reproduce almost the same drop points. On the other hand, there are some areas where traffic volumes in the model exceed the actual for

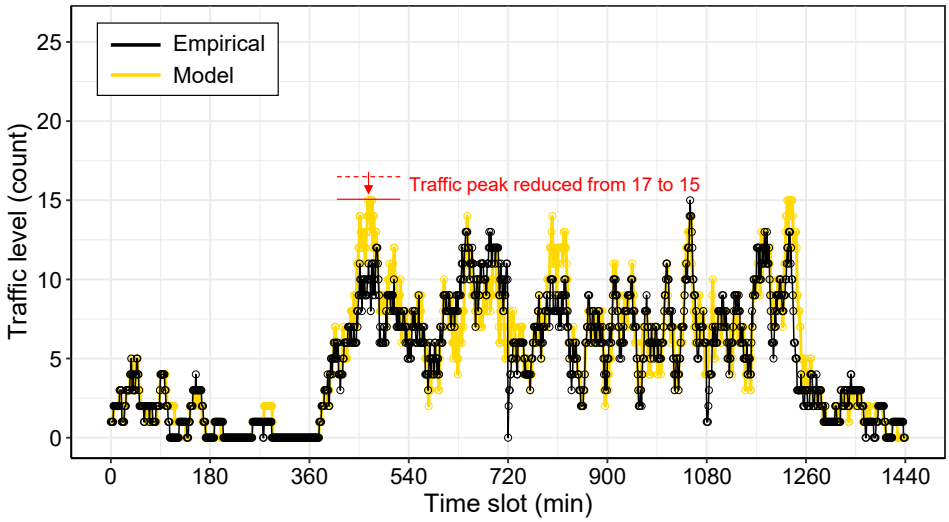
both arrival and departure aircrafts. For example, in the vicinity of time slot 540 (9:00) for arriving aircraft, the model shows around 24 aircrafts while the actual data is around 20 aircrafts. Similarly, for departures, there are time slots where the modeled traffic volume is higher than the actual volume. There are two possible reasons for this:

- **Queue capacity overflow:** Since this airspace model uses common node instances with object Petri nets, the queue capacity $l_q = 5$ at all nodes. This value is set by considering the length of the edges of the abstracted route graph and the horizontal separation in the terminal area. If more inputs than the queue capacity arrive at once, the queue overflows. Similar events have been observed in airport surface traffic flow simulations, where when queue lengths exceed queue capacity, retention accumulates and affects subsequent time slots. In the airspace model, the number of aircrafts that can fly simultaneously on an edge is equal to the capacity of the queue in the node Petri net. Therefore, if the queue overflows, the aircraft instance will be stuck in "float" place. However, we assume that the separation is well arranged in actual airspace, and air traffic control operations are well performed without overflows even during busy times.

  Fig. 42 shows the simulation results when the queue capacity is increased to $l_q = 6$ for confirmation. Although the queue capacity at each node is increased by only 1, the overall congestion in the graph is alleviated and the overhead at the time of maximum congestion is reduced. The actual queue capacity depends on the distance between each node in the abstracted route graph. If further accuracy is desired, the queue capacity should be calculated for each node according to the distance between nodes and the horizontal separation. The airport surface model is constrained by taxiway length. Since the number of aircraft that can be stored on each taxiway is naturally determined by the length of the aircraft and the required separation, the capacity limit is determined to some extent when the airport surface is graphed. On the other hand, the airspace model is based on an abstracted route graph that integrates multiple routes, and unlike the airport surface model, it allows for detour routes and altitude differences due to ATC instructions, which provides some flexibility in queue capacity constraints.



(a) Arrival traffic level



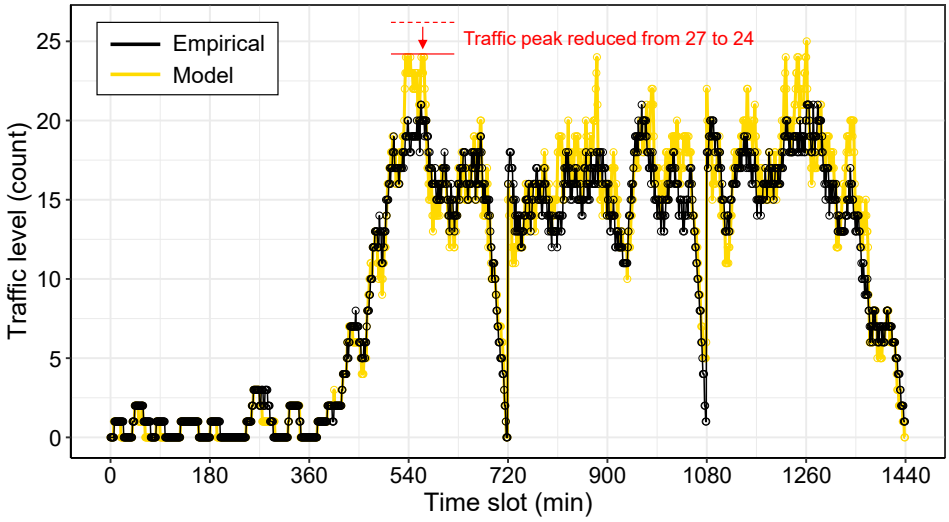
(b) Departure traffic level

Fig. 42: Airspace traffic level (queue size: 6)

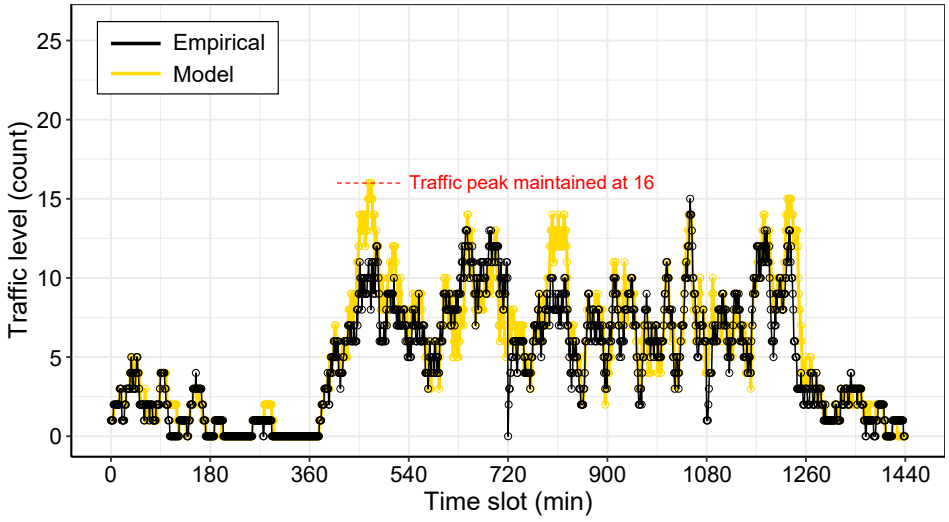
By comparing the model's output with the raw data, the queue capacity of each node is related to the throughput of the airspace to some extent. Therefore, how much the queue capacity can be expanded determines the throughput of the airspace. When the shortest route to a destination is established, air traffic flow is more efficient because fuel consumption and CO_2 emissions can be reduced when the number of aircraft is small. However, the shortest routes reduce queue capacity, resulting in lower throughput of airspace.

However, that unlimited expansion of queue capacity does not necessarily

improve the tracking of raw data. For example, Fig. 43 shows the simulation results when the queue capacity is experimentally doubled to $l_q = 10$. The overhead at maximum congestion of the arrival traffic is almost the same as when $l_q = 6$, although the accuracy is improved over when $l_q = 5$. For departures, there is almost no change between $l_q = 5$ and $l_q = 10$. Therefore, although accuracy can be improved by adjusting queue capacity, it can be said that the error at maximum congestion is not due solely to queue overflow.



