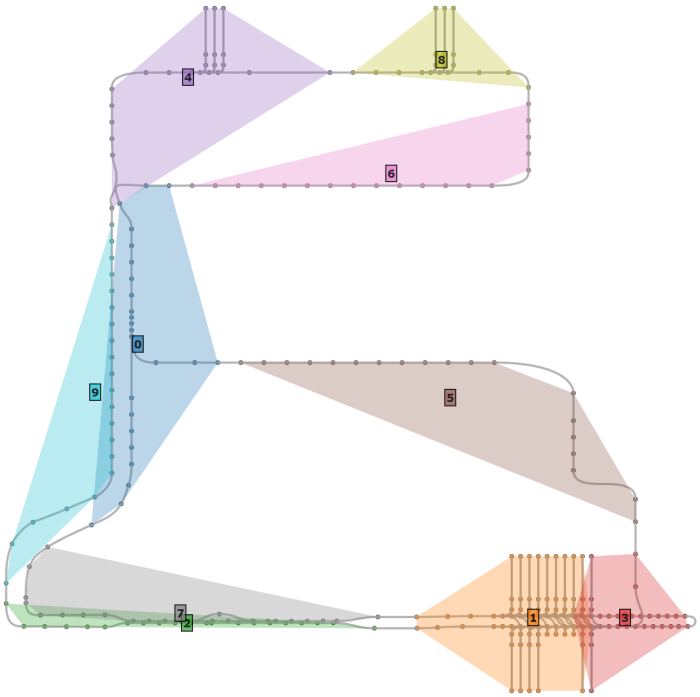


Characterization of Traffic Dynamics in Automated Guided Vehicle Systems

Analyzing and visualizing congestion patterns using causal chains and spectral clustering

Master's thesis in MPCAS & MPDSC

Calle Andersson
Benjamin Veldhuis

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's Thesis in Complex Adaptive Systems & Data Science and AI

Characterization of Traffic Dynamics in Automated Guided Vehicle Systems

Analyzing and visualizing congestion patterns using causal chains and spectral clustering

Calle Andersson
Benjamin Veldhuis

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Automated Systems
Chalmers University of Technology
Gothenburg, Sweden 2025

Characterization of Traffic Dynamics in Automated Guided Vehicle Systems
Calle Andersson
Benjamin Veldhuis

Academic Supervisor: Marco L. Della Vedova, Mechanics and Maritime Sciences
Industrial Supervisor: Rasmus Åkerlund, Kollmorgen
Examiner: Marco L. Della Vedova, Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Cover: Spectral clustering results on a virtual road network for automated guided vehicles.

Typeset in LaTeX
Gothenburg, Sweden 2025

Characterization of Traffic Dynamics in Automated Guided Vehicle Systems
Analyzing and visualizing congestion patterns using causal chains and spectral clustering

Calle Andersson
Benjamin Veldhuis
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Automated Systems
Chalmers University of Technology

Abstract

Automated Guided Vehicle (AGV) systems have revolutionized warehouse logistics by enhancing efficiency and reducing labor costs. However, the complexity of these systems can lead to traffic congestion, adversely affecting performance. This masters thesis investigates traffic dynamics in AGV systems using recursive data structures, interactive visualization, and spectral clustering; with the final aim being to describe and analyze congestion in these systems. Our study shows promise in streamlining design processes by providing more sophisticated analysis tools, data-structures for describing congestion, as well as novel ways to uncover and characterize large-scale traffic dynamics in the domain. Ultimately, this project could act as a stepping-stone for further research, by laying the ground-work for applying more formal methods.

Keywords: automated guided vehicles, traffic analysis, traffic modeling, congestion analysis, congestion modeling, spectral graph theory, spectral clustering.

Preface

This report presents the outcome of our masters thesis project carried out at the Department of Mechanics and Maritime Sciences at Chalmers University of Technology during the autumn of 2024. It was done in collaboration with Kollmorgen Automation AB, located in Mölndal, Sweden.

Acknowledgements

We, the authors, would like to extend our deepest gratitude and appreciation for the opportunity to carry out our master's thesis in collaboration with Kollmorgen. We dedicate special thanks to Rasmus Åkerlund, our industrial supervisor, whose immense efforts, inspiration, and motivation helped us steer the project to completion. We also express our heartfelt thanks to Marco L. Della Vedova, our examiner and academic supervisor, for his invaluable guidance, reassurance, and the freedom he provided us to pave our own path. Finally, we extend our sincere thanks to all our co-workers at Kollmorgen for their relentless support and insightful feedback. The whiteboard sessions, discussions, and lunch talks truly made a significant difference. Especially, we wish to highlight Michael Cederman, senior software developer at Kollmorgen, for sharing his profound insights and extensive knowledge of the AGV domain with us.

Calle Andersson, Benjamin Veldhuis, Gothenburg, February 2025

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Purpose . . . 2
- 1.3 Goals . . . 3
- 1.4 Limitations . . . 3

2 Theory **5**
- 2.1 AGV Systems . . . 5
  - 2.1.1 Layout . . . 5
  - 2.1.2 Automated Guided Vehicles . . . 6
  - 2.1.3 Traffic in AGV Systems . . . 7
  - 2.1.4 Waste in AGV Systems . . . 7
  - 2.1.5 Simulated AGV system . . . 9
- 2.2 Spectral Graph Theory . . . 9
  - 2.2.1 The Laplacian . . . 10
  - 2.2.2 Eigenvalues of the Laplacian . . . 12
  - 2.2.3 Eigenvectors of the Laplacian . . . 12
  - 2.2.4 Extension to Directed Graphs . . . 13
- 2.3 Spectral Clustering . . . 14
  - 2.3.1 Spectral Embedding . . . 15
  - 2.3.2 k-means . . . 15
  - 2.3.3 Eigengap Heuristic . . . 15
  - 2.3.4 Algorithm . . . 16
  - 2.3.5 Why does Spectral Clustering work? . . . 17
- 2.4 Adjacent Domains . . . 17
  - 2.4.1 Complex Systems . . . 18
  - 2.4.2 Railway Traffic . . . 18
  - 2.4.3 City & Highway Traffic . . . 18
- 2.5 Related Work: Delay Propagation Model . . . 19
  - 2.5.1 Model . . . 19
  - 2.5.2 Clustering . . . 20
  - 2.5.3 Clustering Visualization . . . 20
  - 2.5.4 Clustering Metrics . . . 21

3 Dataset **23**
- 3.1 Data Generation . . . 23

3.2 Dataset Components . . . 23
3.3 Our dataset . . . 24

4 Methods **27**
4.1 Domain Specific Definitions . . . 27
4.1.1 Blocking . . . 27
4.1.2 Blocking Reason . . . 27
4.1.3 Blocking Graphs . . . 27
4.2 Micro-level Descriptive Analytics . . . 28
4.2.1 Case Study . . . 28
4.2.2 Parsing and Blocking Graph Creation . . . 29
4.2.3 Visualization Tool . . . 30
4.3 Macro-level Modeling . . . 30
4.3.1 Layout Preprocessing . . . 31
4.3.2 Layout Adjacency Matrix . . . 31
4.3.3 Layout Spectral Clustering . . . 33
4.3.4 Modeling System Behavior . . . 34
4.3.5 Cluster Visualization . . . 40
4.3.6 Cluster Relationships . . . 40
4.3.7 Cluster Characterization . . . 41
4.4 Development language and tools . . . 42

5 Results **45**
5.1 Micro-level Descriptive Analytics . . . 45
5.1.1 Visualization tool . . . 45
5.1.2 Traffic Dynamics of Dataset . . . 48
5.2 Macro-level Modeling . . . 54
5.2.1 Spectral Embedding . . . 54
5.2.2 Base System . . . 55
5.2.3 Separated System . . . 59
5.2.4 Overlapping System . . . 62
5.2.5 Congestion System . . . 67

6 Discussion **71**
6.1 Micro-level Descriptive Analytics . . . 71
6.1.1 Blocking graphs . . . 71
6.1.2 Metrics . . . 71
6.1.3 Visualization Tool . . . 73
6.2 Macro-level Spectral Clustering . . . 74
6.2.1 Cluster Creation . . . 74
6.2.2 Clustering analysis . . . 77

7 Conclusion **79**

Bibliography **81**

1
Introduction

In recent years, the market for automated logistics systems has seen a steady growth [1]. The growth is not only in traditional logistics - like warehouses and factories - there has also been successful adoption by new markets such as hospitals and automatic parking lots [2]. As market reports indicate continued growth for logistics automation and the emergence of automation in new sectors, these systems are poised to have a significant impact. Logistics automation systems include various subdomains, with Automated Guided Vehicle (AGV) systems being a prominent solution. The global AGV market is projected to reach approximately $5.4 billion in 2024, with a compound annual growth rate (CAGR) of 9.8% forecasted from 2024 to 2030 [3]. To remain competitive in this expanding market, logistics automation businesses must strive to design these systems as efficiently as possible. This thesis aims to enhance that design process.

1.1 Background

AGV systems operate on a virtual road network, where vehicle movement is confined to predetermined paths divided into directed segments. We refer to this network as the AGV layout. When creating this layout, the person in charge is called the layout designer. The iterative process the layout designer employs when creating a layout usually consists of three stages, as illustrated in Figure 1.1. The first part is the design stage, where the layout is initially constructed and verified to follow the physical limitations of the site, such as walls, pillars and doors, as well as making sure the design is feasible in regards to the traffic throughput wanted by the customer. For simulation the specified throughput is converted to a list of orders. Each order instructs an AGV to move a load from one station to another. As a second stage, the system is simulated and the results are analyzed. If the results are unsatisfactory the third step is to interpret what is causing this to happen and to redesign the layout based on that. The redesigned layout is then re-simulated. In reality it is often necessary to repeat the simulation-analysis-redesign step many times, and since each iteration takes a long time it is crucial to try and do the most with each simulation.
The problem arises when the layout designer has to interpret why the system failed to pass the requirements. Currently, the process of troubleshooting the layout commonly includes observing the traffic flow in the simulation as its running - which is highly time-consuming - or finding problematic areas using bar-charts on Key Performance Indicators (KPIs). But even then there is no formula to find the root cause

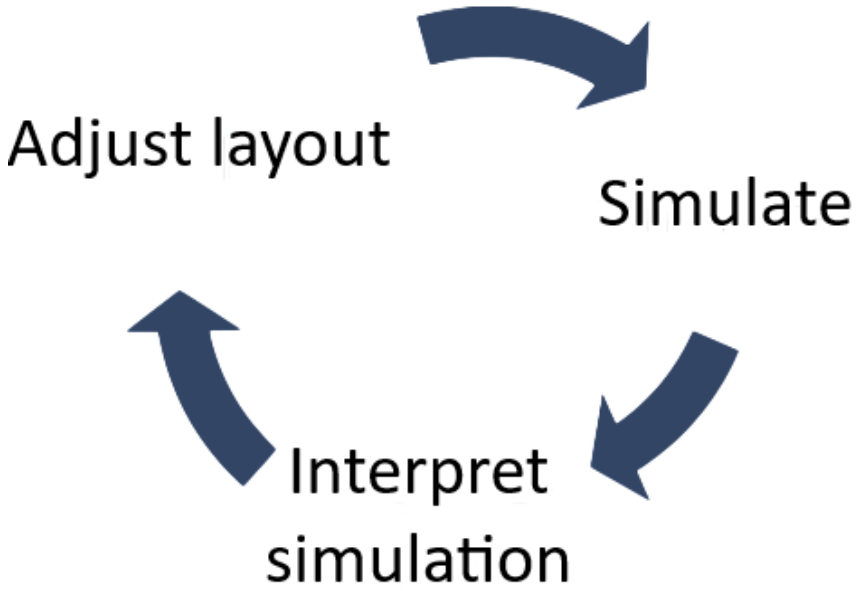


Figure 1.1: After creating a layout, it is simulated to assess performance. If the results are subpar, the designer analyzes the simulation, adjusts the layout, and repeats the process. This cycle—design, simulate, and refine—is known as the iterative three-stage process of layout design.

of the issue, instead, engineers rely on years of hard-earned intuition and experience. This leads to two undesirable outcomes. Firstly, when the methods used come from experience, there's an immense gap to cross for junior layout designers before being effective. There is also no intuitive way to explore the simulation output, making the experience unnecessarily hard to obtain. Secondly, the fact that even domain experts draw different conclusions when troubleshooting the same layout can lead to confusion and unforced errors in judgment. This points to there being a need for a more standardized and intuitive way to understand and draw insights from the simulation data.

1.2 Purpose

The purpose of this project is therefore to improve the processes used to explore the simulation data. To address the need for more sophisticated tooling and insights, the main idea is to try and explain the AGV system traffic dynamics on both a micro- and macroscopic level. Reflecting these two different approaches, there will be two main purposes.

1. **Micro-level Descriptive Analytics**, understand what is actually happening in an AGV system in terms of individual vehicle traffic dynamics in limited areas.
2. **Macro-level Modeling**, uncover and characterize key areas in an AGV layout by analyzing large-scale traffic dynamics.

The first part serves the purpose of being more practically grounded, focusing on

gaining intuition about the system and providing tools and data-structures that can interactively visualize what has in fact happened and why. On the other hand, the second part is a more theoretical approach, and can be further sub-divided into three smaller research questions.

- How to **uncover** areas of interest in a layout?
- How do **relate** areas to each other?
- How to **characterize** which area is the most interesting?

1.3 Goals

As mentioned, it is important for the project to assist layout designers, both junior and senior, in understanding their data better, in turn helping bridge the gap needed to make data-driven decisions when designing a layout. This is one of the main goals of the project, and how this goal fits into the larger scope is summarized in Figure 1.2. Another goal is to pioneer and apply more experimental methods, and in this breaking of new ground, the hope is that the project will build a strong theoretical base for describing AGV system traffic dynamics, stimulating further research.

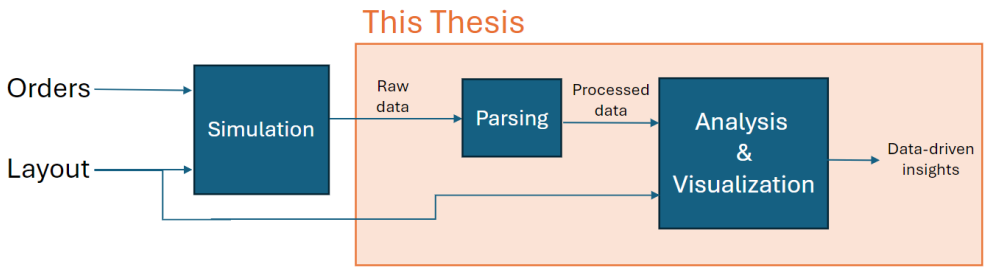


Figure 1.2: Diagram of how our tool fits in the layout design pipeline. In addition to the layout the simulation receives a set of orders describing when and where the AGVs need to be dispatched. We strive to provide data driven insights by parsing, analyzing and visualizing the simulation data.

1.4 Limitations

The project is in the field of descriptive analytics and modeling, which means that the main goal is to explain and describe the data. There is no form of predictive modeling, which is otherwise common in the field of traffic flow analysis. Additionally, the time aspect of the data is largely disregarded, since the focus is on aggregated data. This makes it easier to visualize and discover general trends, although, with that said, aggregation also means that there will be a certain loss of information. Analyzing these temporal aspects are not in the scope of this project.

2

Theory

This chapter provides the knowledge base required in AGV systems and provides a mathematical description of the system. Additionally, the reader will get introduced to spectral graph theory and one of its key applications, spectral clustering. Finally, there will be a review of adjacent domains and related works.

2.1 AGV Systems

Automated Guided Vehicles (AGVs) are robotic systems used for material handling and transportation in various industries. The first AGV was introduced more than 70 years ago, revolutionizing material handling in manufacturing and warehousing. Since then, the development of AGVs has exploded, with advancements in technology making them an essential tool for automating processes in many fields [2].

2.1.1 Layout

The cornerstone of any AGV system is its virtual road network, commonly referred to as the *layout*. This layout consists of points connected by one or more directed segments, defining how vehicles can navigate. The connectivity and directionality impose strict movement constraints, further reinforced by a key characteristic of AGV systems: only one AGV can occupy a segment at any given time. Apart from this, there are even more properties that dictate how vehicles can interact with the layout. Certain points are designated with special roles, such as charging ports, loading/drop-off bays, or home areas, signifying locations where specific actions can occur. These points are collectively referred to as *stations*. See an example of a layout, with stations and segments in Figure 2.1. Segments, on the other hand, are represented as splines, defined by a start- and end-point, and intermediate control points that determine the shape. Additionally, since there can be multiple vehicle types in the same layout, and different vehicle types require different topological considerations, each segment can have one or multiple allowed vehicle types. Finally, segments have an expected drive time, also called weight. This is both dependent on the length of the segment and its shape, since both these factors change how quickly a vehicle can traverse the segment.

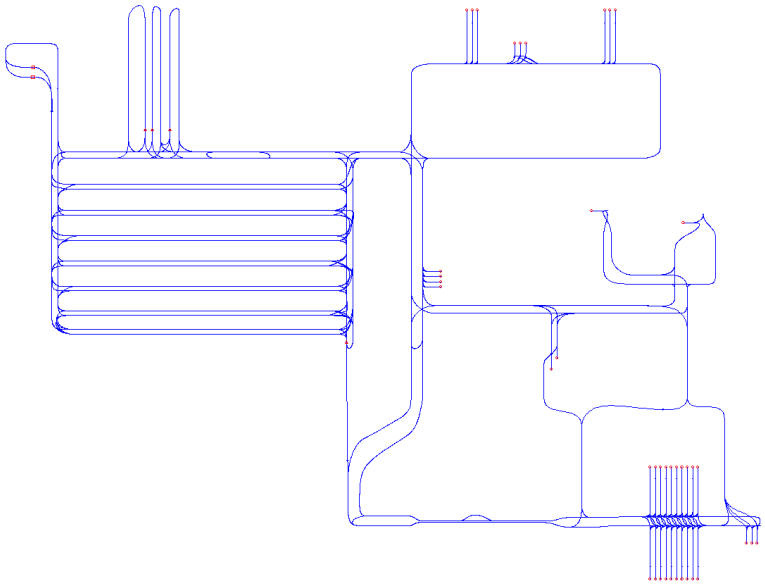


Figure 2.1: An AGV road network as displayed in Kollmorgens in-house software. The dark-blue lines are regular drivable segments and the red points are stations corresponding to load- and drop-off bays.

To describe the layout using classical mathematical notation, it is natural to turn to graph theory [4]. The layout, as described above, consists of points that are connected by one or multiple directed segments. Interpreting these points as nodes and the segments as directed edges provides a complete representation of the layout's connectivity. Incorporating properties such as positions and shapes of nodes and edges further defines the layout. The *segment weight*, corresponding to the *expected travel time*, could be directly encoded as edge weights. Although in graph theory, for simplicity, it is often assumed that the graph is unweighted, and handling weighted graphs is seen as an optional extension that may or may not be possible given the mathematics at hand. This thesis will follow a similar approach, and from here on, if nothing else is said the reader should assume that the layout refers to an *unweighted graph*. This also means that we will defer from calling the expected travel time as weight and rather just keep it as expected traversal time. If not for the described reason, it will become more clear later in the report. Finally, remember that there might be multiple segments connecting the same two points in an AGV system. Thus, the layout can be represented as a *directed multigraph*, where multigraph denotes a graph which can potentially have several edges between each pair of nodes [4].

2.1.2 Automated Guided Vehicles

Another core part of an AGV system are the *vehicles* that traverse the layout. They are the ones that physically transport goods from one area to another, and, in the domain, performing this successfully may also be called to carry out an order or finishing a mission. Since the vehicles aren't truly autonomous, AGVs are

automated by a querying a centralized processing unit –the fleet manager– and then actions are taken based on the replies. The fleet manager has a bird's eye view of the whole system; keeping track of all vehicle locations, distributing missions and restricting AGV movement using a certain set of rules. Sometimes, upholding these rules can result in one or more vehicles being forced to wait, which is commonly called a *blocking*. As we will see in the coming sections, blockings are necessary to control traffic, but at the same time excessive blockings represent a form of *waste*. Therefore, one of the main challenges for the layout designer is to identify which blockings are wasteful and then minimize their occurrence.

2.1.3 Traffic in AGV Systems

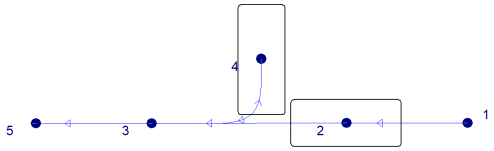
As stated above, the fleet manager enforces a certain set of rules that restrict AGV movements. These are commonly referred to as *traffic rules*. The main purpose of the rules is to avoid unsolvable traffic conflicts and vehicle collisions (see Figure 2.2 for a concrete example). Rules of this type are crucial for a system's feasibility and therefore are largely generated automatically during the layout design process. Other rules, which focus more on improving layout performance, are not created automatically. These rules are instead manually designed by the layout designer to manipulate traffic in advantageous ways. Common examples include forcing vehicles to wait outside a narrow aisle, avoiding a traffic jam by not driving too close to a station when another vehicle is loading or unloading. In practice, there can be many of these extra rules, leading to highly complex traffic dynamics.
Generally, each traffic rule can be parameterized by two sets,

1. a set of *boolean conditions*,
2. a set of *segments*.

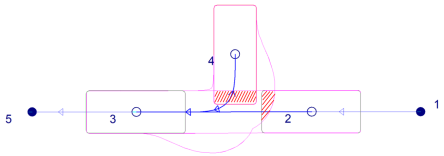
When all of the *boolean conditions* are true, it is said that the traffic rule has been activated, consequently blocking the *segments*. So therefore, a traffic rule can be seen as a function and a traffic rule activation, or blocking, can be seen as an instance of that function being active. Note that similarly to how a red traffic-light is not guaranteed to have waiting cars, a traffic rule activation does not necessarily imply that any AGVs are blocked from moving. This definition allows for more flexibility than how the traffic rules typically tend to be defined, however, by having a definition that is as general as possible we make sure our work is more widely applicable. As a bonus it also simplifies the notation. In the AGV domain, the term blockings is sometimes used interchangeably for both the traffic rules themselves and the instances of them being activated. For clarity, we will maintain the separation between the two concepts, and use the term *blockings* only for the latter.

2.1.4 Waste in AGV Systems

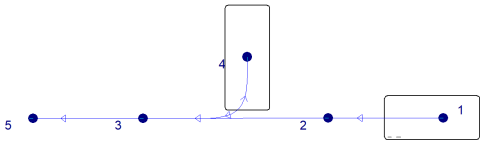
At first glance it might seem like blockings are to be avoided in general. But, as hinted at earlier, some blockings are actually planned by the layout designer to improve layout performance. It should be noted that these tactically imposed blockings are a subjective decision that is made to try and solve some specific problem in the layout, and subjective decision-making opens up the risk for unforeseen



(a) Inescapable deadlock



(b) Sweep of vehicles



(c) Traffic-rule consequence

Figure 2.2: This figure shows why traffic rules are a necessity in AGV systems. In (a), a situation called an inescapable deadlock is shown. This is a traffic situation that the system cant solve by itself, since whichever vehicle moves, it will cause a collision. This is due to the sweep shown in (b). To avoid this scenario and many others, traffic rules enforce vehicles to be blocked, see (c) for the consequence of the traffic rule.

consequences. As a design grows more complex, it becomes harder and harder to determine which of all the blockings are actually *useful* (for streamlining a layout). If a blocking is not useful, we say that it is *wasteful*, but how to define this in more detail?

To be able to known which blockings are the most important to address in terms of waste, one has to be able to *rank* them. A blocking being wasteful or not depends on an extensive list of conditionals, such as if it carries a load, has a mission dedicated to a critical order flow, causes other blockings that hinders other order flows, etc. This means that the relationship between blockings and waste is multidimensional, and even system and situation specific. In turn, this makes it hard to define a general measure of waste. This is left as an interesting research topic for the future.

Although most of the waste (in a typical AGV system) comes from these wasteful blockings, there are also other sources of *wasted time*. Some examples include when an AGV:

- drives a longer path than the shortest one,
- needs to charge too often due to having an old battery,
- drives without having a load,
- stands still due to not having a mission dedicated to it.

All of these (and more) lead to some sort of inefficiency in the system, either in terms of optimizing money spent, traffic-flow or carrying out the required number of orders per hour. Taking this into account, in combination with the complexities mentioned for wasteful blockings, makes it quite difficult to determine a single metric of how

much inefficiency, or *waste*, there is in a given system. In this project we therefore limit ourselves to blockings, and we do not distinguish between a blocking and a wasteful blocking.

2.1.5 Simulated AGV system

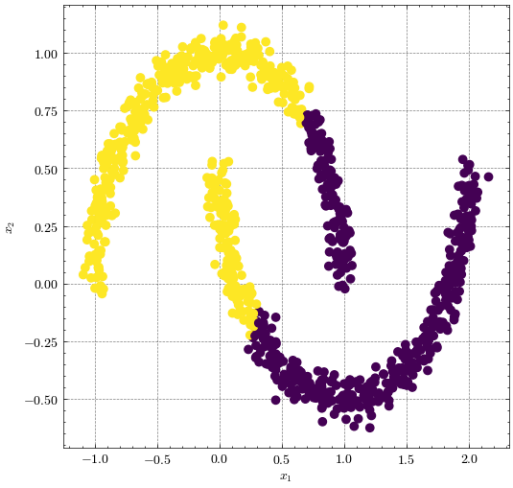
To be able to verify that a layout is working as expected, one can simulate the AGV system instead of having vehicles drive on the physical layout. Basically, this works by creating pseudo AGVs that mimic real ones, in the sense that they communicate the same type of requests to the fleet manager. Then the pseudo AGVs will act based on the replies, emulating the behavior of a physical AGV system.

The simulated system has obvious advantages; for one it's faster, both in the sense that there are less preparations required before testing the layout, and in the sense that the simulation literally can run faster than real life. There is also less risk involved, as mistakes in the physical system could break equipment or AGVs. There are not many disadvantages with the simulation, apart from it yielding slightly different outputs than real life. This, in turn, means that it is uncertain that the system works as intended until it has been verified on the physical layout. The difference between the simulated and physical system also grows bigger as the simulation is sped up, which leads to even more uncertain results, meaning that it is not always easy to determine the balance of when to go from testing a system in simulation to testing it in real-life. The hope is that making the simulation data easier to analyze will aid in taking this decision.

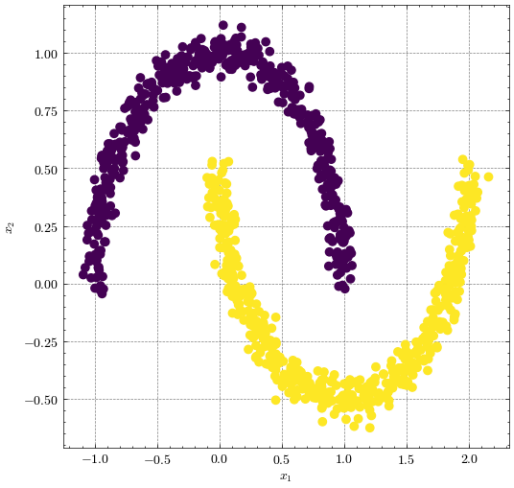
2.2 Spectral Graph Theory

Spectral graph theory is a field that is about analyzing the spectral properties of graphs. The analysis of the eigenspectra of matrices corresponding to a graph, such as the adjacency- or Laplacian matrix, is precisely what gives spectral graph theory its name [5]. Although the name suggests the analysis of graphs, one can also apply it for more general problems, as long as it is possible to convert the problem to a graph. A classical example is spectral clustering, wherein the problem of finding communities in a graph is solved by applying spectral graph theory and clustering in tandem [6]. But the problem of finding communities in a graph is also synonymous to finding a relevant partition of some arbitrary data, given a measure of how similar the data points are to each other. With the correct similarity metrics, this method can even find communities in data with non-linear boundaries, such as seen in Figure 2.3. We will come back to spectral clustering in the following sections. Other applications range from ranking data [7], graph cutting, graph drawing [8], and image segmentation [9].

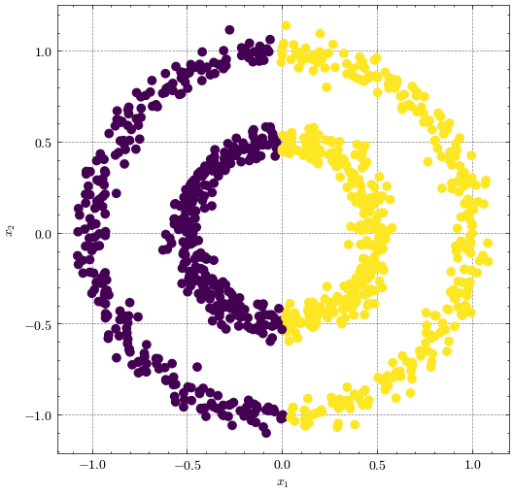
Traditionally, spectral graph theory analyzes simple graphs, that is, unweighted, undirected graphs containing no loops or multiple edges [5]. Although there are also extensions for directed graphs, which will be presented later in Section 2.2.4.



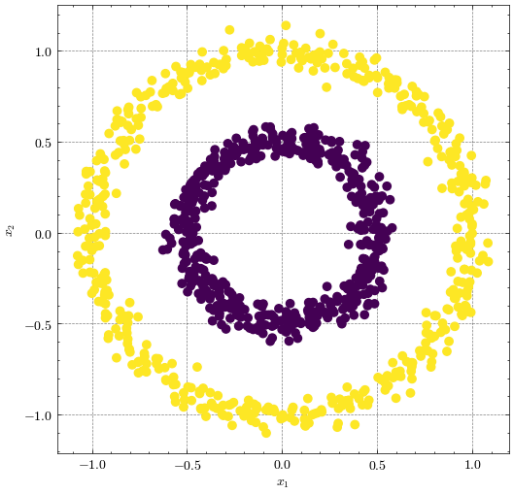
(a) k-means clustering on moons dataset



(b) Spectral clustering on moons dataset



(c) k-means clustering on circles dataset



(d) Spectral clustering on circles dataset

Figure 2.3: Example showing the advantage of spectral methods. As you can see it enables clustering even when different communities have non-linear separation boundaries. This is a stark advantage compared to just applying k-means directly.

2.2.1 The Laplacian

A cornerstone of spectral graph theory is the Laplacian matrix. In the literature it has many different definitions, and in its simplest form it can be seen as a matrix representation of a graph, embedding its connectivity, node-degree and edge-weights. For undirected graphs, but also in general, the Laplacian has useful mathematical properties that make it convenient to work with [10]. This is the reason why it is the preferred matrix representation in spectral graph theory, instead of the adjacency matrix, for example.

In general, there are Laplacian matrices defined for a multitude of graph types, and

the most basic one is the *Laplacian*, $\mathcal{L}$, which will be first defined for unweighted graphs. In Hamilton [10] this is denoted as the *unnormalized Laplacian*. For a graph G with n nodes, the Laplacian $\mathcal{L}$ is a $n \times n$ matrix with each entry $\mathcal{L}_{ij}$ defined as follows, where d_j denotes the degree of node j:

$$\mathcal{L}_{ij} = \begin{cases} d_j & \text{if } i = j, \\ -1 & \text{if } i \text{ and } j \text{ are adjacent}, \\ 0 & \text{otherwise.} \end{cases}$$

For a more shorthand notation, define a diagonal matrix D as *the degree matrix*, with entry D_{jj} corresponding to d_j, and A as *the adjacency matrix*. Now $\mathcal{L}$ can equivalently be written as:

$$\mathcal{L} = D - A. \tag{2.1}$$

Additionally, since there are no self-connections, diagonal elements only depend on the degree matrix D. This means that if one node has a high degree compared to other nodes, it will have a larger diagonal entry, leading to it dominating the matrix properties. In the literature, the proposed solution for this is to normalize $\mathcal{L}$, which can be done in numerous ways. An option is *symmetric normalized Laplacian*, $\mathcal{L}^{sym}$, with the following definition [10]:

$$\mathcal{L}^{sym}_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } d_j \neq 0, \\ -\frac{1}{\sqrt{d_i d_j}} & \text{if } i \text{ and } j \text{ are adjacent}, \\ 0 & \text{otherwise.} \end{cases}$$

The meaning of this is that all rows and columns are normalized symmetrically, which in other words mean that all rows and columns are normalized as to sum to zero. In other methods the goal can be to normalize $\mathcal{L}$ per row or per column. Most notably there is the *left-normalized Laplacian* (also called the random-walk Laplacian), corresponding to normalizing per row (so that each row sums to zero). This has applications in modeling random-walks in graphs, as it turns out to be closely related to a transition matrix or right-stochastic matrix [11]. In Section 2.2.4 this will be more clearly explained.
To extend this to weighted graphs, one can simply use the *weighted adjacency matrix*, W, and *weighted degree matrix*, D, which still is a diagonal matrix, but with entry D_{ii} corresponding to the weighted degree of node i:

$$d_i = \sum_{u \neq i}^{n} W_{iu}.$$

Hence, the definition of $\mathcal{L}^{sym}$ for weighted graphs is:

$$\mathcal{L}^{sym}_{ij} = \begin{cases} 1 - \frac{W_{ij}}{d_i} & \text{if } i = j \text{ and } d_i \neq 0, \\ -\frac{W_{ij}}{\sqrt{d_i d_j}} & \text{if } i \text{ and } j \text{ are adjacent}, \\ 0 & \text{otherwise.} \end{cases}$$

The shorthand notation for $\mathcal{L}^{sym}$, both in the unweighted and weighted case, can be defined similarly as (2.1):

$$\mathcal{L}^{sym} = D^{-1/2}\mathcal{L}D^{-1/2}. \tag{2.2}$$

Note that in linear algebra, the operator $D^{-1/2}$ refers to the matrix that, when applied, outputs the element-wise square root of each element of the inverse of matrix D. From now on, assume that $\mathcal{L}$ denotes the symmetric normalized Laplacian, $\mathcal{L}^{sym}$.

2.2.2 Eigenvalues of the Laplacian

The ordering as well as the magnitude of the eigenvalues, known as the spectrum of $\mathcal{L}$, encodes a lot of information about the structure and the properties of the graph. Firstly, by construction $\mathcal{L}$ is guaranteed to be positive semi-definite [5], so containing only real and non-negative eigenvalues [12]. Secondly, since $\mathcal{L}$ is singular (all rows sum to zero [12]), it can be proven that there must always exist at least one eigenvector, the one-vector, corresponding to the eigenvalue zero [5]. From these two statements it follows that the smallest eigenvalue is always zero. This is summarized in the following equation, where n is the number of nodes in G and λ_i denotes the ith eigenvalue in the spectrum of $\mathcal{L}$:

$$0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}.$$

Additionally, it can be shown that the number of zero eigenvalues is equal to the number of connected components in the graph [5]. This means that for a connected graph, where each node can be reached from every other node, the following always holds true:

$$\begin{aligned} &\lambda_0 = 0, \\ &\lambda_i > 0 \qquad \forall i \quad 0 < i < n. \end{aligned}$$

The second smallest eigenvalue, λ_1, is known as the *algebraic connectivity* [13]. It has many proven properties that are highly important in the field of spectral graph theory [5]. For example, a higher magnitude can be related to a graph being more resistant to the removal of nodes or edges, i.e., having a stronger connectivity. If instead it has a low magnitude, it indicates that the graph is more vulnerable to fragmentation. Most importantly, λ_1 provides a way to partition the graph according to a relaxation of the *minimal cut* problem [4].

2.2.3 Eigenvectors of the Laplacian

A graph's spectrum reveals much more than just its eigenvalues. By ordering the eigenvectors according to the spectrum, we can uncover hidden geometric patterns and topological features within the graph's structure. This is proven to be a powerful tool for understanding deeper properties of graphs.
As mentioned above, if the underlying graph is connected then the eigenvector corresponding to the smallest eigenvalue is simply the one-vector. This is the trivial solution to the eigenproblem, and is analogous to the case where all nodes in the

graph are equally weighted. Although this eigenvector is often discarded in real-life applications, since it does not contain any meaningful information.
Similar to the second eigenvalue, the second eigenvector also has some special properties. In honor of the Czech mathematician Miroslav Fiedler, it is commonly referred to as the *Fiedler vector*. Its interest comes from the fact that the coefficients provide a natural binary partition of the graph - one set of nodes that correspond to the negative coefficients, and one to the positive. This partition is proven to be a solution to a relaxed version of the minimal cut problem, which means to split the graph such that the least amount of edges are on the border between the two sides. For more details on this, see [6].
Further eigenvectors, corresponding to eigenvalues larger than λ_1, can distinguish more complex structures in the graph. Due to the fact that $\mathcal{L}$ is symmetric (since it is positive semi definite), all its eigenvectors are orthogonal [14], in turn leading to them containing different information *by definition*. However, larger eigenvalues correspond to eigenvectors with less useful information, since the smaller the eigenvalues the more sparse the cut [5, 6]. Thus, utilizing the eigenvectors associated with the smallest eigenvalues offers a powerful mechanism to reduce the dimensionality of the problem when searching for patterns in a graph. This is precisely the motivation behind spectral clustering which will be further detailed in Section 2.3. Still, it is a very hard problem in general to decide when using further eigenvectors yields diminishing returns, and in spectral clustering this is usually referred to as picking the first $k-1$ eigenvectors (remember, the first one is discarded). Some other methods are presented in [6], especially the so-called *eigengap heuristic* [15], which will be revisited later.

2.2.4 Extension to Directed Graphs

Even though the focus of this work will be to apply the theory for undirected graphs, it should be mentioned that it is possible to perform it on directed graphs as well. This is proven to be useful in some applications where directionality matters, such as in ranking web-pages [7], and also showed promising results when finding communities in directed graphs [16].
Spectral graph theory has been extended to directed graphs [11]. The key idea behind this extension is the concept of *random walks* [17]. A walk on a graph is simply a valid traversal of a set of nodes in the graph, and a random walk is a walk where, at any given node i, the choice of what neighbor node j comes next is randomly selected according to some probability P_{ij}. For unweighted graphs this probability is typically uniform, whereas for weighted graphs a weighted out-degree is used instead. The *transition probability matrix*, P, which contains the probabilities P_{ij}, can be defined as follows:

$$P_{ij} = \begin{cases} \frac{W_{ij}}{\sum_z W_{iz}} & \text{if (i,j) is an edge,} \\ 0 & \text{otherwise} \end{cases}$$

where W is the weighted adjacency matrix. Note that W in general can be non-symmetric for directed graphs, meaning that P is also non-symmetric in general.

Before continuing it is important to state intuition of how random walks are connected with spectral graph theory. Going back to the idea of a minimal cut, dividing the graph into these two partitions is actually similar to finding the two areas in which a random walk is most likely to be contained. This also makes sense intuitively, as noted in [6]: "a balanced partition with a low cut will also have the property that the random walk does not have many opportunities to jump between clusters". This is also connected to *spectral clustering*, which will later be mentioned in detail for undirected graphs, since the focus of this paper is not related to directed graphs.

If the underlying graph G is strongly connected, it can be shown that the transition probability matrix P has a unique (left) eigenvector ϕ with all coefficients being strictly positive [11]. This vector is called the *Perron vector* (of P), and corresponds to the simple case when the eigenvalue is equal to one. More importantly, if G also is aperiodic, it can be shown that ϕ is the *stationary distribution*, in other words the convergence of a random walk on G. This, and many other properties of the matrix P, is described in detail by Lovász in [17].