(a) Arrival traffic level



(b) Departure traffic level

Fig. 43: Airspace traffic level (queue size: 10)

Tab. 9 shows the maximum error, mean error, and standard deviation between the raw data and the model at the traffic level for each queue size. For the initial value of $l_q = 5$ the mean error is 1.175 for arrivals and 0.973 for departures, which is generally within the range of about one aircraft for each time slot. As l_q is increased, the error becomes smaller for the arrival aircraft, but conversely, the error becomes larger for the departure aircraft.

Tab. 9: Comparison results between raw data and model output at traffic level

Route	Queue size	Maximum error	Mean error	Std. dev.
Arrival	5	7	1.175000	1.380393
	6	6	1.147222	1.337179
	10	6	1.105556	1.258293
Departure	5	9	0.973611	1.301974
	6	9	0.979861	1.290030
	10	10	0.994444	1.313381

- **Runway alternation:** In normal airport operations, the used runway is changed depending on the wind direction or other factors. At large airports with multiple runways, such as Tokyo International Airport, the change of runway has a significant impact on arrival and departure routes, and is a source of error between simulation and actual operations. The results of trace analysis for the arrival traffic shows that the runway direction changed around time slot 360 (6:00). The final approach phase of the traces for the southbound approaching aircraft had been $\langle 14^a, 13^a, 11^a \rangle$, but from around time slot 360, the phase changes to $\langle 22^a, 20^a, 9^a \rangle$. As a result, both the aircrafts approaching from the north and south use common segments, which affected the following aircraft in the simulation and caused delays. In actual operations, however, congestion was mitigated by ATC instructions, which probably caused the error between the simulation and actual data. In actual ATC operations, runways are not suddenly switched without prior arrangement. Since the wind direction is predictable, arrangement should be made in advance to minimize the impact on operations before the runway is changed. Therefore, the actual data shows less congestion than the simulation.

Although there are errors between actual operations and the model output for traffic peaks during maximum congestion, the errors converge quickly, and the model traffic volume follows the actual data in a short time. The airspace model reproduces the traffic flow well and retains the information needed for analysis even after downsizing by mesoscopic modeling.

7.3.2 Flight time validation

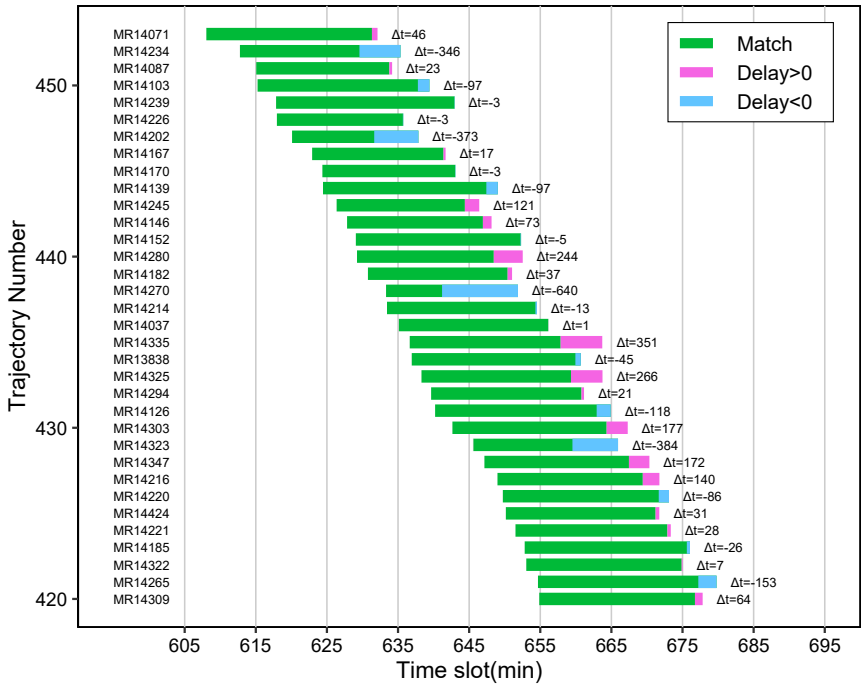
Next, we evaluate the flight time of each aircraft. The flight time is defined as the time between the appearance and disappearance of an aircraft in the real data, and is compared with the appearance and disappearance times in the simulation. In the simulation by the airspace model, only the appearance time is given to the aircraft instance as an individual property. Therefore, if the “float” place is congested at the original appearance time and the aircraft cannot enter the first node, the appearance time is delayed. If the preceding aircraft is stuck in the node and it takes time to transition to the node, the accumulated delay will be the delay of the disappearance time.

The comparison results of flight times between the raw data and the simulation are shown for arriving aircraft in Fig. 44 and for departing aircraft in Fig. 45. Then Fig. 44 (a) and Fig. 45 (a) show the results of the comparison of flight times during the time period when good accuracy was obtained in the traffic volume comparison. The simulation reproduces the actual operation generally well, with minimal errors for both departing and arriving aircraft. On the other hand, Fig. 44 (b) and Fig. 45 (b) show the results of the flight time comparison during the time period when the model outperformed the actual in the traffic volume comparison in 7.3.1. It can be seen that the operating time increases in the model, affecting the following aircraft. For arriving aircraft, delays begin to appear around time slot 510 (8:30), and about 10 more aircraft are affected by the delays that follow, with the delays gradually increasing. For departures, delays also begin to be noticeable around time slot 800 (13:20), and Fig. 41 (b) shows that we can observe there a gap in traffic volume just at that time.

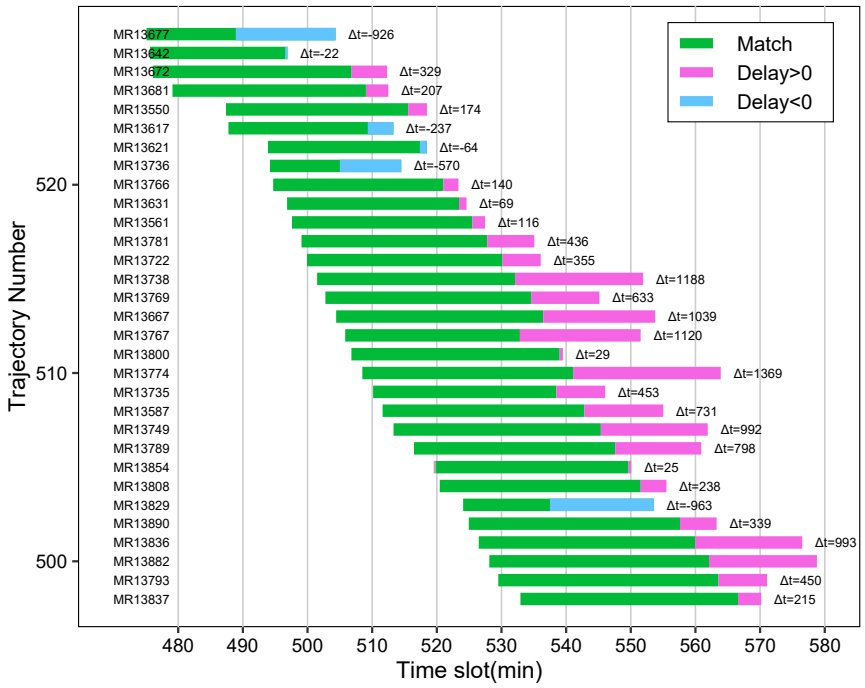
Unlike taxiing aircraft on the ground, aircraft in flight cannot stop in mid-air, so if the preceding aircraft is stuck and cannot proceed, the following aircraft has to either

enter a holding pattern and wait for the congestion to ease, or make a detour. Holding aircraft creates additional delay time and has an additional impact on the following aircraft. In practice, however, few aircraft are forced to hold in mid-air. Even in congested airspace, the situation is well coordinated through the ATC intervention, and efforts are made to prevent the deterioration of punctuality. Let us focus on the green bars in Fig. 44 (b), although slightly longer than in Fig. 44 (a), is the same length for almost all aircraft, with no significant bias. It is thought that ATC interventions are made to avoid negative effects on congestion by adjusting the separations by radar vectors, adjusting the speed of following aircraft, and applying narrower separation with tight margin.