Skipping some important details, we can now conclude this section by defining the Laplacian for directed graphs [11] based on the transition probability matrix P and the Perron vector ϕ. For a simpler notation, use the matrix Φ, which is a diagonal matrix with its diagonal corresponding to coefficients of ϕ. This gives us:

$$\mathcal{L} = I - \frac{\Phi^{1/2} P \Phi^{-1/2} + \Phi^{-1/2} P^* \Phi^{1/2}}{2} \tag{2.3}$$

Here P^* denotes the conjugated transpose of P and from (2.3) it also follows that $\mathcal{L} = \mathcal{L}^*$ [11], implying that $\mathcal{L}$ is a *hermitian* matrix and therefore its eigenvalues are real and its eigenvectors are orthogonal [14]. Additionally, its rows trivially sum to zero (due to the symmetrical normalization). Practically, one can now apply spectral graph theory on $\mathcal{L}$ to generate insights of the directed graph G, since this definition by Chung makes sure the spectrum is well-defined [11].

Before finishing this section, it should be mentioned that random walks is an extensive research topic which is not fully addressed in this project. Still, we see potential in future work done in this area, and we refer interested readers to the relevant chapters in Hamilton [10].

2.3 Spectral Clustering

One application of spectral graph theory is the field of spectral clustering. The idea is to group nodes of a graph into k clusters using the spectral embedding of the graph. The spectral properties of the graph lead to efficient ways of performing dimension reduction that lets a clustering algorithm group similar nodes together [6]. It is also worth noting that spectral clustering is synonymous to a more general notion in data science, which is to perform clustering in a reduced dimension or lower feature space.

2.3.1 Spectral Embedding

By using the Laplacian and its eigenspectrum, spectral properties of the underlying graph can be analyzed. Each eigenvalue has a corresponding eigenvector, which coefficients assign a feature value to each node. For each connected component of the graph there will exist a zero eigenvalue. Note that, if the graph has multiple connected components, it implies that these form a natural partition of the graph already [10]. These eigenvectors will not be used for dimension reduction. If we were to consider all eigenvectors for the clustering the information will be diluted. We want the chosen vectors to be impactful so the dimension of the spectral embedding is optimal for clustering performance. The first $k-1$ non-zero eigenvalues will be chosen and their respective eigenvectors will be extracted into a matrix U. This means that the dimension used for clustering is $k-1$. Each row l of U corresponds to an embedded feature vector for node l. In this embedding clustering will be applied. It is structured as such:

$$U_{n\times(k-1)} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1(k-1)} \\ u_{21} & u_{22} & \cdots & u_{2(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{n(k-1)} \end{pmatrix}$$

2.3.2 k-means

Although other clustering algorithms can be used, k-means is commonly employed in spectral clustering literature. This preference is due to its simplicity, ease of implementation, and reliable performance on spectral embeddings [18]. k-means is an unsupervised algorithm that classifies n points into k clusters [19]. The clustering process is based on a feature vector for each point, with a cluster centroid initialized in the feature space for each k. Each iteration of the algorithm assigns each point to its nearest centroid and updates the centroid position to minimize intra-cluster variance. This is done by moving the centroid to the mean position of all points currently assigned to its class. The algorithm usually iterates until each centroid converged to a position and then the final classification for each point is determined by which centroid they are closest to in the feature space.

2.3.3 Eigengap Heuristic

A problem that needs to be addressed with all clustering algorithms is finding a good way of picking this k. A often used method to determine k in spectral clustering is the eigengap heuristic [15]. The basis of this method is that eigenvectors with low eigenvalues have smoother ways of splitting the graph. By identifying which eigenvalues can be determined as low enough and which contain irrelevant information a appropriate k can be chosen. In essence it can be expressed as:

$$k = \arg\max_i(\lambda_{i+1} - \lambda_i)$$

The eigengap heuristic helps choosing a valid k by exploiting these insightful properties embedded within the eigenvalues. The large gap between values means that the

eigenvalues above the chosen k are determined to contain less significant information. This also uncovers a more understated advantage of spectral clustering, which is that the common challenge of selecting the number of clusters k while performing clustering can be done in a quite natural way using the eigengap heuristic. Usually a k_{max} is used however, meaning that the lowest eigenvalues up to k_{max} are considered while performing the heuristic. An example of the eigengap heuristic works is shown in figure 2.4.

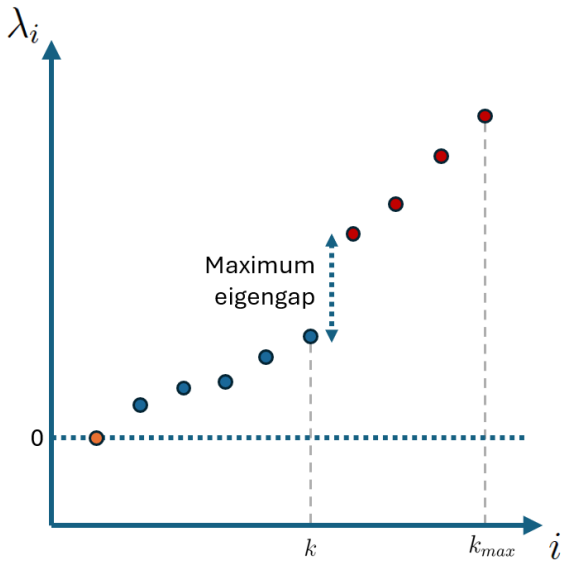


Figure 2.4: An example of the eigengap heuristic applied to an eigenspectrum. In this example $k_{max} = 10$ and the plot indicates $k = 6$ as a suitable choice for clustering.

2.3.4 Algorithm

Combining the spectral embedding, k-means and the eigengap heuristic renders a relatively straight forward algorithm to find clusters in a graph, see Algorithm 1.

Algorithm 1: General Spectral Clustering Algorithm

Input: Weighted adjacency matrix W
Output: Clustered nodes

Step 1: Create a weighted adjacency matrix W;
Step 2: Calculate the Laplacian $\mathcal{L}$ of the graph using W;
Step 3: Using the eigengap heuristic of $\mathcal{L}$ compute a suitable k;
Step 4: Compute the first k eigenvectors of $\mathcal{L}$;
Step 5: Form a matrix U using these eigenvectors, where row l of U corresponds to the features of node l;
Step 6: Apply k-means clustering on the rows of U to cluster the nodes;

2.3.5 Why does Spectral Clustering work?

Spectral Clustering partitions graphs by looking at the structural connectivity. Without prior knowledge, the inner workings of why spectral clustering works can be difficult to understand. To gain a deeper understanding some intuition is needed. Lower magnitude eigenvalues correspond with eigenvectors that split the graph better. This is the reason why the Fiedler vector represents the best way to split a graph in two. It is defined as the lowest value and is therefor highlighting how to value each node to make sure the connectivity is as low as possible between those communities. The eigenvalue of the Fiedler vector is sometimes also called the algebraic connectivity. Extending these to the next eigenvalues the same principle applies. Low eigenvalues indicate an effective method for further splitting the graph.
The spectral properties of the Laplacian ensure that the eigenvectors corresponding to the lowest eigenvalues capture the most significant and smoothest modes of variation. This means they are less affected by noise and provide meaningful representations of variance within the graph.
The orthogonality of additional eigenvectors ensures that they represent different aspects of the graph's connectivity. By including multiple eigenvectors in the spectral embedding, you create a feature space where each feature vector captures a distinct and impactful way of dividing the graph. This approach enables the k-means algorithm to effectively cluster the nodes based on the spectral embedding of the graph. This is precisely what allows the k-means algorithm to properly cluster the nodes when applied to the spectral embedding of the graph.
However, the benefit of including more eigenvectors has diminishing returns as each next eigenvector represents a slightly less efficient way of dividing the graph (as it corresponds to a larger eigenvalue). By looking at $k-1$ eigenvectors the dimensions are reduced to a suitable amount as to make k-means as effective as possible [6]. This is why the eigengap heuristic is used to pick a suitable k. A large jump in the eigenspectrum means that the next eigenvector represents a relatively worse graph split. By only looking at the lowest eigenvalues and ignoring the higher ones we ensure that each eigenvector considered contains relevant feature data for the clustering.
Another benefit of spectral clustering is that it also balances cluster sizes well. When minimizing the cut between the different clusters it is quite easy to imagine a solution that just isolates a single vertex from the rest, thus minimizing the total cut. This, however, is an unsatisfactory result as the clusters should contain communities of nodes with similar properties - not just single individual vertices. For an in-depth explanation of how spectral clustering produces well balanced please see [6].

2.4 Adjacent Domains

The AGV domain, like many others, involves entities moving and interacting with each other and their environment. By examining the core components of the AGV system, one can find numerous similarities with fields such as *complex systems* and *complex networks*. Additionally, *traffic analysis* is a vast area of study that aligns closely with the objectives of this paper. The following section provides a brief

overview of these related fields, highlighting their similarities and differences with the AGV domain. Beyond offering insights and guidance on how to approach the problem at hand, this section also aims to shed light on some of the many research directions still unexplored by our paper, potentially inspiring future research in this topic.

2.4.1 Complex Systems

In [20], Volpe *et al.* defines a complex system: "A complex system can be defined as a system composed of many interacting parts that features emergent behavior. Here, the keywords are many, interacting, and emergent". Looking at this through the perspective of AGV systems, there are *many* vehicles that *interact* with each other and the layout, causing *emergent* behaviors in the form of causal chains of events, for example that certain traffic flows and blockings form spatiotemporal patterns. Understanding complex systems is also inherently difficult [21]. Still, complex systems is a big research field that one could take a lot of inspiration from - despite its research challenges [22]. Even though we will not mention this parallelism any further, we mention it here to potentially inspire future work to explore this matter further.

2.4.2 Railway Traffic

A distinct property of railway systems is that they are divided into a series of sections, each only allowing one train at a time. This is very similar to how traffic works in the AGV domain, and since traffic analysis is more explored in the railway domain (see for example [23, 24, 25]), this could be a good basis for inspiration in future work.
While trains operate on strict layouts following rails, AGV systems utilize virtual road networks, presenting a significant difference. A railway network is a physical road network, meaning that once constructed, it is very difficult to alter. Consequently, decisions made during the construction of railways must be highly calculated. In contrast, AGV systems allow for multiple cycles of design, offering greater flexibility.
AGV systems are inherently more adaptable and subject to change. The cost associated with these systems is not in rebuilding the layout, as with railways, but in ensuring the quality of traffic flow in each new revision.

2.4.3 City & Highway Traffic

A road network, much like an AGV system in a warehouse, can be viewed as a graph where streets represent edges and intersections represent nodes. Just as AGVs move pallets, cars act as agents transporting people between locations. Given that the field of road traffic analysis is more extensively explored, it makes sense to analyze AGV systems with inspiration from city and highway traffic methodologies. This cross-domain inspiration could provide valuable insights and innovative approaches to understanding and managing traffic dynamics in AGV systems.

However, a key difference lies in the continuous operation of cars and other vehicles, as opposed to the discrete, segment-based behavior of AGVs or rail systems. Consequently, road traffic analysis often focuses on aggregation and stochastic models, rather than interactions between individual vehicles. For instance, research might employ models based on spatial abstraction [26] and Markov models [27], as well as studies related to the fundamental diagram of traffic flow [28]. Additionally, research in road traffic tends to be more predictive, aiming to forecast future conditions, rather than merely describing current states. An example of this is the exploration of congestion in relation to empirical driver demand [29] or this paper which tries to optimize traffic using Markov chain traffic assignment [30].

2.5 Related Work: Delay Propagation Model

A great inspiration to this thesis has been Dekker's work titled "Geographic delay characterization of railway systems" [31] where, using spectral clustering, four national railway systems are partitioned with the use of a delay propagation model. The model consists of two parameters. One parameter α_{ij} models the proportion of outgoing traffic-flow from a station i along an edge towards station j. The second parameter, β_{ij}, models how the delay changes on average when going from i to j. These two parameters together form a product that describes how much delay is propagated and changed between stations. Using this model as the weights in a weighted adjacency matrix, Dekker clusters the network and analyzes the results with two delay specific metrics.

2.5.1 Model

The α-parameter consists of two factors. The first one models the direct proportion of outgoing traffic from a node along each of its outgoing edges edge. As a result α will be higher for edges with large frequency with respect to the other outgoing edges of a node. The second factor models another spatial non-uniformity. In railway systems the running time is the time to traverse a section of track. Some areas have short edges resulting in short running times τ. This is used to create a second factor combining the stations average running times with the system-wide minimum, τ_{min}. Together the two factors combine and model the spatial non-uniformities found in the traffic flow, α:

$$\alpha_{ij} = \frac{f_{ij} \cdot \tau_{min}}{\sum_{j'} f_{ij'} \cdot \tau_i}$$

where f_{ij} is the number of departures going from station i to j.
Note that all the prerequisite variables to calculate α are attainable from the timetable and thus only rely on the planned traffic flow, not actual statistics. To also model the delays in the empirical data a parameter β is introduced: it describes the factor at which delay changes when traversing between the two stations and it is defined as such:

$$\beta_{ij} = \frac{\langle D^{ij}_{arr} \rangle}{\langle D^{ij}_{dep} \rangle}$$

where $\langle D^{ij}_{arr} \rangle$ and $\langle D^{ij}_{dep} \rangle$ are the average observed delays (relative to the timetable) when arriving and departing from i to j. The intuition is that $\beta > 1$ means that delay increased along the edge while $\beta < 1$ means it decreased. Dekker calls this factor the spatial non-uniformity in delay statistics. Together these two spatial non-uniformities reflect the geographic spread of delays across a rail system. They combine to create a matrix M of the system:

$$M_{ij} = \begin{cases} \alpha_{ij}\beta_{ij} & \text{if } i \text{ and } j \text{ are adjacent,} \\ 1 - \sum_k \alpha_{ik}\beta_{ik} & \text{if } i = j, \\ 0 & \text{elsewhere.} \end{cases}$$

This matrix is actually a *transition matrix* (or *probability matrix*) and, as mentioned in Section 2.2.4, it needs to fulfill certain properties. This is why Dekker normalizes the diagonal as to make each row sum to one. Interestingly, he also mentions that this matrix corresponds to a *dynamical model*, where a transition describes how an initial delay vector develops in the system after a time step dt.

2.5.2 Clustering

After creating the model denoted by the matrix M, Dekker uses this as the weighted adjacency matrix W and performs Spectral Clustering as described in Algorithm 1. As a consequence, the stations in the railway network are assigned to clusters based on the strength of their connections in the delay model.

2.5.3 Clustering Visualization

After spectral clustering is performed, how does one best *visualize* the results? Dekker proposes to showcase the clustering results in two different views.
The first is a *physical view*, which is done by plotting the clusters as colored polygons on top of the original graph. The polygon locations correspond to the actual positions of the nodes in that cluster. This makes it easy to get an overview how the physical system is partitioned by the clustering, for example, where exactly are the cluster borders? Different clusters receive different colors and are therefore easily distinguishable when looking at the visualization.
The second figure that Dekker uses to present his clustering results is an *abstract view*. This abstracts the physical view of the clusters into a graph, with the clusters themselves being nodes, and edges representing whether or not two clusters have any connection in the physical view. The node positions are dependent on the position of the cluster in the physical view, which gives the figure some resemblance and familiarity when comparing the two views. Additionally, the size of the edges between the clusters depend on the sum of weights between the two clusters, whilst the size of the nodes are determined by the number of stations in that cluster. This enables one to get more insight how the "model is thinking", for example, what kind

of cluster-border is between two clusters? Also, note that one could extend this visualization to show any kind of metric as edge- and node-size.

2.5.4 Clustering Metrics

To gain deeper insights into the clusters, Dekker constructed specific *cluster metrics* [31]. These metrics are single values calculated for each cluster based on their properties and, optionally, how these properties compare to the rest of the system (relative properties). One of the metrics defined is called delay severity, and it's equation is defined as such:

$$S(n) = \frac{D_{total}(n)}{\frac{1}{N}\sum_{i}^{N} D_{total}(i)}$$

where:

$$\begin{aligned} S(n) &:= \text{delay severity in cluster n} \\ D_{total}(n) &:= \text{the total delay in cluster } n \\ N &:= \text{total number of clusters} \end{aligned}$$

Delay severity quantifies relative cluster delay significance. Clusters with a delay severity of more than one have more severe delay totals than an average cluster and vice versa. This metric makes it easy to identify dominant clusters from a delay perspective. Delay independence is a ratio between the exchanged internal delays in a cluster with respect to the external ones:

$$I(n) = \frac{D_{int}(n)}{\sqrt{G(n)} \cdot (D_{imp}(n) + D_{exp}(n))}$$

where:

$$\begin{aligned} I(n) &:= \text{cluster independence in cluster n} \\ D_{int}(n) &:= \text{the total internal delay in cluster } n \\ G(n) &:= \text{the number of stations in cluster } n \\ D_{imp}(n) &:= \text{the total imported delay in cluster } n \\ D_{exp}(n) &:= \text{the total exported delay in cluster } n \\ N &:= \text{total number of clusters} \end{aligned}$$

The term $\sqrt{G(n)}$ counteracts the bias where large clusters automatically exchange less with other clusters than they do internally.
These metrics do not only help with the interpretation of what is happening inside a single cluster, it also enables to put the clusters on a scale based on their metric value, thereby making it easier to compare different clusters. This scale can even be two-dimensional, as in Dekker's paper, where clusters are compared to each other by plotting their $S(n)$ and $I(n)$ values on two axes.

3
Dataset

3.1 Data Generation

Over the course of an AGV system running, no matter if its simulated or physical, certain information about the system can be saved. This output, also called *trace files*, contain key signals of what communication is happening between the AGVs and the fleet manager, as well as detailed data of individual AGV positions, over time. It can be thought of as time-series data that describes system events in chronological order. This data enables analysis and understanding of traffic flow in relation to the layout, which is one of the key goals of this report.

3.2 Dataset Components

To analyze an AGV system, we need two components; the layout graph and empirical data of system dynamics (time-series data).

Layout Graph

The layout is loaded in a proprietary file format, and it contains all the properties of a layout as mentioned in Section 2.1.1. What we use, though, is only the graph-related properties of the layout; namely node positions and their connectivity, as well as edge directionality and spline shape. Other layout properties are not of importance for this report.

Time-series Data

As mentioned above, data can be saved in trace files while an AGV system, either simulated or real, is running. For example, it is possible to understand that at timestamp t, AGV i was blocked at segment x due to AGV j occupying segment y. Detailed blocking information such as this, as well as vehicle positions and distribution of missions over the course of the simulation, is what we mainly focus on in this report.

3.3 Our dataset

The dataset that will be used in this report is based on a system called *NDC8Demo*, see its layout graph below in Figure 3.1. This is a demo system that is created to be representative of how real AGV systems typically look like. Therefore we will make the assumption that it is good enough to showcase our methods on.

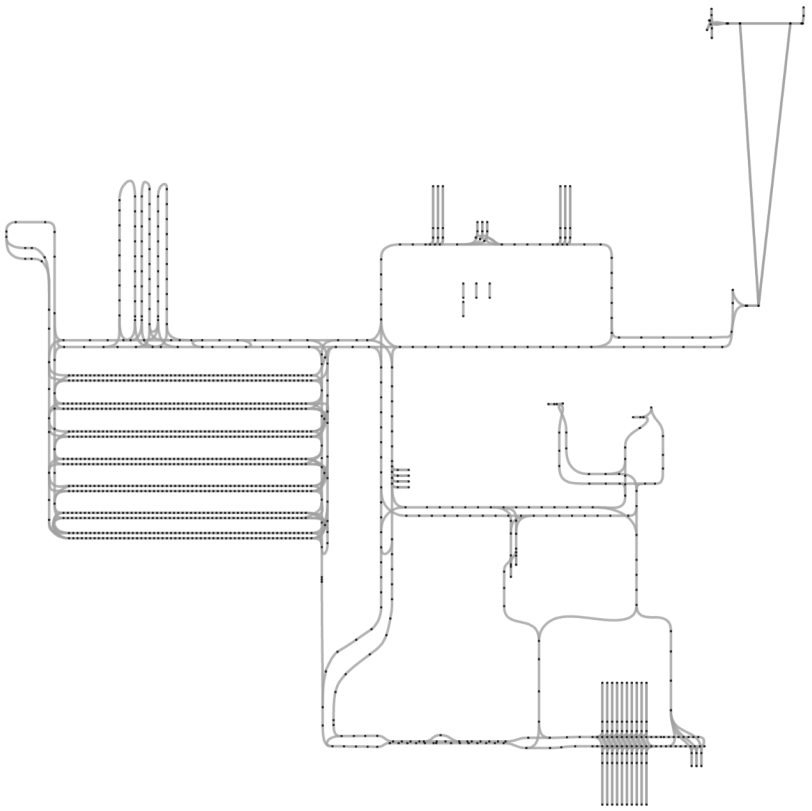


Figure 3.1: The layout we use as our base system, NDC8Demo. This layout is designed for educational purposes to teach layout designers how a real AGV network is often designed.

NDC8Demo has some some typical characteristics of an AGV system, most notably that it is sparsely connected meaning the nodes have a low average degree. As can be seen in Table 3.1, most nodes only have two neighbors, corresponding to one incoming edge and one outgoing edge in the directed graph. Additionally, the average betweenness centrality is quite low, suggesting that there are on average multiple ways to traverse between two different nodes. This is also a typical quality for AGV systems, since it correlates quite well with designing to avoid congestion. If the reader is not familiar with the term betweenness centrality, it is defined as a

Layout	Nodes	Edges	Degree	Betweenness Centrality
NDC8Demo	1386	1604	2.3	0.05
Real system 1	1675	3264	3.9	0.02
Real system 2	100	204	4.1	0.1

Table 3.1: Some graph insights into the three systems to show similarities and disparities that can exist between systems.

metric on the frequency of times a node is on the shortest path between all other pairs of nodes in the graph [32]. Table 3.1 shows, for comparison, the characteristics of two real AGV systems. Although these systems are more well-connected they still are quite sparse, and a low betweenness centrality is still common among all three. NDC8Demo is not only a demo of how real AGV systems typically look, it is also demo of the best practices in layout design. Is is even used to teach layout designers. This means that is has few issues, and, since one of the main purposes of this report is to find problematic areas of a layout, this is not ideal. Additionally, only using one system as a dataset would also not be ideal. Given this, three slightly different variants of NDC8Demo will be added to the dataset. Each of these systems have some small change in them to try and strengthen a specific behavior not found in the original system. The complete dataset thereby consists of four different systems, and these will be summarized below. The exact changes introduced in each system is also described.

1. *Base System*: Base version of NDC8Demo. Behavior mimics how a real AGV system could behave.
2. *Separated System*: Designed specifically to have two separated lines of traffic.
3. *Overlapping System*: Designed in a way that makes a certain section of the layout contain overlapping lines of traffic.
4. *Congestion System*: One intersection designed specifically to be very slow, ensuring that a lot of blockings will propagate out from this area.

With these slightly different systems and scenarios we expect to see varying behavior in our methods, thereby verifying our methods are extensible to AGV systems of different kinds and at different design stages.

4
Methods

4.1 Domain Specific Definitions

During the project, some domain specific terminology was missing. To support the text better, some new definitions are created, and these will be described below.

4.1.1 Blocking

To capture the AGV blockings propagating through a system we first need to model a blocking. Each instance of a blocking happening in the system creates its own blocking object. A blocking object has a start- and end-time, a blocking reason (more on this below), and all the necessary information about the status of the blocked AGV(s), *blockee(s)*. This status information contains key data for each blockee, such as vehicle id, planned path, target station etc. Understanding the intentions of the blockee can help the user determine if a blocking was inevitable or perhaps the result of a poor layout design.

4.1.2 Blocking Reason

As mentioned above, each blocking object also contains information about the reason for the blocking occurring. Going back to the generalization of traffic rules in Section 2.1.3, its important to state that a blocking is an instance of a traffic rule being activated, whereas a blocking reason is the actual traffic rule itself. In our system, we limit ourselves to capture two types of blocking reasons:

1. those due to other AGV(s), called *AGV blocking*, and
2. those due to signals from external systems, called *signal blocking*

Note that in the case of an AGV blocking, the AGV(s) responsible can be called *blocker(s)*.

4.1.3 Blocking Graphs

To capture blocking propagation in the system, we use a data structure where blocking objects are represented as causal chains. Any prior blockings that directly causes a new blocking are considered parents, while the new blocking is the child. Consequently, a blocking can have multiple parents and multiple children. This becomes a recursive data-structure, and is suitably represented as a directed acyclic graph (DAG) [33]. The graph is directed because the parent/child relationship is directional, and it is acyclic because blockings occur in chronological order. This DAG

of blockings allows us to chain together every single blocking in the system with its causes and repercussions. In figure 4.1, one can observe an example of how a blocking graph can look. By applying recursive functions to a specific blocking, we can traverse the data-structure to find out different properties about that blocking. Such as what the blocking reason (root cause) is or how severe the blocking is. By aggregation on specific data points, even more advanced insights can be yielded. For example "On segment X the most common blocking reason is Y and the average blocking time is Z". Note that if the data contains deadlocks, such as two vehicles blocking each other, it would create a cycle in the graph, thereby contradicting the acyclic property of the DAG. Deadlocks signify some design flaw in the layout and the issue of addressing them is already something that Kollmorgen is doing through other means, and therefore it is not a focus of this report.

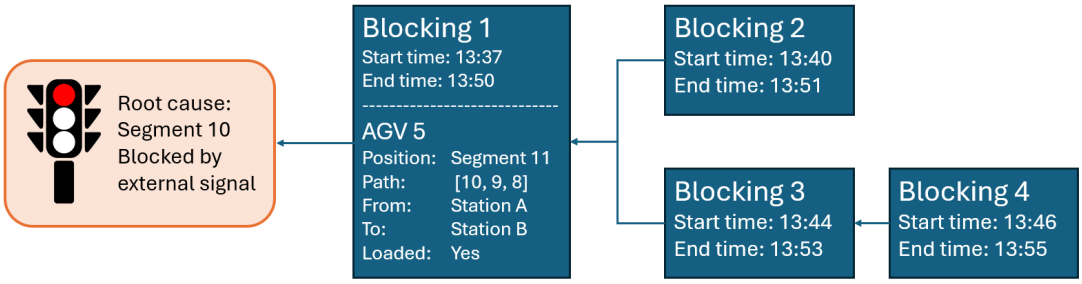


Figure 4.1: A small example of how a blocking graph can be structured. Blocking 1 is also expanded to show some of the AGV status as well. This blocking is caused by an external signal at the next segment in its path. Because of Blocking 1 two additional vehicles are blocked. This is shown as Blocking 2 and 3 being linked as children to Blocking 1. In this scenario a fourth AGV is also blocked as a reason of Blocking 3.

4.2 Micro-level Descriptive Analytics

To answer the first research question, the idea is to work on understanding the system on a micro-level. Ideally, one would like to show the data as is, but as mentioned in chapter 1, the simulation output is difficult to interpret. It is also not trivial to extract information from *chains-of-blockings* (what we call blocking graphs), since the time-series data simply only contains the first order blocking reason.

4.2.1 Case Study

Before extracting the data, it was important to understand exactly what parts of the data are necessary as well as in what way to best present it. Since the target audience mainly are layout designers, many interviews were carried out to answer these questions and find out more in detail what the pain-points are when designing a layout. This not only gave us direction and motivation, it also greatly increased our knowledge about the AGV domain and how to align our work with its unique challenges.

4.2.2 Parsing and Blocking Graph Creation

As mentioned in Section 3.2, the simulation output is in the format of time series data, also called *trace files*. Since the original data is unsuitable for our purposes and contains more information than needed, there is a need to first parse it and then save only whats relevant. The purpose of parsing is to populate our data structures with information as recorded in the system.

The trace files contain a lot of different types of information, we only need to look for messages containing information relevant to our output structures. The output from the parsing stage will be the following:

- Segment counts: A dictionary that contains the number of AGV traversals per segment. It updates when a message indicates that an AGV has moved into a new segment.
- Mission counts: A dictionary that contains the number of AGV missions between pairs of stations. It updates when a message indicates the start of a new AGV mission, i.e., when an AGV is dispatched to a new station.
- Blocking graphs: A list of all blocking graphs. Since each blocking is part of a graph, either individually or with other blockings, this list will encompass all blockings that occurred in the trace files.

While the segment and mission count outputs are straightforward to populate, the blocking graph list requires two additional temporary data structures during parse time. These structures keep track of the system's state during parsing:

- AGV statuses: Keeps a snapshot of the most recent status update for each AGV in the system. This is needed since when a blocking object is created since we want to be able to model each involved AGVs current state at blocking time.
- Active blockings: A list of blocking objects that have been initiated but are still ongoing.

To ensure the continuous blocking update logic is straightforward, these two data structures are updated with relevant information. When a new blocking occurs, a blocking object is created using data from the blocked AGV and the comprehensive blocking reason. The latest AGV status is loaded from AGV statuses, which contains the most recent snapshot of each AGV's status. This new blocking object is then added to the active blockings data structure.

If the new blocking is due to an already blocked AGV, the newly created blocking object is appended to the existing blocking graph as a child of the blocked AGV's blocking object. Otherwise, the new blocking object is considered the root of a new blocking graph and is added to the list of blocking graphs.

When parsing determines that a blocking is released, meaning the blocked AGV is allowed to continue its path, the blocking object is *ended*. This involves assigning an end time to the blocking object and removing it from the list of active blockings. Consequently, no more blockings can be appended as children to this blocking object since it is no longer active. For a brief overview of the blocking object creation logic see Figure 4.2.

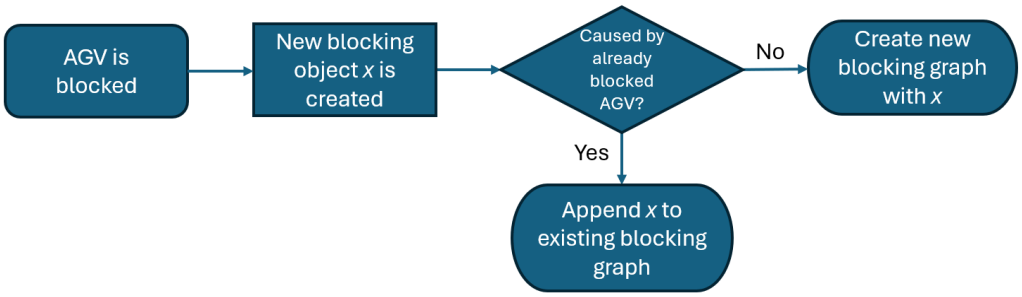


Figure 4.2: Flowchart showcasing logic for creating a new blocking object.

4.2.3 Visualization Tool

We have compiled a set of data structures containing valuable insights into the systems traffic dynamics. To make this data accessible and facilitate the discovery of insights, we decided to develop a simple *visualization tool*. This tool will feature various views to display multiple metrics and graphs, leading to the following specifications.
The app should

1. let a user interactively change between views
2. within a view let a user change between different objects
3. easy to prototype (the focus of the project is not this web-app)

To achieve this type of interactivity, Plotly Dash[1] was used. Designing an interface with this library allows the user to analyze and move seamlessly between different views of the same data and even switch what data is being shown. We theorized that enabling this type of workflow was highly crucial to be able to understand the complex traffic dynamics happening in an AGV system. The end product will be a simple and lightweight tool that allows the user to further examine the traffic dynamics in terms of the extracted data structures. Additionally, the tool and its core ideas should be easily extendable. This is important since we do not consider it a polished and finished product but rather a helpful research tool and a step in the right direction to understand the AGV system better.

4.3 Macro-level Modeling

To answer the second research question, we propose methods for uncovering and characterizing areas of interest in the layout, thereby better understanding the system on a macro-level. To apply the relevant theory on AGV systems specifically, certain adaptions have to be made. We start by describing and motivating these in detail. Uncovering areas of interest is done by applying the theory described in Sections 2.2 and 2.3 on the dataset, combined with exploring different techniques to model system behavior, as we explore an already existing method and novel ones. To characterize different areas and relate them to each other, we propose different ways to visualize the results.

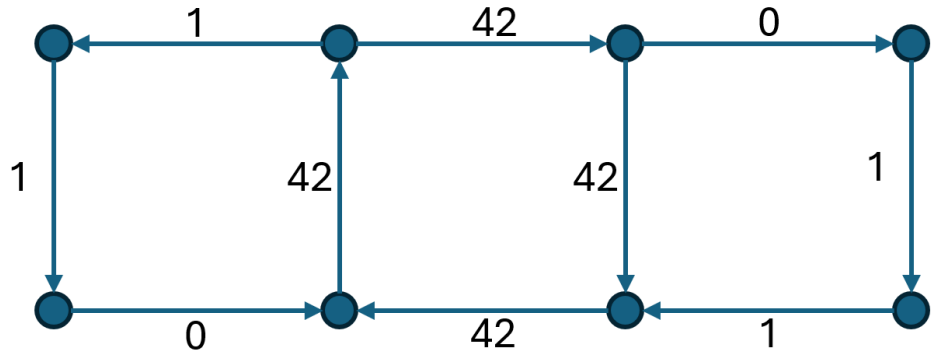
[1] `https://dash.plotly.com/`

4.3.1 Layout Preprocessing

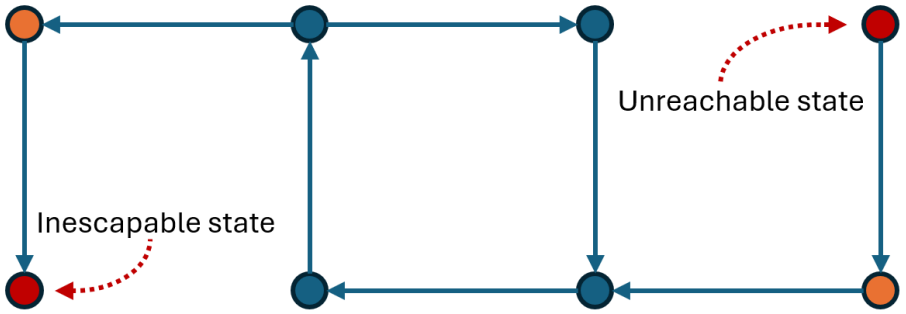
An AGV layout can be quite complex and when trying to analyze it using the proposed theory, there are some issues that quickly become apparent. Especially given the strict graph properties required for spectral graph theory. In practice, this means that some preprocessing steps are essential before even converting the layout to an adjacency matrix.
A layout consists of many segments, with some of these even corresponding to different strongly connected components in the layout graph. This could for example be by design, where one purposefully decides to make two separate layouts in the same system. As mentioned in Section 2.3, if a graph already has multiple components, this is already a natural partition of the graph. It does not make sense to look at these components as one graph in the perspective of spectral clustering. In this case, one would rather apply the theory on each of the components instead. In typical AGV layouts all the movement happens in the same component. Therefore, the steps that follow are assumed to be applied for a single strongly connected graph (or otherwise applied individually for each strongly connected component).
Even after identifying a single component, there is still the issue that some segments might not become visited during the system running. This depends on various factors such as simulation length, order flow and layout topology, and even then, some segments only allow certain vehicle types. These together, lead to the issue of some segments having no corresponding data points, which can cause problems later. Therefore, the first preprocessing step consists of removing all unvisited segments from the graph.
When removing unvisited segments, it can have the side effect of potentially creating new strongly connected components. This is undesirable, since we remind the reader that the relevant theory for directed graphs strictly requires strong connectivity, and connectedness for undirected graphs. For our case, we propose to use the undirected version of the graph. In contrast to the directed graph, we can guarantee that this results in a graph that is connected. The reason we can guarantee this for the undirected case and not for the directed one, is that if a segment is visited, it must be reachable from all other segments in the undirected graph - this comes from the property mentioned in Section 2.1.1, which states that all AGVs need to traverse on a continuous line of segments. For a directed graph, this is not true, since then there is the possibility for *unreachable* or *inescapable* states, see Figure 4.3. One could argue that it would be possible to prune these states, but there is also sense in not removing more data than absolutely necessary.
The disadvantage of using the undirected graph instead of the directed, is that some information is lost. At this stage it is unclear what impact it could have on the results, so it might be an interesting future research direction. In this report we have made the decision to focus on the undirected graph though, so we will have to return to this topic more in detail in a later chapter.

4.3.2 Layout Adjacency Matrix

With the layout now being preprocessed to fulfill the required properties, we move on to another integral part of Spectral Clustering, the Laplacian matrix. This matrix



(a) Before preprocessing. Values are the segment counts of each edge.



(b) After preprocessing.

Figure 4.3: In this example we highlight a potential issue with directed graphs. By removing segments that haven't been traversed there is a risk to create inescapable and unreachable states. Since that will create a graph that is not strongly connected.

is based on the (weighted) adjacency matrix, and it needs to be well-defined before applying the theory. Therefore, we now wish to define these matrices for the AGV layout graph. First for the simple case when the layout has no weights, i.e., only connectivity, and then continuing with the weighted graph definition below.
Creating the adjacency matrix might at first seem like a trivial step, but, due to the layout possibly having multiple segments between each pair of nodes, it actually requires some careful thinking. For our case, we simply consider multiple parallel edges as a single connection, and additionally disregard direction. Thereby, we end up with the symmetric adjacency matrix, which for clarity is stated as:

$$A_{ij} = A_{ji} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0 & \text{elsewhere.} \end{cases}$$

For defining the degree matrix, the reader is referred back to Section 2.2. Returning to the case with a weighted AGV layout graph, now comes the issue of determining the weights for each connection ij in the adjacency matrix. For this, a *model* will be defined. It is defined as a scaling function w that given a pair of nodes i and j should output a value which is used to scale the corresponding entries in the

adjacency matrix, i.e. A_{ij} and A_{ji}. The reason that w is called a scaling function is to be clear that only connected nodes should be able to receive a weight - by scaling the entries of the adjacency matrix, this is precisely what is achieved.
Although the model could be defined arbitrarily, it is of more interest for the weights to be scaled proportionally to some system behavior that one wishes to analyze, for example segment count or blocking time. The issue which needs to be addressed, though, is that the metrics are measured on a segment-level, not between node-pairs. To convert from the domain of segment-metrics to the domain of node-pair-metrics, we need to aggregate all data between each pair of nodes. The average operation is suitable, to keep the values numerically stable and comparable across different simulation lengths. Practically, this means that the node-pair-metric corresponding to connection ij is averaged across all occurrences of data flowing immediately from i to j or j to i. After this, the segment-metrics are translated to node-pair-metrics, which easily can be used to define models w. Additionally, note that the model w ideally should be designed so the output is non-zero for all connected nodes, since otherwise the connectivity constraints corresponding to the original graph could be violated. Again, this is the reason why unvisited segments, i.e. segments without data, were removed from the layout graph in the previous subsection.
Given this, the entries W_{ij} in the weighted adjacency matrix can be calculated by scaling each entry A_{ij} with the corresponding model output. The weighted degree matrix is defined just as in Section 2.2, but it will be written here as well to be explicit:

$$W_{ij} = W_{ji} = A_{ij} w(i,j) \quad \text{for all nodes } i \text{ and } j.$$
$$D_{ij} = \begin{cases} \sum_k W_{ik} & \text{if } i = j,\ k = 0 \dots N, \\ 0 & \text{elsewhere.} \end{cases}$$

4.3.3 Layout Spectral Clustering

As mentioned in Section 2.3, all that is required to apply spectral clustering is a graph that fulfills the required properties and its Laplacian matrix, which in turn is based on the (weighted) adjacency matrix. After the previous two sections, it should be clear how the layout is preprocessed and how the (weighted) adjacency matrix is created, either without a model (only connectivity) or with a model (weights proportional to some system behavior). With this done, we can now propose how to apply spectral clustering on the AGV layout.
The steps we follow are analogous to the *general spectral clustering* algorithm (see Algorithm 1), but with some minor changes to account for the special handling required for our purposes (see Sections 4.3.1 and 4.3.2).
In addition to this, some crucial output of these steps are saved as figures to enable further analysis. Step 4 of Algorithm 2 produces an eigenspectrum, which is saved in a figure, including what k was chosen. Step 5 and step 6 creates a spectral embedding using the first k eigenvectors, and the binary partition of the layout as described by first eigenvector (the Fiedler vector in Section 2.2), is also saved.
Note that it is important to carefully consider the choice of model w, as it completely

Algorithm 2: Layout Spectral Clustering Algorithm

Input: AGV layout graph G
Output: Clustered nodes

Preprocessing: Preprocess G to fulfill the properties required by the theory;
Step 1: Use the preprocessed G to create the adjacency matrix A;
  Step 1.1: Optionally define a model w;
  Step 1.2: Create the weighted adjacency matrix W by scaling A with the model w;
Step 2: Create the Laplacian $\mathcal{L}$ by using W (or A if no model w was defined);
Step 3: Choose a k_{max} for the eigengap heuristic based on layout size and problem description needs.;
Step 4: Run the eigengap heuristic algorithm (with k_{max}) on $\mathcal{L}$ to compute a suitable k;
Step 5: Compute the first k eigenvectors of $\mathcal{L}$;
Step 6: Form a matrix U using these eigenvectors, where row l of U corresponds to the features of node l;
Step 7: Apply k-means clustering on the rows of U to cluster the nodes of G;

changes the clustering results as well as the intuition of what the clustering means. Below, this will be addressed in more detail.