The created airspace model does not reproduce the flexible spacing adjustments made by ATC intervention, making subsequent aircraft more susceptible to congestion and changes in runways in use. However, the duration of the impact is slight, and the model shows considerable accuracy at other times of the day. Thus, as with traffic volume assessment, the airspace model reproduces aircraft behavior to some extent in the assessment of individual aircraft operating times, and the necessary information is retained even after downsizing by mesoscopic modeling.

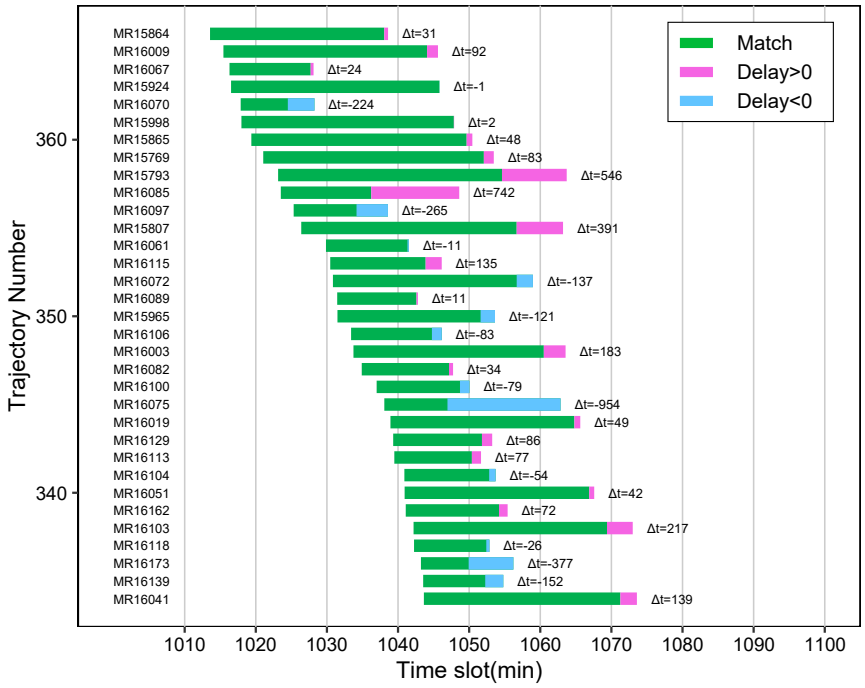


(a) Time periods showing good consistency (Arrival)

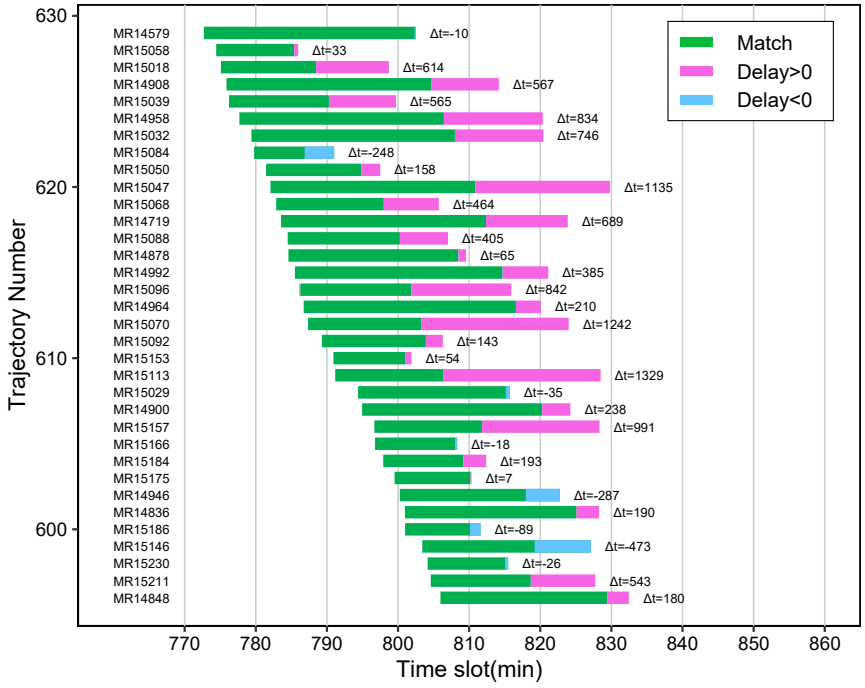


(b) Time period showing deviation (Arrival)

Fig. 44: Flight time comparison between model and actual data for arrival traffic. Green bars represent the model matches the real data. If the model took longer flight time relative to the real data, these are indicated by magenta bars; and if the model took less time relative to the real data, these are indicated by blue bars. The strings on the right side of the y-axis indicate the flight ID.



(a) Time periods showing good consistency (Departure)



(b) Time period showing deviation (Departure)

Fig. 45: Flight time comparison between model and actual data for departure traffic. Green bars represent the model matches the real data. If the model took longer flight time relative to the real data, these are indicated by magenta bars; and if the model took less time relative to the real data, these are indicated by blue bars. The strings on the right side of the y-axis indicate the flight ID.

7.4 Summary

In this chapter, we created and evaluated an airspace model based on the abstracted route graph. Since the airspace models are complex in composition, unlike airport surface models, they are difficult to construct using only information on network composition. Therefore, here we took advantage of the characteristics of object Petri nets and allowed the aircraft itself to swim among nodes in “float”, thereby reducing the amount of work required to create the airspace network and improving the efficiency of model implementation.

In the airspace model evaluation, good reproducibility was demonstrated for traffic volume and flight time. This means that the mesoscopic modeling framework developed in this dissertation can reproduce air traffic flow with some accuracy while significantly downsizing large volume data, which demonstrates the effectiveness of the proposed

modeling method. The raw air traffic flow data used in the airspace modeling is several gigabytes in size, but was downsized to a few megabytes by replacing all of the node-to-node movement information with statistical data in the model.

Depending on the scale of traffic flow data and the granularity required by designers for mesoscopic models, the modeling effort required can be enormous. Mesoscopic models originally had the advantage of providing accurate results to some extent with lightweight processing instead of directly handling raw data. However, if modeling itself takes a great deal of time, it would be faster to handle raw data directly to obtain results. Therefore, it is desirable to consider streamlining the modeling process as shown in this chapter or model designing that takes advantage of the characteristics of the modeling language.

Chapter 8

Discussions

In previous chapters, we systematize the necessary techniques for mesoscopic modeling and developed a framework. We also created an airport surface traffic flow model and an airspace traffic flow model and confirmed their effectiveness. Mesoscopic modeling can greatly reduce the amount of data while still maintaining some accuracy. It enables a significant reduction in the time cost required of the model designers. Since our focus in this dissertation is on the modeling methodology itself, we leave room for further development in terms of improving the accuracy of the model and evaluation methods. This chapter describes future issues and measures to further enhance the proposed framework.

8.1 Future research opportunities

To further improve the accuracy of the mesoscopic model in the proposed framework, consideration of vertical elements absorbed during the abstraction process and optimization through parameter tuning are possible. These are not in the scope of this dissertation, so we treat the following items as the future work.