4.3.4 Modeling System Behavior

As mentioned earlier, the choice of model w is quite arbitrary but at the same time crucial. The key aspect is that the model output should be proportional to some property of the layout that one wants to analyze. So, in this section, the wish is to propose different models of system behavior, and to motivate their intuition and suitability in terms of modeling AGV system traffic dynamics.

Delay Model

As mentioned in Section 2.4.2, Dekker proposed a model for partitioning a railway system into separate regions based on where delays are expected to spread more easily [31]. The wish is to translate this model to the AGV domain as accurately as possible, and then apply it. The parameter α, basically corresponding to proportional outgoing traffic frequency, can relatively easily be translated to an analogous definition, but the delay factor β, on the other hand, is not as easily defined. The notion of a timetable does not exist in the AGV system and therefore, the concept of delay is not clear.
Before trying to solve this issue of defining β, we turn to the details of α. In Dekker's paper [31] there is a term τ in the formula, which purpose is to scale α inversely proportional to the segment running time, commonly referred to as segment weight in the AGV domain. This might be relevant in the railway domain - since segments with longer running times propagate less delay average - but for our domain it certainly is not; in the AGV system, segment weight has no connection to the

amount of delay that is propagated on average (since under normal circumstances the traversal time is fixed). Therefore the choice was made to remove this term. This means that α is purely dependent on the traffic frequency f, and consequently is defined as follows for each segment ij:

$$\alpha_{ij} = \frac{f_{ij}}{\sum_{j'} f_{ij'}}, \text{ where } j' \in \{\text{nodes connected to } i\}$$

Returning the focus to β, the issue of not having a timetable was solved by fabricating one during parse-time. This is done by using the expected time it takes to traverse a segment compared to actual time it took, with the difference between these then being the delay for that segment. This required a new type of measurement to be introduced in the parsing of the trace files, called *delay per mission*. When an AGV sets out for a new mission, e.g. heading towards a new station, the delay of the mission is initiated to zero. Then for each segment traversed the added delay is incremented to the total delay in that mission:

$$\begin{aligned} d_0 &= 0, \\ d_n &= d_{n-1} + t^n_{actual} - t^n_{expected}. \end{aligned}$$

where:

$$\begin{aligned} n &:= n\text{th segment in the current mission}, \\ d_n &:= \text{total delay after traversing } n\text{th segment}, \\ t^n_{actual} &:= \text{actual traversal time of } n\text{th segment}, \\ t^n_{expected} &:= \text{expected traversal time of } n\text{th segment}. \end{aligned}$$

Gathering this data over the course of a simulation creates multiple pairs of delays per segment. Each pair indicates an instance of a delay before and after traversing a segment. Then, these two values are averaged over the whole simulation, in turn enabling us to define β in a similar fashion to Dekker:

$$\beta_{ij} = \frac{\langle D^{ij}_{after} \rangle}{\langle D^{ij}_{before} \rangle}$$

where:

$$\begin{aligned} \langle D^{ij}_{after} \rangle &:= \text{average delay after traversing segment ij}, \\ \langle D^{ij}_{before} \rangle &:= \text{average delay before traversing segment ij}. \end{aligned}$$

With an almost direct translation of the terms used by Dekker, we now have obtained a definition of two segment-wise metrics, α_{ij} and β_{ij}, which can be used similarly as described in the theory to model edge-weights $\alpha_{ij}\beta_{ij}$. The only step that is left now is to convert these segment-level metrics to node-pair metrics. This conversion is done precisely as described in Section 4.3.2. Which in turn yields the model-function w:

$$w(i, j) = \alpha_{ij}\beta_{ij}.$$

This model is not perfect though. In contrast to trains, AGVs don't actually run based on a timetable so the idea of a associated mission delay for an AGV doesn't really make sense. An absence of a timetable results in a lack of actual delays. Even with adaptions the AGV delay parameter doesn't create a fully satisfiable model. This is described more in detail later in the discussion of this report.

Topology Model

Now, we present the option of simply using uniform weights, corresponding to the adjacency matrix. This is well-supported in literature and analogous to the case of applying the theory on an unweighted graph. As a bonus, this modeling does not even require any knowledge of the system behavior. Here is the simple definition:

$$w(i, j) = 1.$$

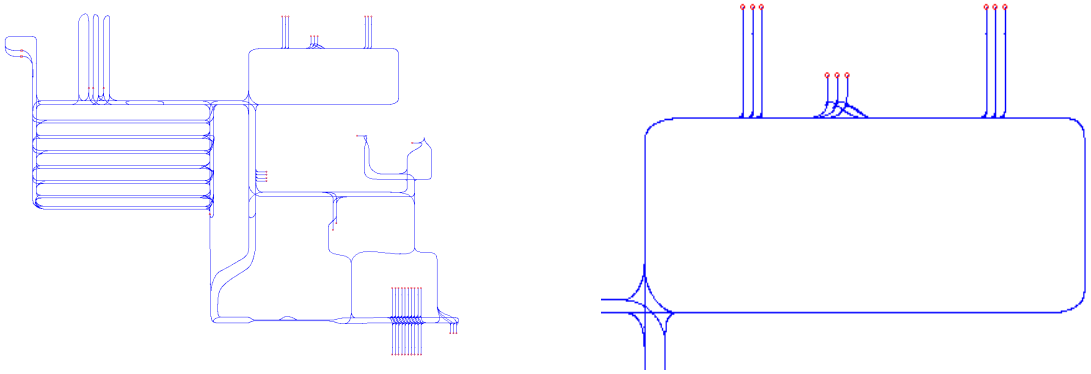
Since this means that all weights are equal, all that matters for the clustering results is the graphs topology, hence the title *topology-based modeling*. The intuition of this is that the clustering is based on structural patterns in the graph, with densely connected areas roughly corresponding to cluster-centers and sparsely connected areas to cluster-borders. Specific areas of an AGV layout often correlate with certain connectivity, for example, areas close to load and drop-off bays usually are more well-connected whereas the areas corresponding to highways between stations are more sparse. The expected results are therefore that the clustering based on this model could potentially characterize different areas in the layout graph based on their structural properties (denseness of connectivity).
Another way to view this this model is that it can act as a *baseline model*, i.e. something to compare other models to. This is because no matter the weights, the topology will always have a certain impact on the clustering. This is due to the properties mentioned in section 2.3, which states that spectral clustering produces clusters corresponding to sparse cuts, in other words taking both connectivity and weights into account. If one compares the results from this model to other models, then one can might uncover what parts of the clustering emerge from topology and what parts emerge from the weights.

Traffic-flow Model

To distinguish further patterns, we now would like to introduce *traffic-flow* data in the modeling. The reason for this is to find topological features whilst taking part of the actual traffic dynamics into account. This might at first seem analogous to Dekkers α parameter, which models the *proportional* frequency. But traffic-flow is instead defined as the magnitude of the frequency, that is the number of visits per segment during the whole simulation, previously called the *segment count*. This metric is one part of the traffic dynamics, with blockings being the other part. Later in this section, blocking-based models and mixed models that take both blockings and traffic flow into account will be presented and motivated.
The frequency f_{ij}, denotes the aggregated segment count between nodes i and j, as described previously in this section. Thereby the *traffic-flow model* w is defined as follows:

$$w(i,j) = f_{ij}.$$

This modeling emphasizes the most visited node pairs, which in reality should correspond to the actual order-flows in the system. For efficiency reasons, the AGV layout and order-flow is usually designed so that there is one or multiple loops that absorb the majority of traffic, with the more in-frequent traffic happening outside the loop when AGVs need to perform some special action (charging, load cargo, drop-off etc.). These loops can topologically be seen as roundabouts, with exits and entrances. The clusters-borders of this model are expected be placed such that they distinguish disconnected loops, i.e. separate order flows, from each other as well as identifying when there are exits/entrances from a loop that lead to more in-frequently visited areas. See figure 4.4 for an example of such a roundabout traffic-loop.



(a) AGV road network

(b) Traffic-loop with exits and intersections

Figure 4.4: This figure shows a zoomed-out AGV road network in the left panel. In the right panel, we have zoomed in on a certain area of the layout. This area has a roundabout structure with exits and intersections. When clustering based on traffic, on might expect the cluster border to be where traffic diverges (i.e. exits or intersections).

Blocking Model

One part of AGV system traffic dynamics is the traffic-flow, as modeled above, but another core part are of course the blockings. Therefore we here wish to find patterns using *blocking-based modeling*, where the weights are proportional to the amount of time spent blocked. This requires some careful thinking, as in practice blockings are often quite sparse, leading to missing data for many segments. An example is if one would use a model like the following, based only on the aggregated blocking time $t^{ij}_{blocking}$ between each node-pair ij, defined as:

$$w(i,j) = t^{ij}_{blocking}$$

Since the majority of node-pairs ij in practice have a total blocking time of zero, it means that the weighted adjacency matrix will become *very* disconnected. One

could think that these disconnected components are actually the clusters we would want to see, but in this case one could simply look at the *micro-level descriptive analytics*. The wish with the *macro-level modeling* is to provide an overview, and therefore this result would not suffice. We want a way to connect these isolated components. One solution is to take inspiration from Dekker's β parameter and model the total blocking time as a factor. We will denote this parameter as γ and call it the *blocking delay factor*. Its definition is as follows, where we remind the reader that $t^{ij}_{expected}$ is the aggregation of the expected traversal times for every time one of the segments between node-pair ij got visited:

$$\gamma_{ij} = \frac{t^{ij}_{blocking} + t^{ij}_{expected}}{t^{ij}_{expected}}$$

This parameter will trivially have weight 1 one for node-pairs without blockings, and otherwise a factor of how much longer time it takes to traverse between node-pairs due to blockings (on average). Also, it does not suffer from the same disadvantages as β in the Dekker-inspired model. There is no need to fabricate a time-table and therefore many of the mentioned problems are fixed. We believe this parameter γ is more suitable to model the AGV system traffic dynamics than the β, with the main disadvantage being that the blockings can be too sparse - in the next section we seek to solve this issue by using a a mix of models.
With this as motivation, we now define the *blocking delay factor model* for each node-pair ij:

$$w(i,j) = \gamma_{ij}.$$

The intuition of clustering the layout based on this model is that stretches of segments that often have blockings occurring should be differentiated from areas of the layout that have no blockings occurring. Although, in practice, the blockings usually are quite sparse. This means that in a stretch of the layout edges with no blockings could be intertwined, and even alternating, with edges that have many blockings. This on and off behavior could lead to it being harder to distinguish problematic areas in the clustering results.

Congestion Model

As previously mentioned the traffic dynamics consists of both traffic-flow and blockings. What if we included both in a model? The idea is to use the two previously defined parameters f and γ in a *mixed model*, which we hope can find patterns in terms of both parts of the traffic dynamics; traffic-flow and blockings. We remind the reader that f is the traffic-flow, and γ is the blocking delay factor. Using them together we define the mixed model as such:

$$w(i,j) = \gamma_{ij} f_{ij}.$$

While γ explains how much average time is wasted on blockings it does not take into account the number of traversals on the segment. Thereby we mix it with f,

to give higher weights to segments with many AGV traversals. The reasoning being that clustering by the blocking delay factors alone could lead to a model that is too sparse, and by including the traffic-flow as well the model can be more flexible.

Naively, one could define w as above, but after giving it some more thought, it can make sense with introducing a *penalized traffic-flow model* to use in combination with the blockings. This model works by penalizing large frequency values f_{ij} using some penalty function g. The motivation for this is that severe blockings happening on high-traffic segments should be of the most importance, but severe blockings happening on medium- to low-traffic segments should also have a certain importance, and not be too overpowered by high-traffic segments with low blocking severity. Therefore larger frequency values are penalized whilst smaller ones are kept relatively similar.

For simplicity we denote the penalized traffic-flow for a node-pair ij as ω_{ij}:

$$\omega_{ij} = g(f_{ij})$$

Which finally gives the new definition of the mixed model w:

$$w(i,j) = \gamma_{ij}\omega_{ij}.$$

The decision of penalty function g can seem quite arbitrary, but there are certain important properties that the function need to have. Properties of g include;

1. Penalizing Larger Values: The penalty function should impose greater penalties on larger values compared to smaller ones. This helps to achieve a more balanced distribution of weights.
2. Preserving Relative Similarity: The penalty function should ensure that the relative distances between inputs remain similar after penalization. This means that if two inputs were close to each other before applying the penalty, they should still be relatively close afterward. This property helps maintain the inherent structure of the data
3. Maintaining Proportional Weights: The penalty function should keep the proportions between weights consistent over time. This means that the relative sizes of the weights should not change as the simulation progresses. This property ensures that the weight distribution remains stable and predictable throughout the simulation.

Whilst there are many functions g that fulfill the properties mentioned, a straightforward and computationally effective option is the square root function. So, from now on, assume that when ω_{ij} is presented it has the following definition:

$$\omega_{ij} = \sqrt{f_{ij}}$$

Summary of Models

Model	**System Behavior**	**Formula for node-pair** ij
Delay	Models delay per mission using a fabricated time-table.	$w(i,j) = \alpha_{ij}\beta_{ij}$
Topology	Models layout connectivity by using uniform weights.	$w(i,j) = 1$
Traffic-flow	Models traffic-flow by using the segment-wise segment count.	$w(i,j) = f_{ij}$
Blocking	Models blockings by using the segment-wise blocking time as a factor.	$w(i,j) = \gamma_{ij}$
Congestion	Models traffic congestion by combining the blocking model with penalized traffic-flow.	$w(i,j) = \gamma_{ij}\omega_{i,j}$

Table 4.1: Summary of models and their behaviors.

4.3.5 Cluster Visualization

When it comes to how to visualize what areas (clusters) are of interest, Dekker [31] proposes a suitable choice in what we previously called the *physical view* (see section 2.5). This visualization provides a solid overview about how clusters relate to the actual layout, and therefore we take inspiration from this idea and visualize the clusters in virtually the same way.

4.3.6 Cluster Relationships

Another part of the second research question is to be able to tell how different areas of the layout relate to each other. For this, we also propose to take inspiration from Dekker [31] and visualize cluster relationships using the *physical view* in combination with *abstract view*. Although our visualization is not exactly the same as Dekker, more on this below.
Using only the physical view of the layout do draw insights is quite abstract, since it is hard to say what impact the model has on the clusters compared to simply clustering based on topology. Therefore, it would be beneficial to abstract the layout into a certain number of areas and describe the system in terms of interactions between and within these areas. This is precisely what is achieved when using the abstract view. At the same time, not including the physical view at all would mean that one easily loses touch with the domain, in turn making it hard to gain any actionable insights. These are the reasons that we propose to use both views together when describing relationships between clusters.
Finally, as briefly mentioned in section 2.5, different metrics can be displayed in the abstract view. For our purpose, we propose using traffic-flow as the metric for edge size and total blocking time as the metric for node size. This approach highlights how traffic flows between different clusters and identifies which clusters are most

problematic in terms of blockings, thereby effectively visualizing how different areas of the layout relate to each other.

4.3.7 Cluster Characterization

The final part of answering the second research question is now the challenge of characterizing these different areas, or in other words, how to quantify which area is the most significant. To achieve this, we will define a number of *cluster metrics*. Putting the clusters on a scale like this makes it easier to see how different clusters stack up against each other, and also which one specifically is the most problematic in terms of the metric defined. Combining multiple metrics further characterizes the clusters in terms of more perspectives than just a single metric. Thus, three different key metrics have been devised for ranking clusters in terms of significance, these will be described below. Although, it should be noted that other metrics are possible than these, depending on how one would like to define a *significant* or *problematic* cluster.
The first metric we call *blocking severity*:

$$S(c) = \frac{B_{total}(c)}{\frac{1}{N}\sum_{i}^{N} B_{total}(i)}$$

where:

$$\begin{aligned} S(c) &:= \text{blocking severity in cluster c} \\ B_{total}(n) &:= \text{the total blocking time in cluster } n \\ N &:= \text{total number of clusters} \end{aligned}$$

Closely related to Dekker's metric *delay severity*, the blocking severity is a measure of how severe the blocking time is in a cluster relative to the other clusters. To increase the understanding and differentiate between clusters with similar degrees of severity, we further split the data with two additional metrics. First we have relative traffic density:

$$D(c) = \frac{\frac{f_{total}(c)}{E_c}}{\frac{1}{N}\sum_{i}^{N} \frac{f_{total}(i)}{E_i}}$$

where:

$$\begin{aligned} D(c) &:= \text{relative traffic density per segment in cluster c} \\ f_{total}(n) &:= \text{the total vehicle edge traversals in cluster } n \\ E_n &:= \text{number of edges in cluster } n \\ N &:= \text{total number of clusters} \end{aligned}$$

Traffic density highlights clusters with a relatively high amount of vehicles traversing through them. A cluster with high blocking severity but low traffic density could be a cluster worth further attention. The reasoning being that traffic in a low density area should flow better than in a high density zone. Anomalies should be spotted

and investigated. The final metric is simply cluster size. If a small cluster has severe blocking times it highlights a denser blocking behavior.
Displaying these three cluster metrics in a scatter plot should let a user understand which clusters are of interest. The two axis are representing blocking severity and traffic density, and the node size is determined by cluster size.

4.4 Development language and tools

Python

In recent years Python [34] has emerged as a very popular language for creating prototypes, as the straightforward syntax makes for readable and simple code. We hypothesize that the final program will not be bottlenecked by performance, so the faster implementation time with Python is a major advantage. This allows us to spend more time on other research areas. We also use some widely known packages for more specific tasks, these will be briefly presented below.

NetworkX

NetworkX is a Python library for working with graphs [35]. It provides tools to create, manipulate and study complex networks and graphs. We use it to convert the AGV system layout to a graph. This graph is used both for visualization and spectral analysis.

Matplotlib

For drawing simple figures and graphs, Matplotlib [36] will be used. It is basically the industry standard for creating visualizations in Python, and and even has built-in support for NetworkX.

Plotly Dash

For interactive visualization the project uses Plotly Dash[2]. It enables the creation of web apps that are capable of displaying multiple types of charts in an intuitive way. Importantly, the web app can be programmed to allow navigation between different views and communication between charts based on user click- and hover-events. In our case, for example, this is used to navigate a recursive data-structure by updating a chart based on click-events.

scikit-learn

To perform the clustering step in the spectral clustering algorithm (see 2.3), we use the k-means implementation from the Python package *scikit-learn* [37]. In the example from figure 2.3, the dataset generator functions `make_moons` and `make_circles` are used.

[2] `https://dash.plotly.com/`

SciPy

For answering the second research question there is a need to solve the eigenproblem for symmetric matrices, and therefore we employ the efficient solvers from the SciPy library [38]. Since the matrices corresponding to the graphs in our dataset are highly sparse, we use the eigen-solvers from the `scipy.linalg.sparse` module.

5
Results

5.1 Micro-level Descriptive Analytics

Here the results for the first research question will be presented. It is divided into two distinct parts, one is about the visualization tool, also called the *Trace file analysis tool*, and the other is about the empirical traffic dynamics observed for each system in the dataset.

5.1.1 Visualization tool

The visualization tool is started via a Python script and runs in a local browser as a web-app. It is capable of visualizing any AGV system as long as one has access to the dataset components mentioned in Chapter 3.
Upon running the tool the user is welcomed with the view in figure 5.1. There are two panels, the left one contains the layout graph and buttons to select different metrics, with the default being a *clear figure*, signifying that no metric is visualized. The right panel contains buttons for different visualization modes of data in the left panel, for example the default option *bar chart*, which shows a plot of the selected metric in the left panel.

Figure 5.1: Startup view of the visualization tool. On the left the user is presented with a physical view of the entire layout and a set of different metrics to display. Since no metric is chosen the bar chart on the right is empty.

Let us now select the metric *segment count* from the buttons in the left panel, the result of this can be seen in figure 5.2. Notice how certain parts of the layout graph

now are lit up, and that the bar-chart is updated. The layout is colored in a certain way based on what value the selected metric has for each segments, on a spectrum from red to orange to yellow, with larger values being red and smaller values being yellow. The gray areas of the layout mean that the metric has no value for these segments. So in the case of segment count, gray segments imply that an AGV never visited that segment in the data.

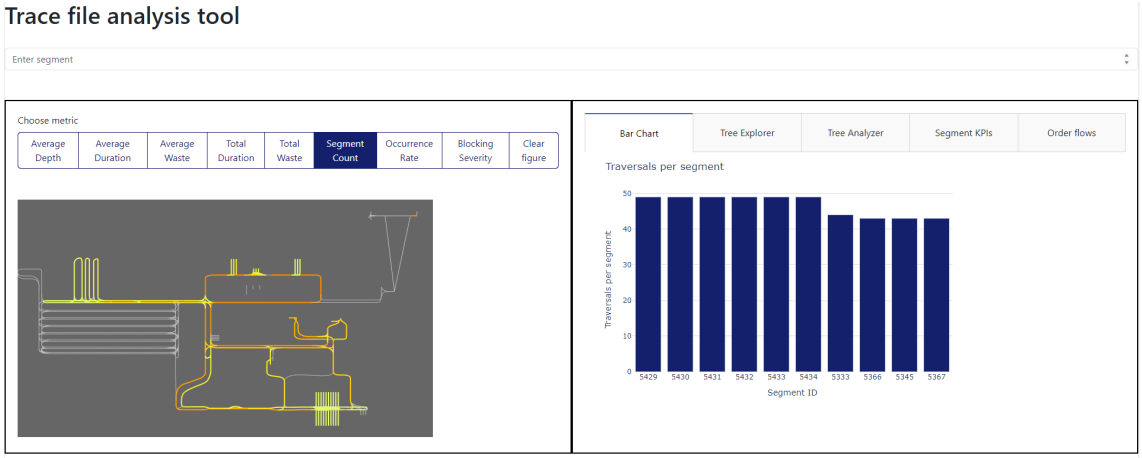
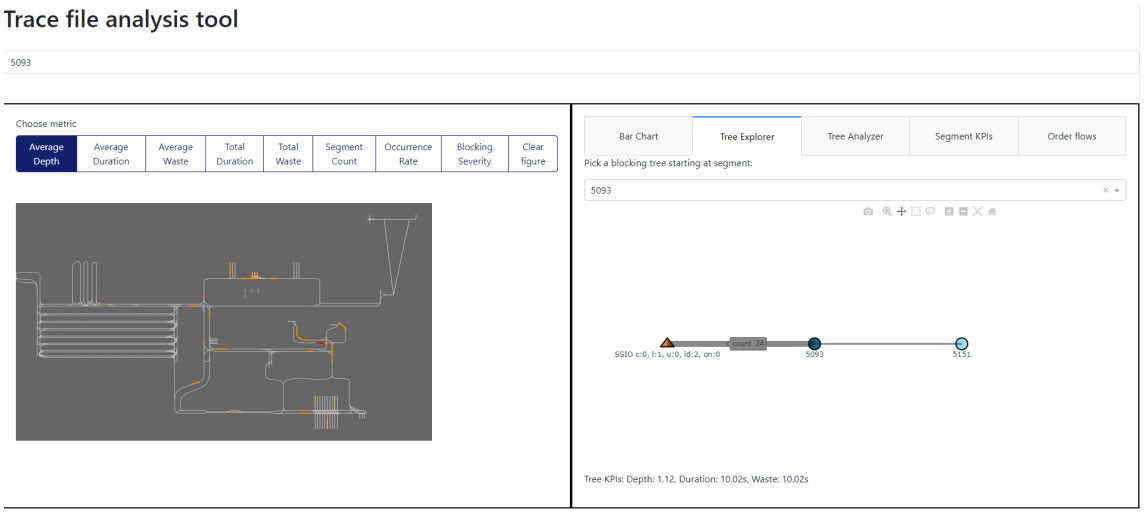


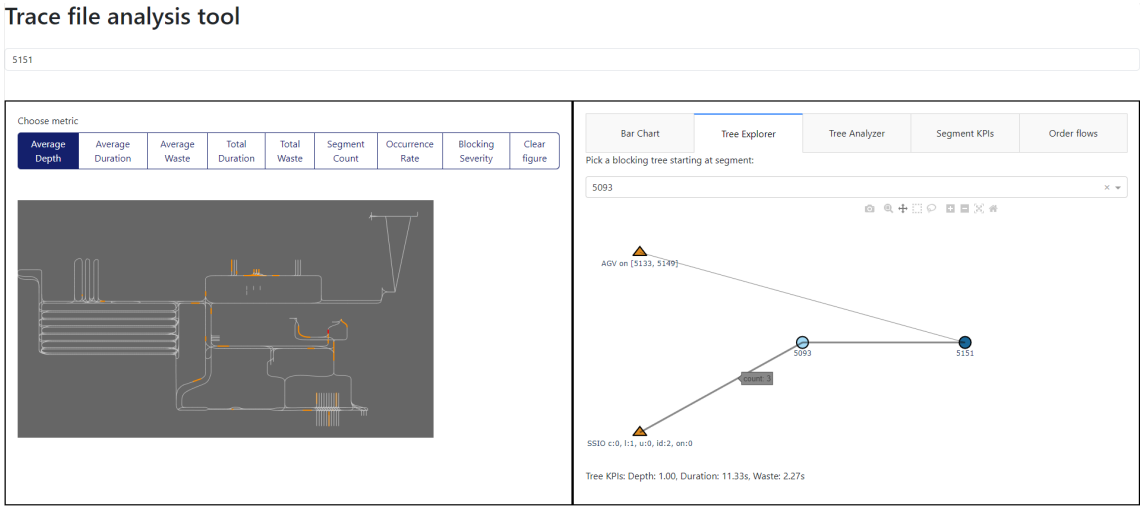
Figure 5.2: When selecting a metric the physical view and bar chart are updated instantly to display the relevant information about the metric.

Now we will move on to showcasing a different functionality. First off, the metric is switched to *average depth*. This metric corresponds to the average depth of all blocking graphs that started at a certain segment, and the corresponding layout graph with the metric shown can be seen in the left panel of figure 5.3a. Notice how much fewer of the segments are colored here compared to in figure 5.2, this is due to the sparseness of blockings, as mentioned in earlier sections. There is a segment that is bright red, and might stand out to the readers eyes. This segment is connected to the currently selected segment in the right panel of Figure 5.3a, which is why it is highlighted. That takes us to the new view in the right panel, the *tree explorer*. The name of this might be a bit confusing, but it simply comes from the fact that at the start of the project we planned that a single blocking should have a single blocking reason (meaning that in fact it could be called a tree), whereas later it made more sense to allow for multiple blockings reasons for a single blocking (in other words not a tree, but a DAG, as explained in Section 4.1). From now on, just assume that it can be called as a *tree* for simplicity (but keep in mind that it technically is a DAG).
Continuing the explanation of the tree explorer, in the right panel of figure 5.3a there can be seen a graph that shows three nodes connected by two edges. The nodes have different symbols, blue circle for an AGV blocked at a certain segment and orange triangle for a root blocking reason. With this said, the correct way to read this graph is as following: due to a *signal blocking*, denoted SSIO, an AGV was blocked 24 times at segment 5093. A fraction of these 24 times (signified by a much narrower edge), an AGV was blocked at 5151 due to the AGV being blocked at 5093. The reason that node 5093 is dark-blue is that the tree Explorer is currently

showing all blocking graphs where an AGV was blocked at segment 5093. If the user instead clicks on the node corresponding to node 5151, the graph morphs to display all blocking graphs where an AGV was blocked at segment 5151. The updated view is shown in Figure 5.3b, also notice how a different segment is now highlighted red in the layout graph. What the tree explorer is effectively doing, is that it aggregates all blocking graphs and allows the user to recursively traverse them up or down, that is towards the root-cause or towards deeper blockings.



(a) Tree explorer showing blocking graphs on segment 5093. As is shown the segment has been blocked 24 times by the same reason.



(b) Tree explorer showing blocking graphs on segment 5151. This segment has two different root causes as shown to the user.

Figure 5.3: Blocking graph explorer functionality

The tree explorer is a tool that can visualize arbitrarily complex blocking graphs. To showcase this more clearly we display a more intricate tree explorer example from one of the anonymized systems mentioned in Chapter 3, in Figure 5.4.
The final functionality that will be shown is *segment KPIs*, or segment Key Perfor-

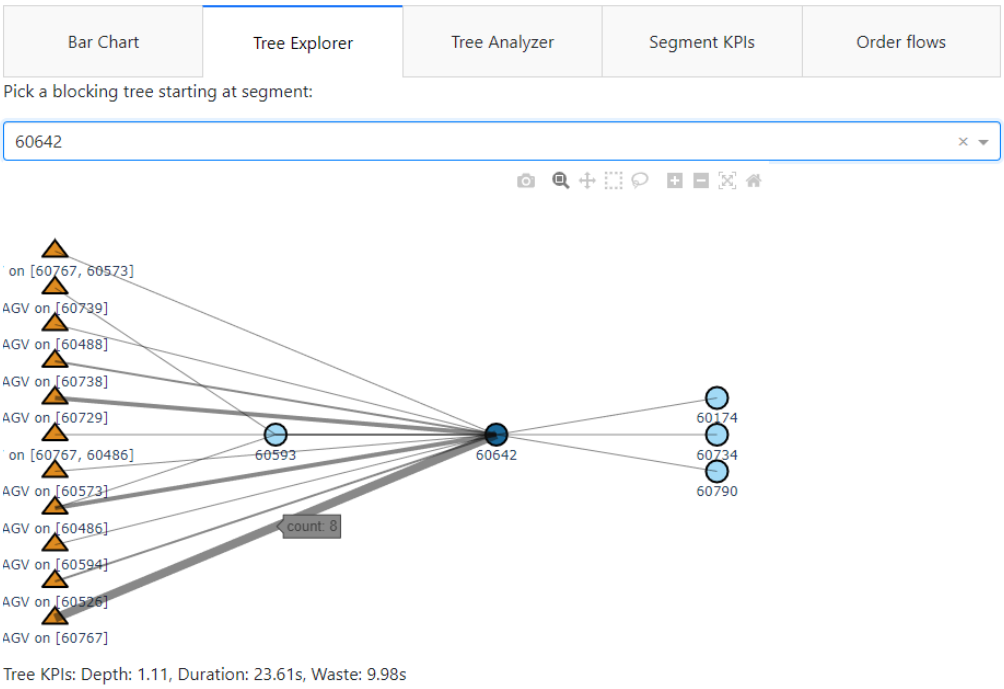
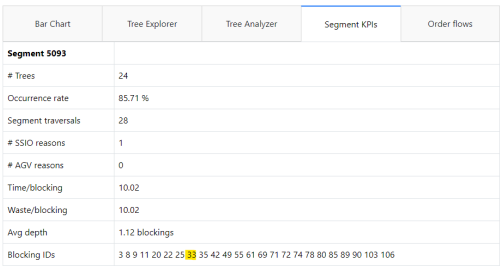


Figure 5.4: Here the tree explorer shows the blocking graphs including segment 60642. As can be seen this segment has seen a blocking due to many different reasons. This is likely because of some area based blocking rule impacting segment 60642.

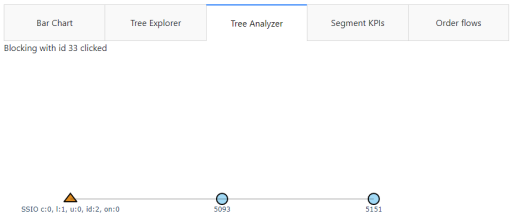
mance Indicators. This view is reached by pressing a button in the right panel, as seen in figure 5.5a. One of its features is to display KPIs about a specific segment (5093 in this case), like for example how often blockings occur when AGVs traverse the segment, or how long the duration of the average blocking is. In the bottom-most row, there are a list of *Blocking IDs*. These correspond to different instances of a blocking happening at the selected segment. Upon pressing one of the values (for example 33, as highlighted), the view changes to the *tree analyzer*, as seen in figure 5.5b. This view is not to be confused with the tree explorer, since it instead contains only one instance of a blocking graph (whereas the tree explorer contains *all*), namely the one that corresponds to the clicked blocking ID. While in the tree analyzer view, it is possible to hover on nodes to obtain more detailed information, as shown in figure 5.5c.

5.1.2 Traffic Dynamics of Dataset

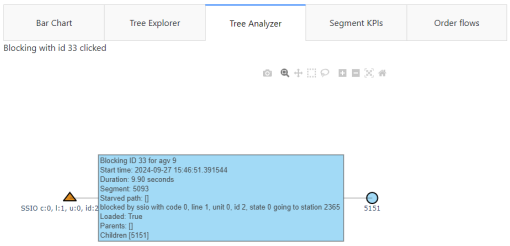
With the visualization tool presented, we now move on to present the empirical traffic dynamics of the systems in the dataset. In other words the wish is to present the data as is, to gain a better understanding what is actually happening in these systems. To accomplish this, the visualization tool will be used as help, along with a histogram of the blocking depths.
In Figure 5.6 the metric *segment count* from the previous section is displayed for each system. To repeat, segments are colored on a scale from red to orange to



(a) Segment 5093 KPIs



(b) Blocking graph with id 33



(c) Hover information

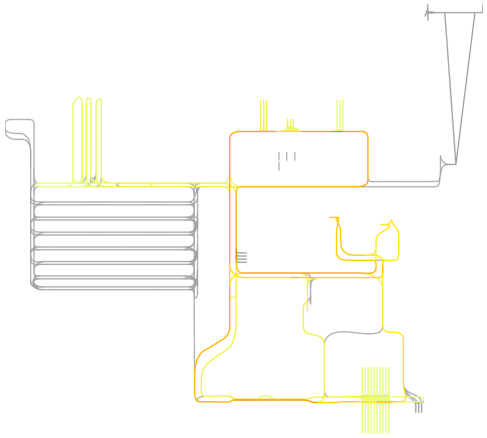
Figure 5.5: Blocking graph analyzer functionality.

yellow, with larger metric values corresponding to red and smaller ones to yellow. The gray areas of the layout mean that the metric has no value for these segments. So, for the case of segment count, the figure shows the most visited parts of the layout as red/orange. Across all the systems, there is a main stretch of segments that have the majority of traffic. These segments are mainly consisting of the loop at the top and the stretch of road starting from the bottom heading left and then turning upwards. The systems do not produce identical traffic distributions, though, since there are some small perturbations in order flow and topology (mentioned in Section 3). These differences produce deviating behavior in the center-right area of the systems.

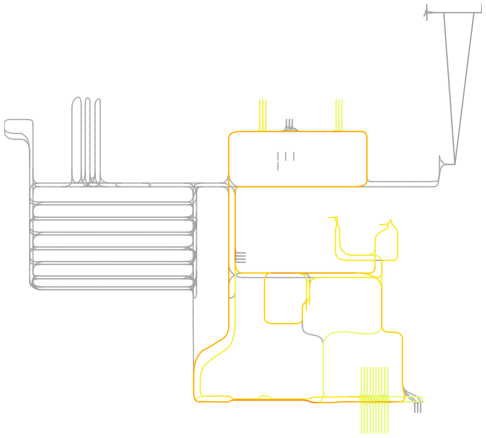
Figure 5.7d contains a scenario much similar to the one discussed in the previous paragraph, but now with the metric *blocking time* instead of segment count. The rest works in the same manner, regarding colors and such. As can be observed by all the gray segments, the blockings are quite sparse in every system - a symptom of good layout design. In all four systems, some areas do actually have blockings. A subset of these areas are common among the systems, while others are unique to a specific system. For example, both the Base system and Separated system have significant blockings in the top and center-right part. The separated system and overlapping system have in common that the most severe area is in the lower center. The one that stands out the most, however, is the congestion system. Here blockings are only significant to the top intersection.

Lastly, Figure 5.8 presents histograms for each system, illustrating the distribution of blocking depths (in the blocking graph context). The x-axis represents various blocking depths, while the y-axis shows frequency bars. The frequency is normalized by the maximum value to facilitate comparison between different systems. The base

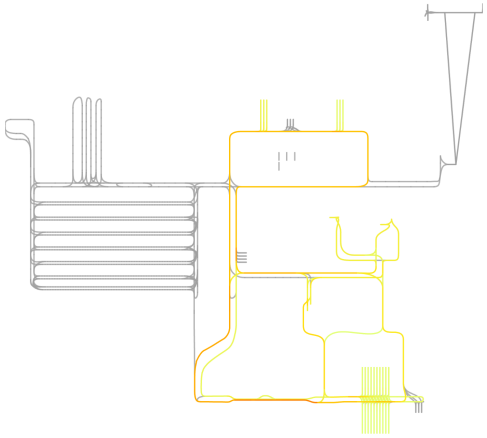
system and the separated system appear nearly identical. The overlapping system shows more two-depth blockings and even some three-depth blockings. Similar to the previous observations, the congestion system stands out again. Although most blockings occur at depth one (as in the other systems), there are also blockings evenly distributed between depths two and eight, with a notable spike at depth five and a minor occurrence at depth nine.