8.1.1 Consideration of vertical elements

Airspace data is more complex because it includes the altitude factor and, unlike airport surface traffic flow, must be considered in three dimensions. In Chapter 6, the differences in altitude factors were absorbed by pre-processing and abstracted route graph creation during the mesoscopic process. However, depending on the degree of abstraction of the mesoscopic model, differences in altitude factors may become unabsorbable. Let us look at the following example.

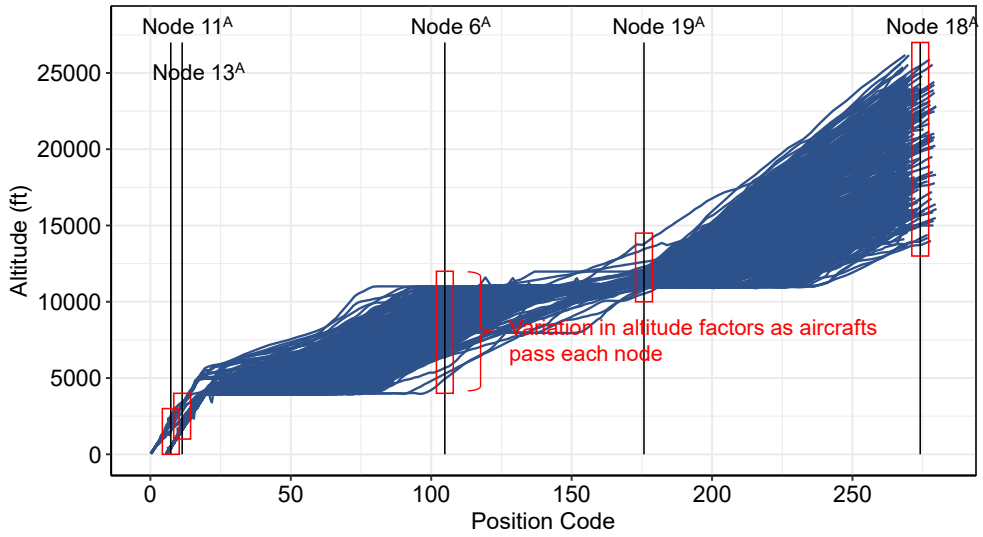


Fig. 46: Vertical plane of the most frequent trace. The position code set on the x-axis is the two-dimensional position information converted to one dimension to align the altitude information of each trajectory on the graph.

Fig. 46 plots the vertical cross section of trajectories for aircraft that flew on the most frequent arrival trajectory $\langle 18^a, 19^a, 6^a, 13^a, 11^a \rangle$ in 7.1.1. Since the destination is the airport, all trajectories will eventually converge on a single point. Furthermore, because the key traffic points are extracted as discrete nodes, the altitude information at the nodes are also concentrated to some extent. For example, at node 19^a, aircraft from various altitudes descend toward a single point and pass at approximately the same altitude. Even if there are differences in height paths on the way to node 19^a, they will eventually arrive at the same point. Therefore, some degree of altitude variation can be treated as one and differences in altitude factors can be ignored by treating the airspace as a broadband conduit. The abstracted route graph absorbs differences in the vertical direction by replacing movement information between nodes with statistical information.

However, if the altitude difference is significantly larger, it needs to be distinguished as a separate conduit. If we look at node 6^a, we see that the trajectories are distributed between 7 000ft and 10 000ft, but if we want to distinguish arrivals within this range, we need to create an abstracted route graph that also takes the altitude difference into account. In the en-route airspace, the same airway can be made bidirectional by setting altitude differences, or treated as a double-track route to increase the capacity of a single route. In this case, clustering only by two-dimensional location information and

absorbing the altitude factor would result in missing information necessary for traffic analysis. Even with terminal flight procedures, it is quite possible for departure and arrival routes to be in close proximity on a plane, depending on the environment around the airport, and if common routes are clustered in trajectory clustering, trajectories that should be distinguished will be merged. If altitude differences are intentionally provided to distinguish routes, the altitude factor cannot be absorbed by abstraction. Even for the same arrival path, there is a mixture of aircraft that pass through the optimal descent path (e.g., continuous descent approach [75]) with good fuel efficiency and those that choose other descent paths. In such cases, their efficiency can be compared by separating the descent paths by altitude and modeling them differently.

To generate clusters while distinguishing differences in altitude, there are several possible methods, such as discriminating trajectories in the target airspace when extracting trajectories in the pre-processing stage, increasing the target dimension to three dimensions in space partition using k-d-b tree, and considering the vertical direction in the distance scale parameter $minLns$ of trajectory clustering algorithm. In particular, in trajectory clustering process, by including an altitude factor when calculating the distance between segments, segments that are close in plane but have altitude differences can be separated. In this case, since the vertical distance is shorter than the horizontal plane distance, the vertical distance must be normalized and scaled to the horizontal plane distance.

8.1.2 Parameter optimization

The air traffic flow model presented in this dissertation showed high accuracy while downsizing through abstraction but leaves room for further optimization. In particular, in the creation of the airspace model shown in Chapters 6 and 7, there are numerous parameters, and changing just one of them will change the form of the model. While this dissertation focuses on establishing a mesoscopic model building method, future work includes parameter tuning and optimization. The parameter sets used in abstracted route graph creation and mesoscopic modeling are as follows:

Parameter set for mesoscopic modeling

α	Parameters that control the degree of approximation of the alpha shape to the boundary of a point set
ε_r	Reference distance for down-sampling in the Ramer-Douglas-Peucker algorithm
ε_d	Reachable radius from any point in DBSCAN algorithm
ε_t	Reachable radius from any segment in TRACLUS algorithm
k	Size of leaf nodes of the k-d-b tree for the data set
l_q	Queue capacity of each node. Adjusting the queue capacity prevents overflow due to sudden inflow, but it should be set to a number that takes into account the route length and the size of the moving object.
m	Delaunay triangle mesh size to be set for envelope polygons in alpha shape output
$minLns$	Minimum number of segments to form a core cluster in TRACLUS algorithm
$minPts$	Minimum number of points to form a core cluster in DBSCAN algorithm
$w_\perp,\ w_\parallel,\ w_\theta$	Weighting coefficients for the inter-segment distance in the TRACLUS algorithm. The weighting is determined for each of the three distances $d_\perp,\ d_\parallel,\ d_\theta$ that consist of the inter-segment distance. In this dissertation, all are set to 1.0.
$\lambda,\ n,\ \sigma,\ \mu$	Probability density distribution parameters (Erlang distribution: λ, n; Gaussian distribution: σ, μ) representing movement information between adjacent nodes in an abstracted route graph

The optimal degree of abstraction depends on what the designer wants the model to do. The degree of abstraction can be defined as the fineness of the network model. For example, a smaller parameter k in k-d-b tree results in a larger number of leaf nodes and finer segmentation of the space. Since the segment size input to trajectory clustering is determined by this segmented space size, the smaller k is, the smaller the deviation from the real data. Therefore, setting $k = 1$ will minimize the error, but that would result in the same granularity as the raw data. It is necessary to determine parameters that minimize

the error from the real data while maintaining the granularity of the mesoscopic model according to the abstraction level.

8.1.3 Development of model evaluation methodology

Whether the created mesoscopic model is good for the analyst is determined by whether the model reproduces the traffic flow well and is useful for understanding the current situation and simulating improvement measures to eliminate congestion. If the overall accuracy in reproducing traffic flow is good but the errors are biased, it is not a good model for the analyst. On the other hand, even if the accuracy is moderate, it may be useful to the analyst if the error is constant throughout the target space.