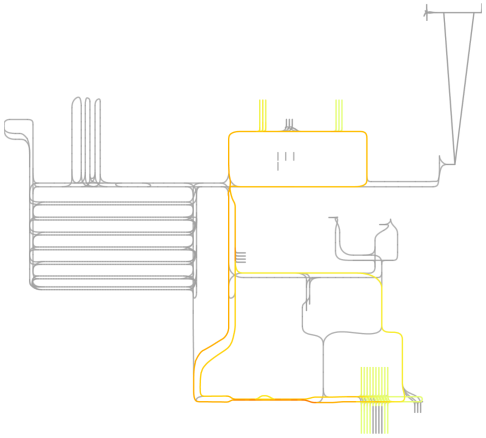
(a) Base system



(b) Separated system

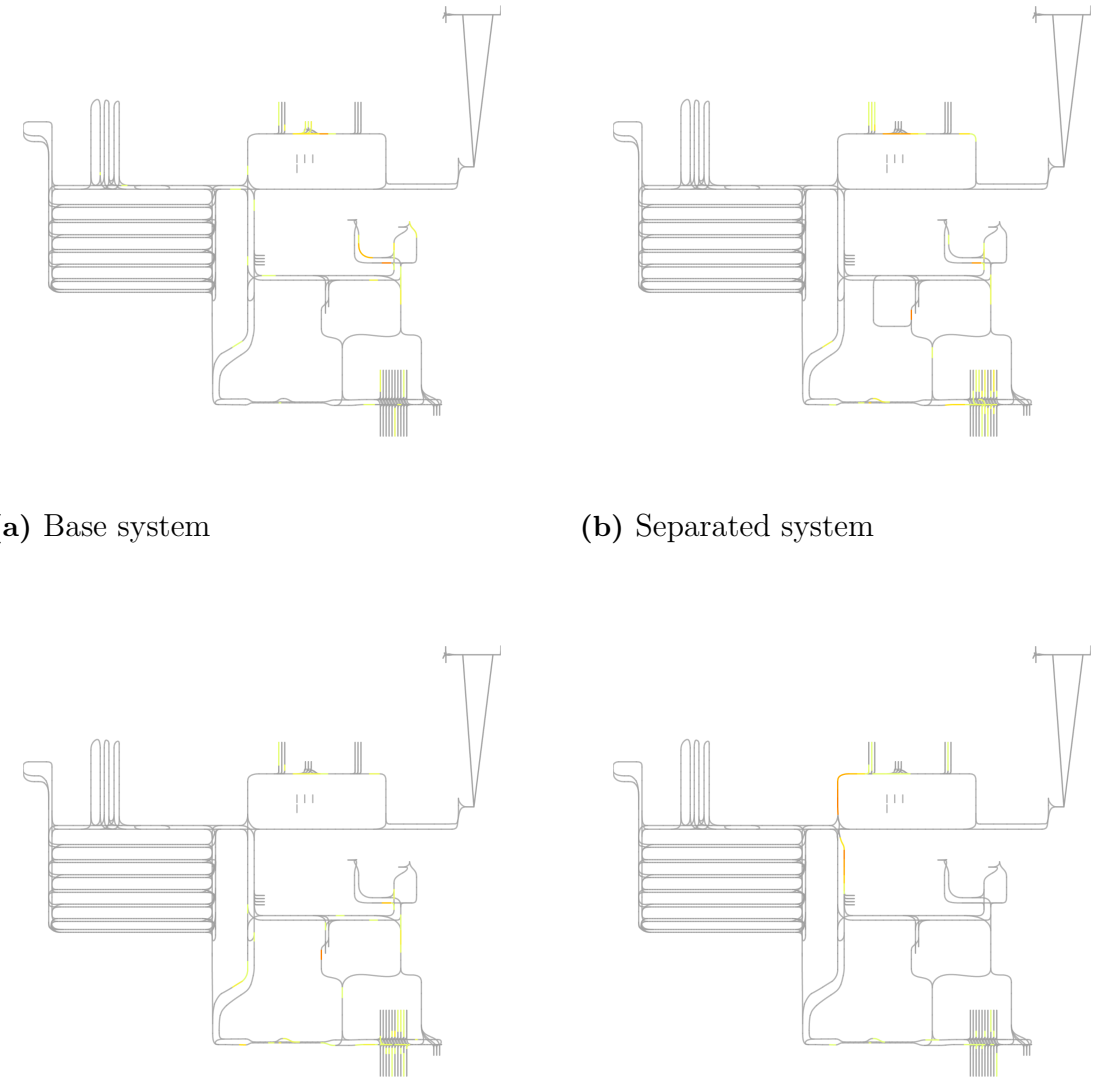


(c) Overlapping system



(d) Congestion system

Figure 5.6: Segment count for the four systems. Main takeaway is that the traffic distribution is quite similar across all systems. A notable difference is that the traffic load varies in the center-right parts.



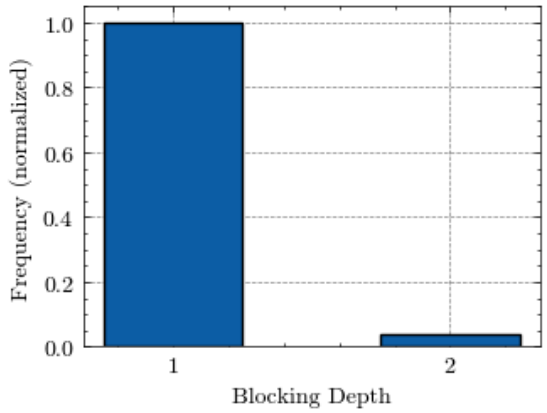
(a) Base system

(b) Separated system

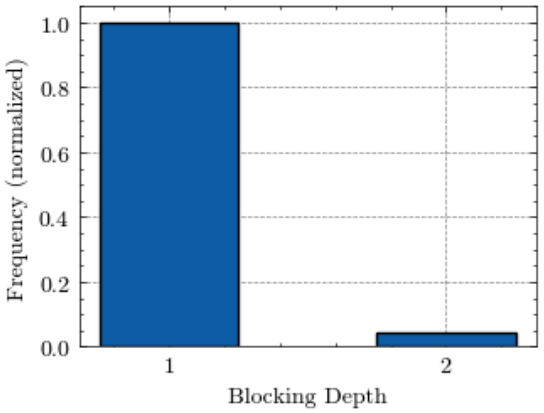
(c) Overlapping system

(d) Congestion system

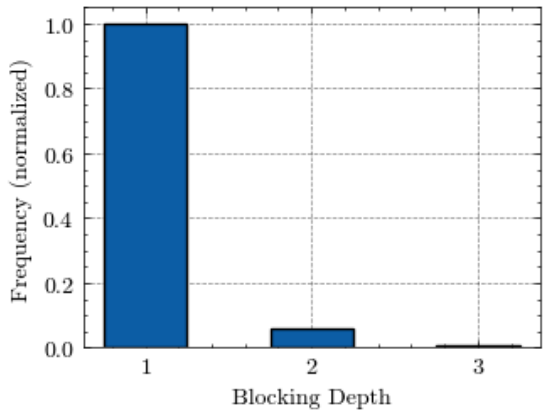
Figure 5.7: Blocking time per segment for the four systems. Key insights are that some systems share major blocking areas, such as the base system, separated system and overlapping system. What stands out is the congestion system that has severe blockings close to the intersection at the top. Apart from this, the blockings are quite sparse across the board.



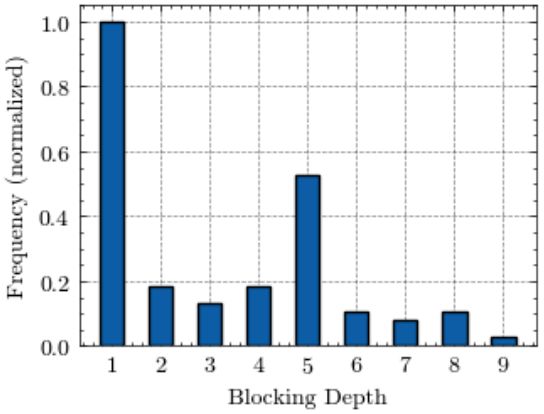
(a) Base system



(b) Separated system



(c) Overlapping system



(d) Congestion system

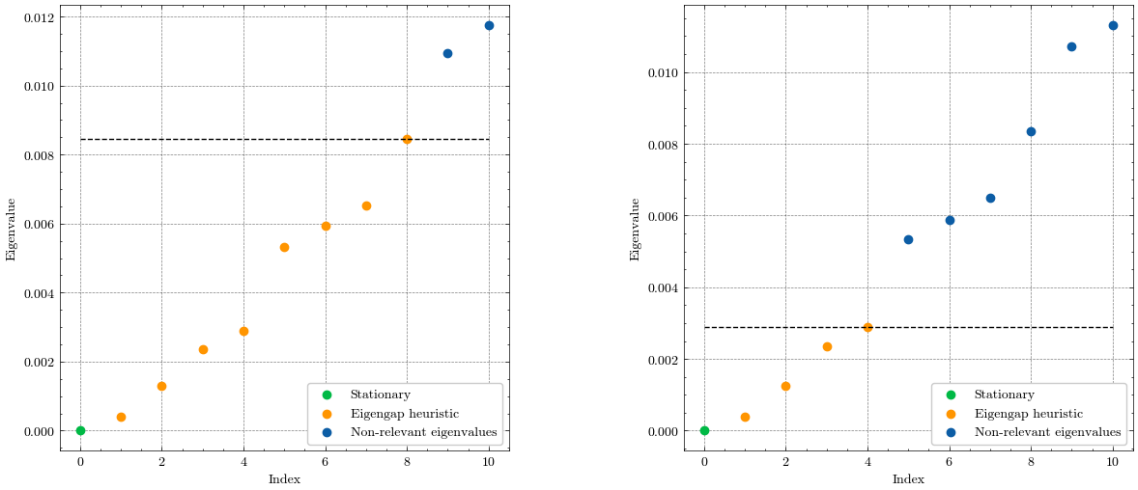
Figure 5.8: Blocking depth histograms for the four systems. The base system and separated system look much the same, whereas the overlapping system has slightly deeper blockings. Finally, the congestion system has much deeper blockings happening, even at a depth of nine sometimes.

5.2 Macro-level Modeling

This section aims to provide results related to the second research question, that is to find the most significant areas in the layout.
The structure of the section is to first give some examples of the insights that can be gained when analyzing the Spectral Embedding, and then to present some results for each system in the dataset.

5.2.1 Spectral Embedding

After using the eigengap heuristic to determine the number of eigenvectors k, some insights can be gained even before performing the clustering part of algorithm 2. These are insights that we say are related to the *spectral embedding*.
In figure 5.9, two figures are portraying the eigenspectrum and the results of the eigengap heuristic for the Separated System. In figure 5.9a the Topology model results are shown, with the result of eigengap giving nine clusters, whereas in figure 5.9b the eigengap results for the Blocking model yields five clusters. Thereby, simply adding a model of system behavior can actually impact the results quite a lot, as will be seen more in detail later.

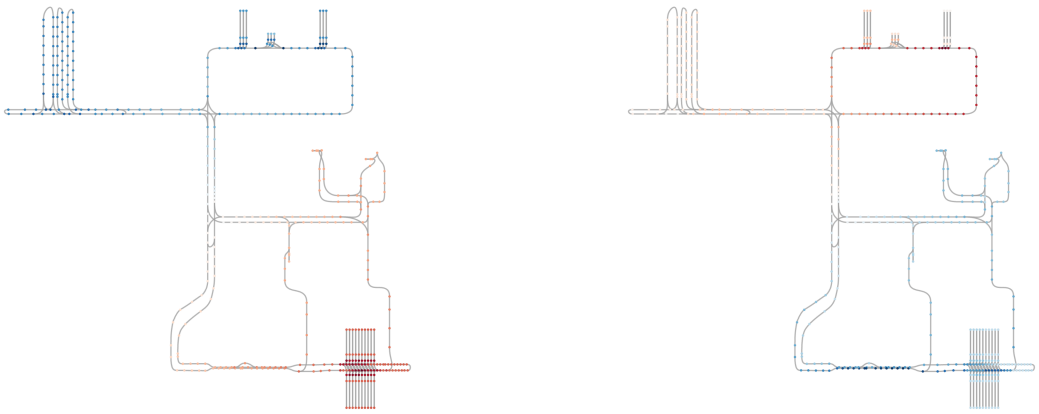


(a) Topology **(b)** Blocking

Figure 5.9: Eigenspectrum with eigengap results for the Separated System, and two different models. For the eigengap heuristic k_{max} is equal to ten and the two models used are topology and blocking, as stated. Main takeaway is how much impact the modeling can have on the number of clusters picked.

As the reader will recall, the first eigenvector is called the Fiedler vector, and it represents a binary split of the layout. This split can be interpreted as the *dominant behavior* for a certain model, and therefore it can be interesting to visualize this split. Perhaps if the model of a system behavior deviates a lot from the topology model, this indicates that the weights have non-uniform distribution, or vice-versa if they are similar.

In figure 5.10 two versions of the Base System layout can be seen, with the nodes being colored according to the coefficients of the respective Fiedler Vector for the two models. In the left figure (5.10a), the Topology model is used. The split is analogous to the sparsest cut (topologically, read more in 2.2), and seems to be on the highway in the center of the layout, with the two negative and positive extremes showing a dipole structure similar to a magnet. For the Traffic-flow model, seen in figure 5.10b to the right, the partitioning is quite different. Although the cut happens in a similar spot (highway in the center), the two negative and positive extremes show a different emphasis. Still upholding the dipole structure, but with two high-traffic stretches of road accounting for the largest magnitude coefficients.



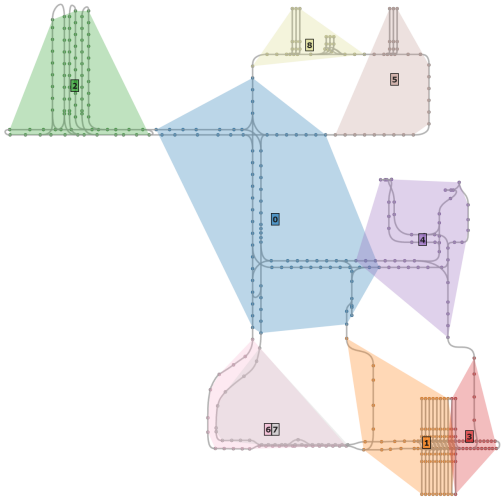
(a) Topology **(b)** Traffic-flow

Figure 5.10: Fiedler vector plots for the base system, two different models. Nodes are colored based the magnitude of the corresponding entry in the Fiedler vector. Left panel shows the topology model and the right panel shows the traffic-flow model. Showcases how models can impact the sparsest cut location and largest magnitude coefficients, i.e. *dominant behavior*.

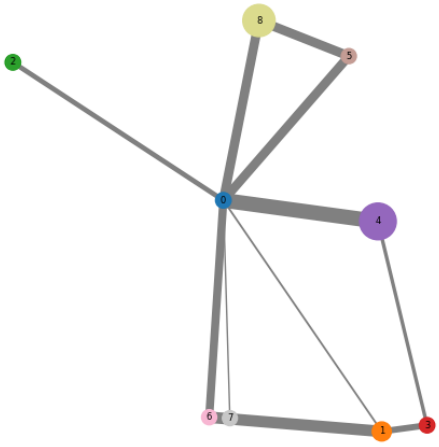
5.2.2 Base System

In this subsection, some hand-picked results will be presented for the *Base System*. We remind the reader that this is the system that is the most alike a real AGV system.
In figure 5.11 the spectral clustering results for the base system is seen. The model used is topology. In the upper-center, figure 5.11a, the resulting partitioning of the layout is shown (physical view), and combined with the abstract view (5.11b) and the cluster metrics (5.11c) below, some interesting insights can be gained. First off, it can be seen in the abstract view that the largest traffic-flow is between cluster 0 (blue) and cluster 4 (purple), and with much less traffic-flow continuing from cluster 4 to cluster 3 (red), this indicates that large amounts of the traffic flows in to cluster 4 via cluster 0, and back again. Secondly, the cluster metric *traffic density* in the

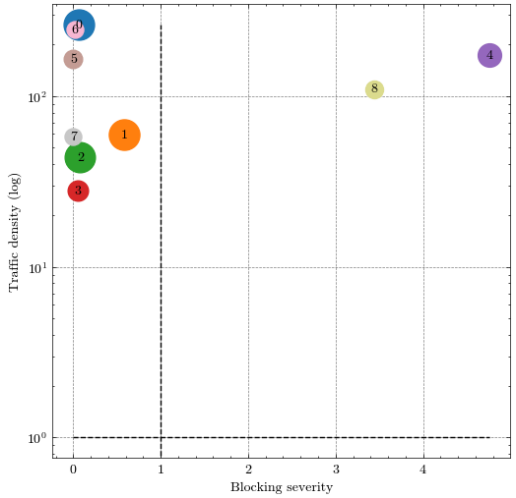
cluster metric plot shows that cluster 0, cluster 4, cluster 5 (brown) and cluster 6 (pink) are the areas with the most amount of traffic inside. Cluster 8 (beige) is not far behind. Thirdly, it can be seen in the cluster metric view that the areas corresponding to cluster 4 and cluster 8 are the most severe in terms of *blocking severity*. A similar structure of the figures is reoccurring the consequent results, so the reader is urged to try and understand the components of figure 5.11.



(a) Physical view



(b) Abstract view

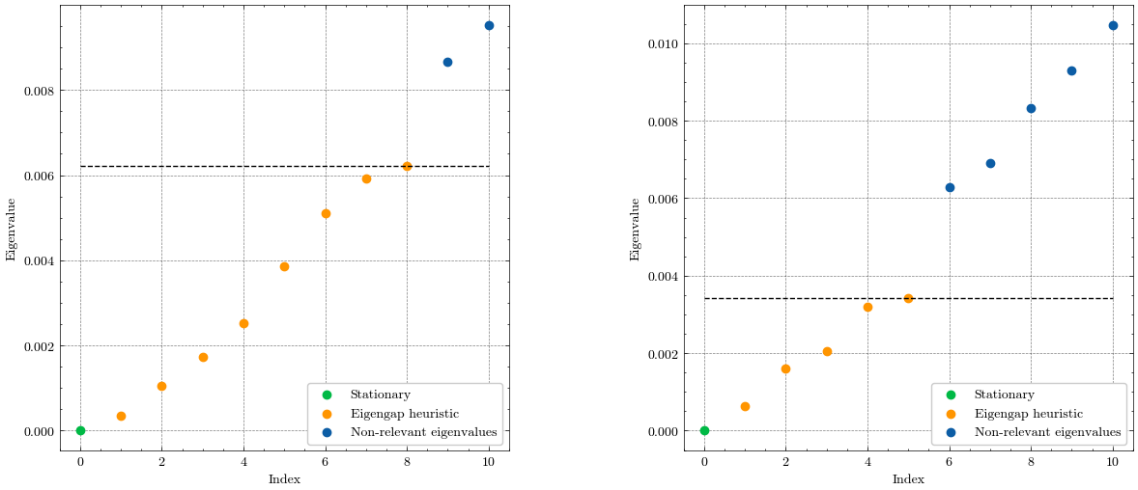


(c) Cluster metrics

Figure 5.11: Topology spectral clustering results for the base system. Eigengap heuristic was ran with k_{max} equal to ten and number of clusters chosen is nine.

As previously mentioned in subsection 5.2.1, models can have a major impact on the eigengap heuristic algorithm results. In figure 5.12 this behavior is seen once again. The eigengap results for two different models (topology and congestion) are

shown in the left panel (figure 5.12a) and right panel (figure 5.12b). It can clearly be seen that the congestion model was quite impactful, since the number of clusters yielded by the algorithm were six instead of nine.



(a) Topology **(b)** Congestion

Figure 5.12: Figure showing two eigengaps from the base system. Comparison shows how a model (congestion) can drastically change the eigengap compared to topology. The k_{max} used is ten for both plots.

Continuing with the same Congestion model as described in the previous paragraph, we now show the corresponding Spectral Clustering results in figure 5.13. Notice how different the clustering is here (compared to 5.11a), both due to fewer amount of clusters but also due to the model weights.
Finally, there is a behavior we wish to highlight with the *traffic-flow* model. In figure 5.14 there can be seen a green cluster which exhibits some unusual behavior. Namely, it spans across two remote areas in the layout. This behavior will be revisited later in the report.

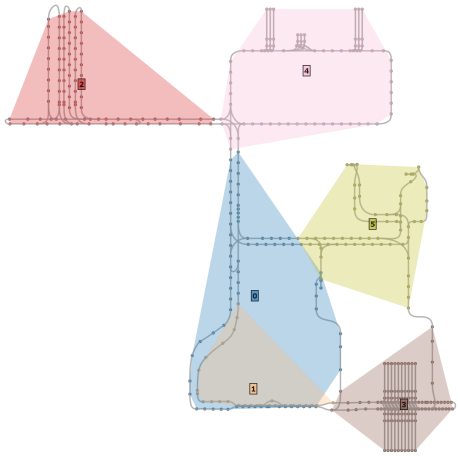


Figure 5.13: Congestion model physical view for base system. Notice that there are fewer clusters here than for the topology model (even though k_{max} still is ten).