Let us consider, for example, the granularity of an abstraction graph. When an abstracted route graph is generated from a certain traffic flow, the finer the nodes of the network, the more granular the model. However, a high granularity, in turn, makes it difficult to keep the overall error amount constant. A rough graph with low granularity tends to have a relatively large amount of error in each segment, but because of the high level of abstraction, it is easier to keep the overall error constant. Therefore, a high level of abstraction does not necessarily lead to a good mesoscopic model. Regardless of the level of abstraction, a graph that well represents the character of the traffic flow is a better model.

In this dissertation, we conducted the evaluation from the perspective of analyzing airport surface traffic flow and airspace traffic flow, but the evaluation scale varies depending on the targeted traffic flow. Therefore, after mesoscopic models are created, the challenge is to develop a method to quantify the goodness of fit for the analyst.

8.1.4 Selection of clustering algorithm

The core of this framework is clustering technology, which abstracts raw data by merging similar paths. There are many clustering algorithms as described in 6.3, and they are mainly classified by distance measure into density-based, trajectory-based, time-series-based, and similarity-based.

In process mining, there is also clustering by trace [76]. Since this is the similarity of the event series contained in the traces, it can be classified into the edit distance of

similarity-based clustering. In this dissertation, we employed density-based and trajectory-based clustering, which are suitable for air traffic flow, but it is important to explore the most suitable clustering algorithm for different types of traffic flow.

8.2 Summary

Since the mesoscopic modeling framework proposed in this dissertation focuses primarily on method development, future works remain regarding optimization of model accuracy and evaluation methods. The parameter tuning described in 8.1.2, which seeks better parameters by feeding back errors, is closely related to the evaluation method described in 8.1.3. Therefore, to further enhance the proposed framework, research on the extraction of optimization models, parameter tuning and development of evaluation methods by applying machine learning is necessary when aiming to improve accuracy.

Chapter 9

Conclusions

9.1 Output of this dissertation

9.1.1 Summary of the output

In this dissertation, in order to represent the abstracted traffic flow as the mesoscopic model, we developed a method for creating a concurrent system in which each moving object operates sequentially and in parallel, competing for resources and operating sequentially. We selected object Petri nets as the model description language and showed a method to combine a large number of small subnets to construct a hierarchical, large-scale model. In addition, a specification conformance check method using process mining technique was developed to demonstrate that the model is operating correctly according to the design. The method can show that the model behaves correctly as intended by the designer and that at least the inspected model output contains no errors and is valid for analysis.

Next, we developed a mesoscopic modeling framework to abstract the traffic flow data, leaving the major streams of the network at a level where bottlenecks and stagnation sites can be identified, and to represent the data as a graph consisting of nodes and edges. In modeling, process mining techniques were applied to efficiently identify traffic patterns, remove noise, and extract statistical information, thereby significantly reducing the required workload. In this dissertation, air traffic flow data was used to model airport surface traffic flow and airspace traffic flow. Compared to airport surface traffic, aircraft behavior in the airspace is more complex due to the lack of a fixed trajectory and the altitude factor. Therefore, two approaches, density-based clustering and skeleton extraction, and space partitioning and trajectory clustering, were used to build a framework for generating graphs suitable for mesoscopic modeling from air traffic flow

data.

Finally, we evaluated that the created model has a certain degree of reproducibility compared to the real data, while significantly reducing the amount of data. The final output mesoscopic model of air traffic flow downsized the amount of data to a few tenths of the original data, while still reflecting the characteristics of the traffic flow and allowing the model designers to achieve their objectives.

9.1.2 Answers to research questions

To the research questions Q1-Q3 presented in 1.1.2, this dissertation provides the following answers A1-A3:

A1 Modeling with down-sampled data and its accuracy: The mesoscopic models presented in this dissertation discretize continuous traffic flow data to a level where it can be treated as an event log, ultimately significantly reducing the amount of data. On the other hand, the airport surface traffic flow model and airspace traffic flow model developed in this dissertation have achieved a certain level of accuracy, as shown in the evaluation results in 5.4 and 7.3. As mentioned in 8.1.2, the accuracy required for the model is determined by the model designer. If the accuracy is insufficient, the designer can modify parameters to make the model more microscopic, i.e., closer to the raw data. The method presented in this dissertation allows adjustment while maintaining a balance between accuracy and granularity. Thus, we have shown that the designer can obtain the desired results even if the amount of data is reduced to some extent by down-sampling the continuous data instead of dealing with it directly.

A2 Selectable modeling abstraction: The method presented in this dissertation allows the model designer to select the level of abstraction according to the level of granularity desired, rather than simply discretizing the data in order to reduce the amount of data. If the model designer wants to know in detail where congestion occurs and the trajectory of each moving object, the granularity can be set high; if the designer simply wants to know the large flows, the granularity can be set low. The level of abstraction can also be changed locally depending on traffic density. This can be fine-tuned by selecting parameters such

as space partition or density-based clustering in an abstracted route graph framework. Thus, the model can be selected to be either macro- or micro-oriented, or even partially micro-oriented with other parts being macro-oriented.

Mesoscopic models can also be created that have the advantages of both macro and micro models. Micro models in existing studies (e.g., agent simulations), while capable of accurate simulations, require workloads to reproduce huge traffic flows. On the other hand, macro models (cell models, queueing models, etc.) reduce the amount of data and workloads required for simulation, but specific detailed features of the traffic flow tend to be omitted in the modeling process. Mesoscopic models can express throughput and traffic capacity estimation, which macroscopic models are good at, and identification of congestion area, which microscopic models are good at, in a single model.

A3 Checking model correctness: In this dissertation, we have developed a method using process mining techniques to check the safety, liveness, and fairness of model behavior. The application of software verification/testing concepts to simulation models and output event logs is a new perspective. To ensure that the model met specifications, we used process mining to detect incorrect things for the model (e.g., latent bugs in the model). Since the method presented in this dissertation can check only the event logs output by the model, there is no need for the model to be equipped with a special inspection mechanism. It is not model-specific, but can be used for mesoscopic models in general.

9.2 Suitability of the proposed method

The mesoscopic modeling approach presented in this paper is not suitable for all purposes due to its nature of extracting and abstracting features of traffic flows. Some cases are suitable, and some are not, depending on the modeling target. Therefore, before modeling, it is necessary to determine whether this method is suitable for the target traffic flow and the purpose of modeling (desired output). Here we show examples of cases where this method is suitable or unsuitable as below:

Suitable case: Suitable for traffic flows where there is a certain degree of common flow

in the trajectories of moving vehicles and major routes exist. It is also suitable for cases where each mobile unit has a common destination (a specific geographic point such as an entrance/exit of a building, a bay port, an airport, etc.) and passes through the area at a high frequency.

Unsuitable case: When moving objects move in disorder and trajectories are uniformly distributed, both density-based and trajectory clustering are unsuitable because it is difficult to form clusters. In addition, because of the abstraction graph, it is not possible to reproduce dynamics such as braking, acceleration/deceleration, and so on.

9.3 Applications

While this dissertation dealt with air traffic flow data and its modeling, the mesoscopic modeling framework of traffic flow presented here can be applied to any spatial data that can be represented by a network model. Based on the "suitable case" mentioned in 9.2, we show some examples as follows:

9.3.1 Traffic flow data

- **Existing traffic modes:** Traffic flow data for vessels and vehicles have been widely used for a long time, and there are a variety of methods for their analysis; Dobrkovic et al [64] and Filipiak et al [65] used vessel AIS data to perform space partition and also used genetic algorithms to extract key traffic flow points of the traffic flow. Besse et al [60] also perform trajectory clustering with SSPD on cab trajectory data. These are applicable to the mesoscopic modeling framework of this dissertation.

- **Next-generation air mobility traffic flow:** The air traffic flow data used in this dissertation is derived from aircraft track data under air traffic control, but it is predicted that the next-generation air mobility [77] [78], called urban and advanced air mobility (UAM/AAM), will emerge and further overcrowd traffic in the airspace. It is necessary to consider air traffic that integrates existing air traffic control and next-generation air mobility, but the UAM/AAM may have a different traffic flow

from the current air traffic flow because it has a smaller margin against ground surfaces, obstacles, and other aircraft than existing aircraft, and its behavior is steeper. Therefore, the existing air traffic flow and UAM/AAM will require variable modeling granularity. The abstraction methods with density-based space partitioning and trajectory clustering described in Chapter 6 are useful.

9.3.2 Observation data of natural phenomena

- **Wind direction and typhoon track:** Open data on wind direction and typhoon tracks are provided by the JMA (Japan Meteorological Agency) [79], ECMWF (European Centre for Medium-Range Weather Forecasts) [80], and NOAA (National Oceanic and Atmospheric Administration) [81]. For example, for typhoon trajectories, though the distance measure used is different from the method of this dissertation, Kordmahalleh et al. used the distance between time series data (DTW [61]) and used the most similar hurricanes for network training [82]. Tupper and Anderson also perform time series wind speed clustering by band distance [83], which is an extension of band depth [84].

9.3.3 Human and animal tracking data

- **Pedestrian tracking data:** An example of available data is the MOTChallenge data [85]. MOTChallenge is a benchmark for machine learning person tracking algorithms, but the publicly available dataset includes video data for machine learning and motion detection data for algorithm evaluation are also included. The pedestrian data in a shopping mall [20] discussed in Chapter 1 is also data for tracking people from camera images, as is the MOTChallenge. Pedestrian data in station premises and plazas are different from those with fixed routes, such as roads, where the randomness of movement is high, and it is difficult to extract the major traffic flow simply by plotting it directly. In the method presented in this dissertation, the space is divided on a density basis and the trajectories of each pedestrian are clustered. Thus, it is possible to extract the major traffic flows hidden in the data.

- **Animal Tracking Data:** Mesoscopic models can also be created from animal

tracking data using the methods in this dissertation. Examples of available data include sheep and goat track data [86] and fish track data [87] . Deer track data are also used in the evaluation of the TRACLUS trajectory clustering [59].

- **Sports Analytics:** Data analytics using track data from various sensors has also been introduced in the sports field, and is used for various statistics such as player performance measurement and pass network analysis. Analysis of the range of player actions and pass networks can be done by downsizing and abstracting the raw data, so the method in this dissertation is applicable. While there are many commercial datasets for track data, there are also many open data sets available for free use, such as soccer [88], ice hockey [89], basketball [90] [91]. Note that image recognition and person detection are required to acquire track data from [90] [91].

9.4 Contributions

The proposed mesoscopic model focuses on reproducing the general flow of traffic flow. It is suitable not only for reproducing the traffic flow as a phenomenon on a computer, but also for using the model to obtain an overview of the traffic flow and to repeat the simulation many times with varying conditions. For example, in the case of vehicular traffic flow, simulations could include new detour roads, changes in traffic signal patterns at intersections, and changes in the surrounding environment that would affect traffic flow (e.g., new shopping malls or train stations). In the case of air traffic flow, examples could include new flight procedures and airways, changes in altitude restriction or airspace design. In the case of human flow, examples could include structural changes in train stations, confirmation of corridor design in new buildings, and simulation of evacuation routes in the event of a disaster.

As mentioned at the beginning of Chapter 1, when measures are introduced at a cost in the above cases, there is a risk that the investment will be wasted if they are not effective. If there is no confidence that the benefits will be obtained reliably, then the decision to introduce the measure cannot be made. Therefore, when making costly changes to traffic flow, it is necessary to try various settings in detail to find the

configuration that will produce the greatest effect. Simulations must be repeated, taking into account not only the transportation infrastructure to be introduced, but also the surrounding environment and weather conditions. Time and effort are required at the stage prior to introducing measures, which delays the manifestation of effects and results in the loss of public benefits.

The mesoscopic model we proposed in this dissertation allows lightweight and fast simulations under different conditions. It is expected to be applied to various traffic modes and to reduce the workload of analysts in introducing measures, etc.

Bibliography

[1] M. Bando, K. Hasebe, A. Nakayama and Y. Sugiyama, "Dynamical model of traffic congestion and numerical simulation," Physical Review E, 51(1), pp. 1035-1042, 1995.

[2] F. Nejadkoorki, K. Nicholson, I. Lake and T. Davies, "An approach for modelling CO2 emissions from road traffic in urban areas," The Science of the total environment. 406(1-2), pp. 269-278, 2008.

[3] C. Linton, S. Grant-Muller and W. F. Gale, "Approaches and Techniques for Modelling CO2 Emissions from Road Transport," Transport Reviews, Vol. 35(4), pp. 533-553, 2015.

[4] H. Gould, J. Tobochnik and W. Christian, "An introduction to computer simulation methods 3rd ed.," Addison Wesley, 2006.

[5] H. Mahmassani, J. C. Williams and R. Herman, "Performance of urban traffic networks," Proceedings of the 10th International Symposium on Transportation and Traffic Theory, pp. 1-20, 1987.

[6] A. Loder, L. Ambühl, M. Menendez and K. W. Axhausen, "Understanding traffic capacity of urban networks," Scientific Reports Vol. 9, 16283, 2019.

[7] Federal Highway Administration, "Guidance on the Level of Effort Required to Conduct Traffic Analysis Using Microsimulation," 2014.

[8] W. Burghout, H. N. Koutsopoulos and I. Andréasson, "Hybrid Mesoscopic-Microscopic Traffic Simulation," Transportation Research Record Journal of the Transportation Research Board, Vol. 1934(1), pp. 218-255, 2005.

[9] W. M. P. v. d. Aalst, "Process Mining: Discovery, Conformance and Enhancement of Business Processes," Springer-Verlag, Berlin, 2011.

[10] D. Chowdhury, L. Santen and A. Schadschneider, "Statistical physics of vehicular traffic and some related systems," Physics Reports, Vol. 329, Issues 4–6, pp. 199-329, 2000.

[11] M. J. Lighthill and G. B. Whitham, "A Theory of Traffic Flow on Long Crowded Roads," Proceedings of the Royal Society of London A, 229, pp. 317-345., 1955.

[12] R. I. Paul, "Shock Waves on the Highway," Operations Research 4, pp. 42-51, 1956.

[13] H. Holden and N. H. Risebro, "A Mathematical Model of Traffic Flow on a Network of Unidirectional Roads," SIAM Journal on Mathematical Analysis, Vol. 26, Issue 4, pp. 999–1017, 1995.

[14] G. M. Coclite and B. Piccoli, "Traffic Flow on a Road Network," SIAM Journal on Mathematical Analysis, Vol. 36, Issue 62005, pp. 1862–1886, 2002.

[15] P. G. Gipps, "A behavioural car-following model for computer simulation," Transportation Research Part B: Methodological, Vol. 15, Issue 2, pp. 105-111, 1981.

[16] O. Derbel, T. Péter, Z. Hossni and M. Benjamin, "Modified Intelligent Driver Model," Periodica Polytechnica Transportation Engineering 40(2), 2013.

[17] S. Wolfram, "A New Kind of Science," Wolfram Media, 2002.

[18] P.-O. Siebers and U. Aickelin, "Introduction to Multi-Agent Simulation," http://arxiv.org/abs/0803.3905 arXiv:0803.3905., 2008.

[19] M. Papageorgiou, C. Diakaki, V. Dinopoulou and Y. Wang, "Review of Road Traffic Control Strategies," Proceedings of the IEEE, Vol. 91, No. 12, pp. 2043-2067, 2003.

[20] D. Brscic, T. Kanda, T. Ikeda and T. Miyashita, "Person position and body direction tracking in large public spaces using 3D range sensors," IEEE Transactions on Human-Machine Systems, Vol. 43, No. 6, pp. 522-534, 2013.

[21] T. Miyamoto, M. Sakamoto and S. Kumagai, "A Survey of Object-Oriented Petri Nets," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E90-A, Issue 10, pp. 2257–2260, 2007.

[22] "The Reference Net Workshop," [Online]. Available: http://www.renew.de/. [Accessed 27 5 2022].

[23] M. Duvigneau, D. Moldt and H. Rölke, "Concurrent Architecture for a Multi-agent Platform," Agent-Oriented Software Engineering III. AOSE 2002. Lecture Notes in Computer Science, Vol. 2585. Springer, Berlin, Heidelberg., 2003.

[24] W. M. P. van der Aalst, et al., "Process Mining Manifesto," BPM 2011. Lecture Notes in Business Information Processing, Vol. 99. Springer, Berlin, Heidelberg, 2012.

[25] "ProM Tools," [Online]. Available: https://www.promtools.org/. [Accessed 27 5 2022].

[26] S. Marelli, G. Mattocks and R. Merry, "The Role of Computer Simulation in Reducing Airplane Turn Time," Aero Magazine 1(1), 1998.

[27] H. V. Landeghem and A. Beuselinck, "Reducing passenger boarding time in airplanes: A simulation approach," European Journal of Operations Research, Vol. 142, No. 2, pp. 294-308, 2002.

[28] M. Schultz, C. Schulz and H. Fricke, "Efficiency of Aircraft Boarding Procedures," Proceedings of the 3rd International Conference on Research in Airport Transportation, 2008.

[29] D. C. Nyquist and K. L. McFadden, "A study of the airline boarding problem," Journal of Air Transport Management, 145 (2008), pp. 197-204, 2008.

[30] A. Steiner and M. Philipp, "Speeding up the Airplane Boarding Process by Using Pre-Boarding Areas," Proceedings of the 9th Swiss Transport Research Conference, 2009.

[31] J. H. Steffen, "Optimal boarding method for airline passengers," Journal of Air Transport Management, Vol. 14, pp. 146-150, 2008.

[32] J. H. Steffen and J. Hotchkiss, "Experimental test of airplane boarding methods," Journal of Air Transport Management, Vol. 18, Issue 1, pp. 64-67.

[33] F. Jaehn and S. Neumann, "Airplane boarding," European Journal of Operational Research, Vol. 244, pp. 339-359, 2015.

[34] M. van den Briel, J. R. Villalobos, G. L. Hogg and T. Lindemann, "America West Airlines Develops Efficient Boarding Strategies," Interfaces, Vol. 35, No. 3, pp. 191-201, 2005.

[35] N. Gupta and D. S. Nau, "On the complexity of blocks-world planning," Artificial Intelligence, Vol. 56 (1992), pp. 223-254, 1992.

[36] R. J. Milne and A. R. Kelly, "A new method for boarding passengers onto an airplane," Journal of Air Transport Management, Vol. 34, pp. 93-100, 2014.

[37] K. Uehara and K. Hiraishi, "Process mining approach for the conformance checking of discrete-event simulation model," 2019 58th Annual Conference of the Society

of Instrument and Control Engineers of Japan (SICE), pp. 615-620, 2019.

[38] M. Schultz, "Field Trial Measurements to Validate a Stochastic Aircraft Boarding Model," Aerospace 2018, Vol. 5(1), No. 27, 2018.

[39] O. Balci, "Quality assessment, verification, and validation of modeling and simulation applications," Proceedings of the 2004 Winter Simulation Conference, pp. 122–129, 2004.

[40] A. M. Law , "How to build valid and credible," 2019 Winter Simulation Conference (WSC), pp. 1402-1414, 2019.

[41] R. G. Sargent, "Verification and validation of simulation models," Journal of Simulation (2013) 7, pp. 12–24, 2013.

[42] M. Raunak and M. Olsen, "Quantifying validation of discrete event simulation models," Proceedings of the Winter Simulation Conference 2014, pp. 628-639, 2014.

[43] B. Hompes, H. M. W. Verbeek and W. M. P. v. d. Aalst, "Finding Suitable Activity Clusters for Decomposed Process Discovery," 4th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA), pp. 32-57, 2014.

[44] W. M. P. v. d. Aalst, H. T. de Beer and B. van Dongen, "Process Mining and Verification of Properties: An Approach Based on Temporal Logic," OTM 2005: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, pp. 130–147, 2005.

[45] A. Pnueli, "The temporal logic of programs," Proceedings of the 18th IEEE symposium on Foundation of Computer science, pp. 46-57, 1977.

[46] H. T. de Beer, "The LTL Checker Plugins: A Reference Manual," Eindhoven University of Technology, 2004.

[47] "CARATS Open Data," [Online]. Available: https://www.mlit.go.jp/koku/koku_tk13_000015.html. [Accessed 27 5 2022].

[48] R. S. MansWil and W. M. P. v. d. Aalst, "Process Mining in Healthcare: Data Challenges When Answering Frequently Posed Questions," ProHealth 2012, KR4HC 2012: Process Support and Knowledge Representation in Health Care, pp. 140-153, 2012.

[49] K. Krinkin, E. Kalishenko and S. P. S. Prakash, "Process Mining Approach for Traffic Analysis in Wireless Mesh Networks," Internet of Things, Smart Spaces, and Next Generation Networking Lecture Notes in Computer Science 7469, pp. 260-269, 2012.

[50] C. Wakup and J. Desel, "Analyzing a TCP/IP-Protocol with Process Mining Techniques," BPM 2014: Business Process Management Workshops, pp. 353-364, 2014.

[51] P. Kecman and R. M. P. Goverde, "Process mining approach for recovery of realized train paths and route conflict identification," Computers in Railways XIII, WIT Press, 2012.

[52] International Civil Aviation Organization (ICAO), "Annex 14 to the Convention on International Civil Aviation – AERODROMES: aerodromes design and operations," 2018.

[53] N. Martin, "Data Quality in Process Mining," Interactive Process Mining in Healthcare, pp. 53-79, 2020.

[54] H. Khadilkar and H. Balakrishnan, "Network Congestion Control of Airport Surface Operations," Journal of Guidance Control and Dynamics 37(3), 2014.

[55] K. Uehara and K. Hiraishi, "An Efficient Mesoscopic Modeling Method for Large Volume Traffic Flow Using Process Mining Techniques," 2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), pp. 1-6, 2021.

[56] P. Burgain, "On the control of aircraft departure operation," PhD thesis, Georgia Institute of Technology, 2010.

[57] S. Kullback and R. Leibler, "On information and sufficiency," Annals of MathematicalStatistics, Vol. 22, No. 1, pp. 79-86, 1951.

[58] K. Uehara, K. Hiraishi and K. Kobayashi, "Mesoscopic Modeling of Airport Surface by Object Petri Nets," 2015 IEEE International Conference on Automation Science and Engineering (CASE), pp. 298-303, 2015.

[59] J.-G. Lee, J. Han and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," Proceedings of the 2007 ACM SIGMOD international conference on Management of data. Association for Computing Machinery, pp. 593–604, 2007.

[60] P. Besse, B. Guillouet, J.-M. Loubes and F. Royer, "Review & Perspective for Distance Based Trajectory Clustering," IEEE Transactions on Intelligent Transportation, 17 (11), pp. 3306-3317, 2016.

[61] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," KDD workshop, Vol. 10, pp. 359-370, 1994.

[62] M. Vlachos, G. Kollios and D. Gun, "Discovering similar multidimensional trajectories," Proceedings of the 18th International Conference on Data Engineering, pp. 673-684, 2002.

[63] L. Chen, M. T. Özsu and V. Oria, "Robust and fast similarity search for moving object trajectories.," Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 491-502, 2005.

[64] A. Dobrkovic, M.-E. Iacob and J. van Hillegersberg, "Maritime pattern extraction and route reconstruction from incomplete AIS data," International Journal of Data Science and Analytics, 5, pp. 111–136, 2018.

[65] D. Filipiak, K. Węcel, M. Stróżyna, M. Michalak and W. Abramowicz, "Extracting Maritime Traffic Networks from AIS Data Using Evolutionary Algorithm," Business & Information Systems Engineering, Vol. 62, pp. 435-450, 2020.

[66] L. Basora, J. Morio and C. Mailhot, "A Trajectory Clustering Framework to Analyse Air Traffic Flows," SID 2017, 7th SESAR Innovation Days, 2017.

[67] X. Olive and M. Jérôme, "Trajectory clustering of air traffic flows around airports," Aerospace Science and Technology, Vol. 84, pp. 776-781, 2019.

[68] M. Ester, H.-P. Kriegel, J. Sander and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pp. 226-231, 1996.

[69] B. R. Bowring, "Total Inverse Solutions for the Geodesic and Great Elliptic," Survey Review, 33, 261, pp. 461-476., 1996.

[70] O. Aichholzer, F. Aurenhammer and D. Alberts, "A novel type of skeleton for polygons," Journal of Universal Computer Science, Vol.1 (12), pp. 752–761, 1995.

[71] J. L. Bentley, "Multidimensional binary search trees used for associative searching,"

Communications of the ACM, Vol. 18, Issue 9, pp. 509–517, 1975.

[72] P. D. Grünwald, J. I. Myung and M. A. Pitt, "Advances in Minimum Description Length: Theory and Applications," The MIT Press, 2005.

[73] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," Computer Graphics and Image Processing, 1(3), pp. 244-256, 1972.

[74] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," The Canadian Cartographer 10(2), pp. 112-122, 1973.

[75] International Civil Aviation Organization (ICAO), "Doc 9931 Continuous Descent Operations (CDO) Manual," 2010.

[76] R. P. J. C. Bose and W. M. P. v. d. Aalst, "Trace Clustering Based on Conserved Patterns: Towards Achieving Better," BPM 2009: Business Process Management Workshops, pp. 170–181, 2009.

[77] Federal Aviation Administration (FAA), "Unmanned Aircraft Systems (UAS) Traffic Management (UTM) Concept of Operations v2.0," 2020.

[78] NASA, "UAM Vision Concept of Operations (ConOps) UAM Maturity Level (UML) 4," 2021.

[79] "Japan Meteoological Agency," [Online]. Available: https://www.data.jma.go.jp/fcd/yoho/typhoon/. [Accessed 27 5 2022].

[80] D. P. Dee, S. M. Uppala, A. J. Simmons and e. al., "The era-interim reanalysis: configuration and performance of the data assimilation system," Q. J. R. Meteorol. Soc. 137, pp. 553–597, 2011.

[81] "NHC Data Archive - National Hurricane Center," [Online]. Available: http://weather.unisys.com/hurricane/atlantic/. [Accessed 27 5 2022].

[82] M. M. Kordmahalleh, M. G. Sefidmazgi, A. Homaifar and S. Liess, "Hurricane Trajectory Prediction Via a Sparse Recurrent Neural Network," Fifth International Workshop on Climate Informatics, 2015.

[83] L. L. Tupper, D. S. Matteson and C. L. Anderson, "Band Depth Clustering for Nonstationary Time Series and Wind Speed Behavior," Technometrics, 60:2, pp. 245-254, 2018.

[84] S. López-Pintado and J. Romo, "On the Concept of Depth for Functional Data," Journal of the American Statistical Association, Vol. 104, No. 486 , pp. 718-734, 2009.

[85] "Multiple Object Tracking Benchmark," [Online]. Available: https://motchallenge.net/. [Accessed 27 5 2022].

[86] J. W. Kamminga, H. C. Bisby, D. V. Le, M. Nirvana and P. J. Havinga, "Generic Online Animal Activity Recognition on Collar Tags," In Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers, pp. 597-606, 2017.

[87] C. Beyan and R. B. Fisher, "Detecting Abnormal Fish Trajectories Using Clustered and Labeled Data," In 2013 IEEE International Conference on Image Processing, pp. 1476–1480, 2013.

[88] "Soccer Video and Player Position Dataset," [Online]. Available: https://datasets.simula.no/alfheim/. [Accessed 27 5 2022].

[89] "McGill Hockey Player Tracking Dataset (MHPTD)," [Online]. Available: https://github.com/grant81/hockeyTrackingDataset. [Accessed 27 5 2022].

[90] "APIDIS Dataset," [Online]. Available: http://sites.uclouvain.be/ispgroup/index.php/. [Accessed 27 5 2022].

[91] "SPIROUDOME Dataset," [Online]. Available: https://sites.uclouvain.be/ispgroup/Softwares/SPIROUDOME. [Accessed 27 5 2022].

[92] M. Pesic, D. Bosnacki and W. M. P. v. d. Aalst, "Enacting Declarative Languages Using LTL: Avoiding Errors and Improving Performance," SPIN'10: Proceedings of the 17th international SPIN conference on Model checking software, pp. 146–161, 2010.

[93] W.-L. Lu, K. Okumura and J. J. Little, "Tracking and recognizing actions of multiple hockey players using the boosted particle filter," Image and Vision Computing 27 (2009), pp. 189–205, 2009.

[94] V. L. Knoop, "Introduction to Traffic Flow Theory: An introduction with exercises,"

TU Delft Open, 2017.

[95] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang and E. Keogh, "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," Proceedings of 34th Very Large Data Bases Endow., Vol. 1, No. 2, pp. 1542–1552, 2008.

[96] G. G. Løvås, "Modeling and simulation of pedestrian traffic flow," Transportation Research Part B: Methodological, Vol. 28(6), pp. 429-443, 1994.

Publications

Journal Paper

[1] Kenji Uehara, Kunihiko Hiraishi, Kokolo Ikeda, “An Efficient Aircraft Boarding Strategy Considering Implementation,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E104.A, 8, 1051-1058, 2021

International conference (refereed)

[2] Kenji Uehara, Kunihiko Hiraishi, Koichi Kobayashi, “Mesoscopic Modeling of Airport Surface by Object Petri Nets,” 2015 IEEE International Conference on Automation Science and Engineering (CASE), pp. 298-303, 2015.

[3] Kenji Uehara, Kunihiko Hiraishi, “Process Mining Approach for the Conformance Checking of Discrete-Event Simulation Model,” 2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), pp. 615-620, 2019.

[4] Kenji Uehara, Kunihiko Hiraishi, “An Efficient Mesoscopic Modeling Method for Large Volume Traffic Flow Using Process Mining Techniques,” IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), pp. 1-6, 2021.

[5] Kenji Uehara, Kunihiko Hiraishi, “A Framework for Extracting Abstracted Route Graphs Toward Air Traffic Flow Modeling,” IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2022. (Accepted)

Domestic conference (non-refereed)

[6] Kenji Uehara, Kunihiko Hiraishi, “Process mining approach for the conformance checking of discrete-event simulation model,” IEICE-MSS, Vol. 118, No. 295, pp. 121-126, 2018.

[7] Kenji Uehara, Kunihiko Hiraishi, “A Method for Mesoscopic Modeling of Large Volume Traffic Flow Applying Process Mining Techniques,” IEICE-MSS, Vol. 121, No. 317, pp. 112-117, 2022.