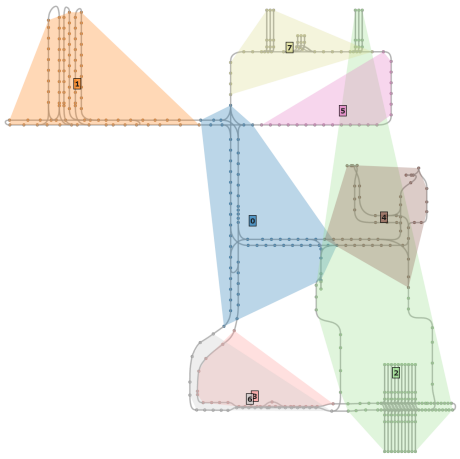
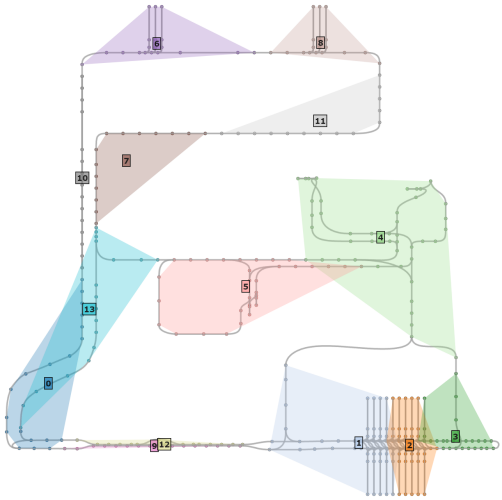


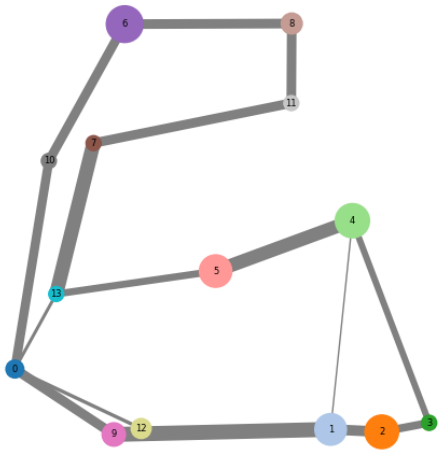
Figure 5.14: Traffic-flow model physical view for the base system. The k_{max} parameter used is 10 and the number of clusters selected by the eigengap heuristic is 8. Notice the green cluster that spans across two remote areas of the layout.

5.2.3 Separated System

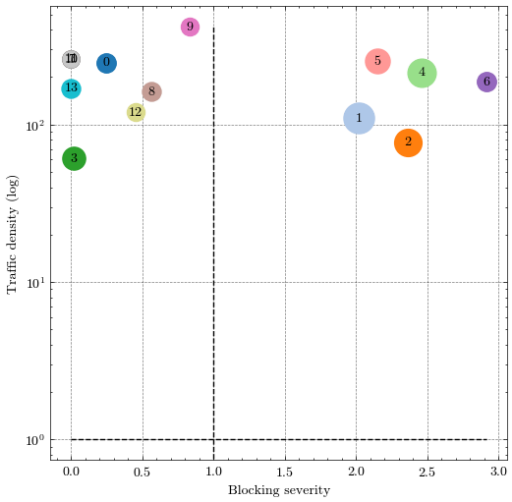
Now, the spectral clustering results are shown for the *separated system*. This system is supposed to have more separation (traffic-flow-wise) than the base system.
The topology model results are shown in figure 5.15. It can be seen in the abstract View how traffic is quite well distributed in two separate areas, one on the left and on the right side, with a weak connection (traffic-wise) between cluster 0 (darkblue) and cluster 13 (lightblue). The areas are not really separated though, since they are connected at the top, via cluster 6 (purple), but here there are a lot of blockings, as indicated by the size of the node in the Abstract View, as well as in the cluster metric plot on the x-axis. Other clusters with high blocking severity include cluster 1 (gray blue), cluster 2 (orange), cluster 4 (lightgreen) and cluster 5 (salmon). Compared to the base system, this system has more evenly distributed traffic-flow, as indicated by the more narrow distribution in the y-plane (traffic density).
In figure 5.16 the results for the *blocking model* can be observed. While looking at the cluster metrics, it is interesting to note that fewer clusters have a high Bblocking severity compared to the topology results (figure 5.15). Not only are fewer clusters deemed as significant (four instead of five), but also, they are more bunched together. Perhaps the blocking model can clump together dense-blocking areas into single clusters? More on this later. Finally, note that the y-axis distribution (traffic-flow) is virtually the same.
Now, the congestion model results. In figure 5.17 the main result that can be seen is that fewer clusters are chosen to be significant in terms of blocking severity when comparing to the topology model in 5.15. Additionally, cluster 2 (beige) stands out as the most significant blocking-wise - this differs from the blocking model (figure 5.16), in which four different clusters had roughly the same, high, blocking severity. Including the traffic-flow in the model, seems to spread out the clusters more on the y-axis (in figure 5.17b), see for example the distinction between cluster 3 (green) and cluster 10 (lightblue).



(a) Physical view

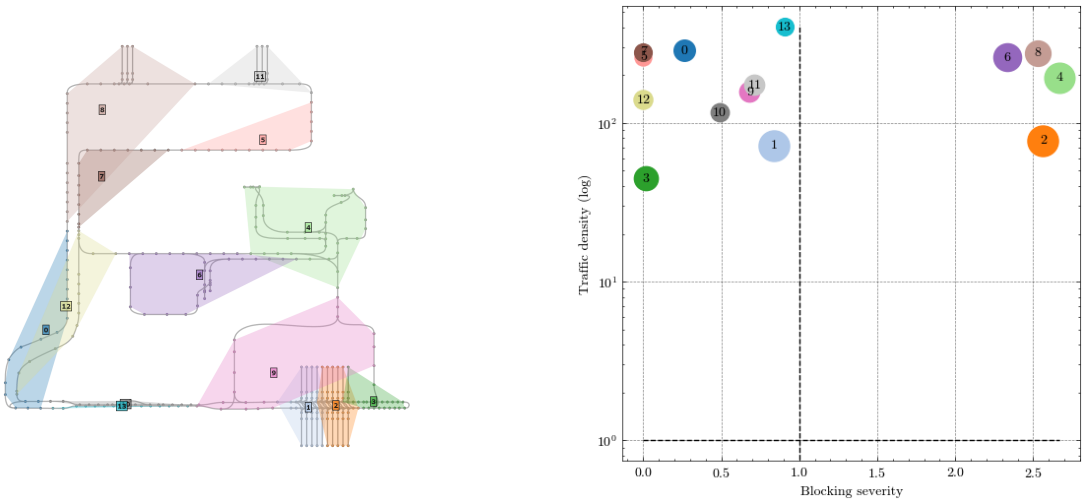


(b) Abstract view



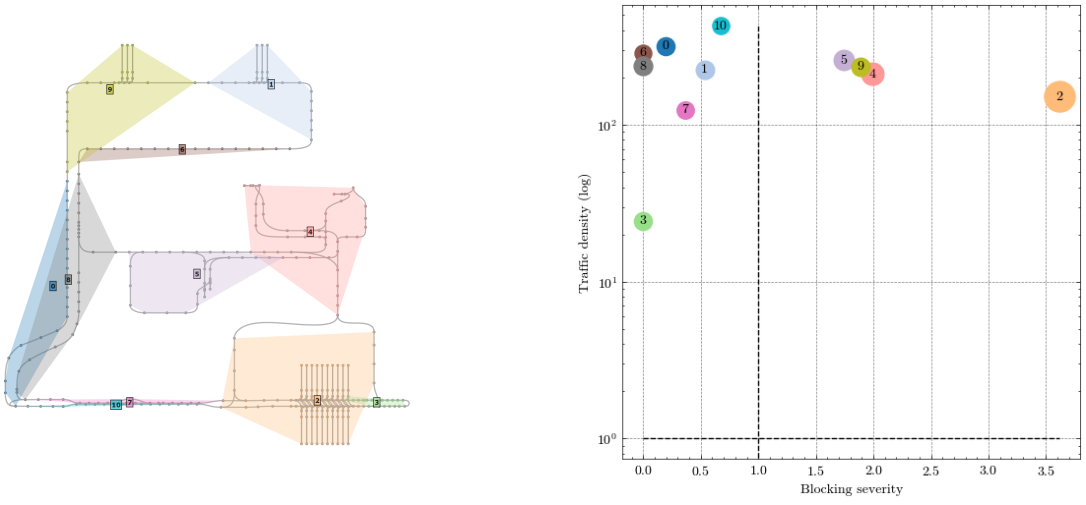
(c) Cluster metrics

Figure 5.15: Topology spectral clustering results for the separated system. Eigengap heuristic was ran with k_{max} equal to 15 and number of clusters chosen is 14.



(a) Physical view **(b)** Cluster metrics

Figure 5.16: Blocking model spectral clustering results for the separated system. Physical view in the left panel and cluster metrics in the right panel. Eigengap heuristic was ran with k_{max} equal to 15 and the number of clusters picked is 14. Notice how the blocking model clumps together more densely blocked areas into single clusters.

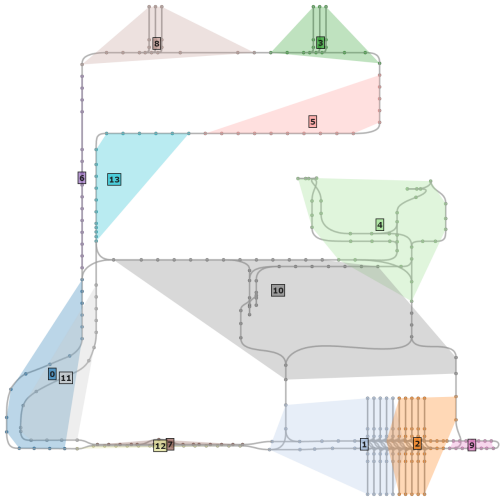


(a) Physical View **(b)** Cluster metrics

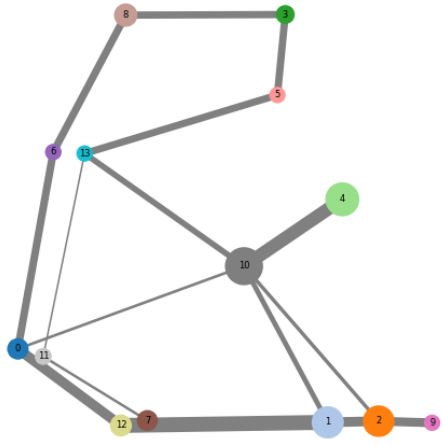
Figure 5.17: Congestion model spectral clustering results for the separated system. Physical View in the left panel and cluster metrics in the right one. For the eigengap heuristic, k_{max} equal to 15 is used, and this yields 11 clusters. In this figure, it is seen how the congestion model spreads out the clusters on both axes.

5.2.4 Overlapping System

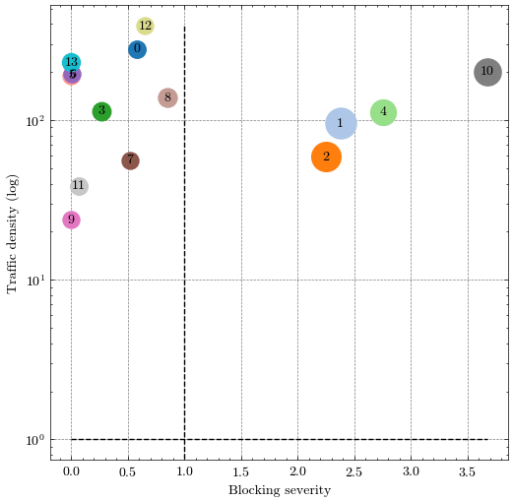
The *overlapping system* is similar to the separated system, but the traffic is directed slightly differently, as to produce a different type of flow. Additionally, its layout is permuted a bit, contributing even further to different traffic dynamics. In this section, the spectral clustering results will be shown for this system.
Now, in figure 5.18, the topology model spectral clustering results can be seen. In the abstract view, the main traffic-flow is observed between cluster 10 (darkgrey) and cluster 4 (lightgreen), as well as along the bottom part of the layout (cluster 2, cluster 1, cluster 12 and cluster 0). This flow is quite interesting, as it seems like part of the traffic is contained by the first mentioned clusters (10 and 4), whereas virtually the rest is in the latter mentioned clusters (2, 1, 12, and 0). Regarding blockings, the clusters with highest Blocking Severity are seen in the cluster metrics plot. Among them cluster 10 stands out as the clear leader, with cluster 4, 1 and 2 close behind. Cluster 12 and 0 have the most traffic density.
Now, the spectral clustering results for the blocking model, see figure 5.19. Apart from the ordering of the cluster numbers, these results are basically identical to the results for the topology model (5.18). This is interesting, since it means that the Blocking model does not impact the clustering in this case. More on this later.
In figure 5.20, we observe the results for the traffic-flow model. It is interesting to see that only two clusters, cluster 3 (purple) and cluster 2 (green), are distinguished as having a blocking severity greater than one, with cluster 3 being the most severe. Additionally, cluster 4 (brown) and cluster 0 (blue) have the most traffic density. This coincides well with the expected system behavior described in chapter 3. Interestingly enough, the traffic-flow model ends up with much fewer clusters than the previously two mentioned models (topology and blocking), only having seven clusters.



(a) Physical view

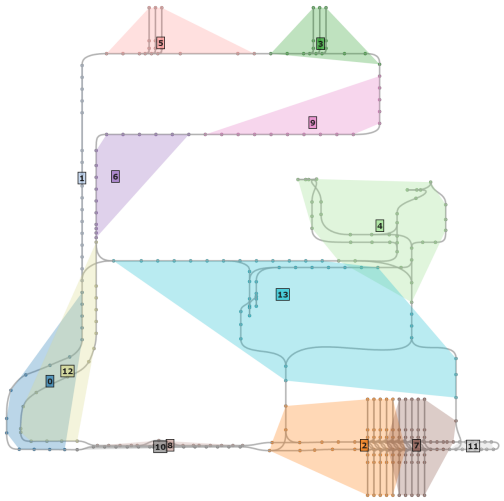


(b) Abstract view

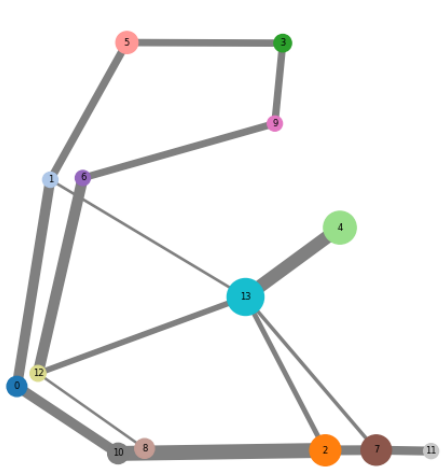


(c) Cluster metrics

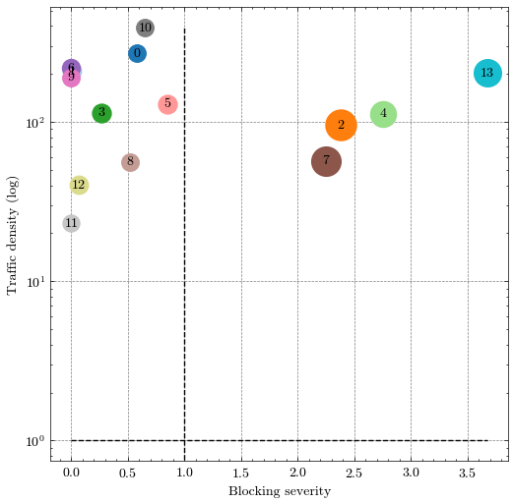
Figure 5.18: Topology model spectral clustering results for overlapping system. k_{max} equal to 15 is used for the eigengap heuristic, which resulted in 14 clusters. The main finding is that the clusters 10, 4, 1, 2 have the most blocking severity, and the clusters 12, 0, 13, 10 have the most traffic density. Therefore cluster 10 (darkgrey) has both a lot of blockings and traffic.



(a) Physical view

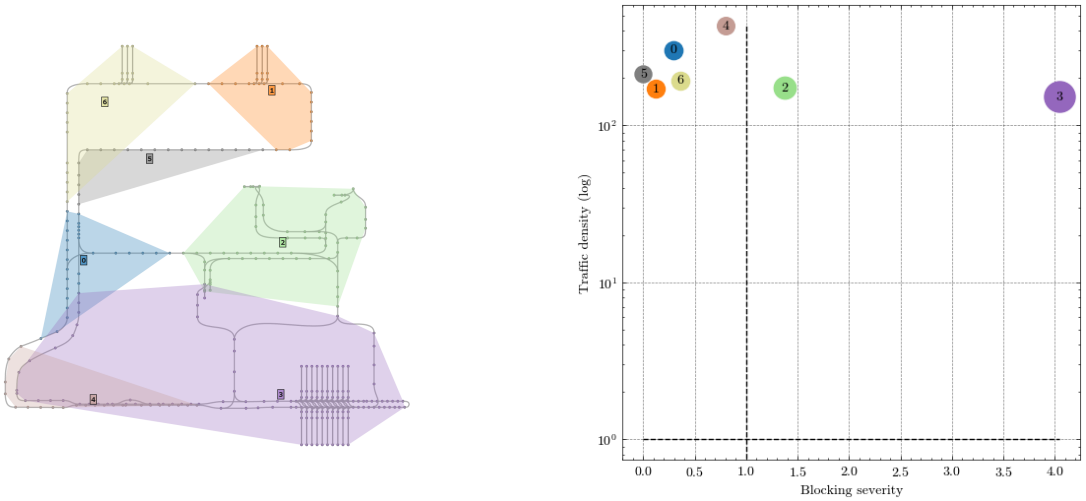


(b) Abstract view



(c) Cluster metrics

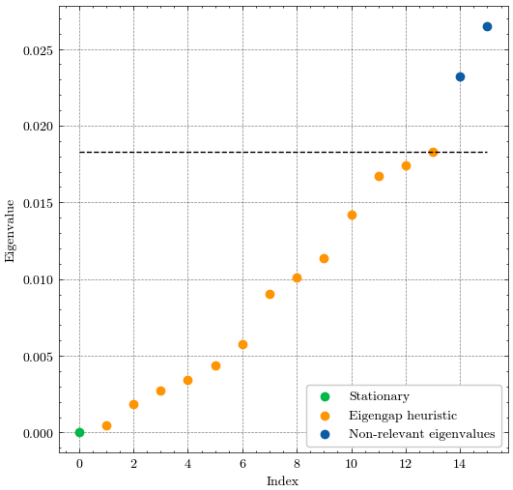
Figure 5.19: Spectral clustering results for the Blocking model, overlapping system. Eigengap heuristic is ran with k_{max} equal to 15 and number of clusters are 14. The main finding is that the blocking model did not affect the clustering results (compared to baseline, topology model).



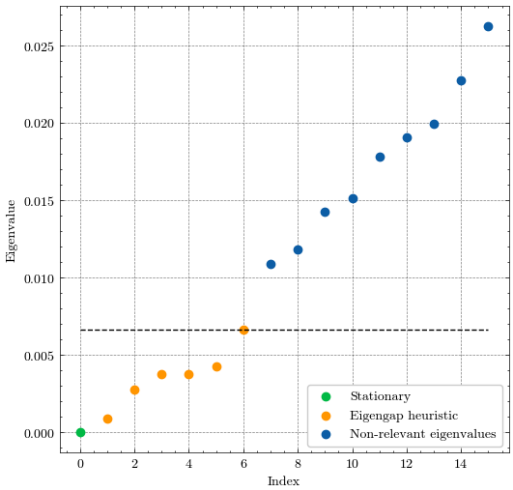
(a) Physical view **(b)** Cluster metrics

Figure 5.20: Spectral Ccustering results of traffic-flow model, overlapping system. Eigengap heuristic runs with $k_{max} = 15$ and number of clusters is seven. The main finding is that this model managed to distinguish just one or two clusters as problematic.

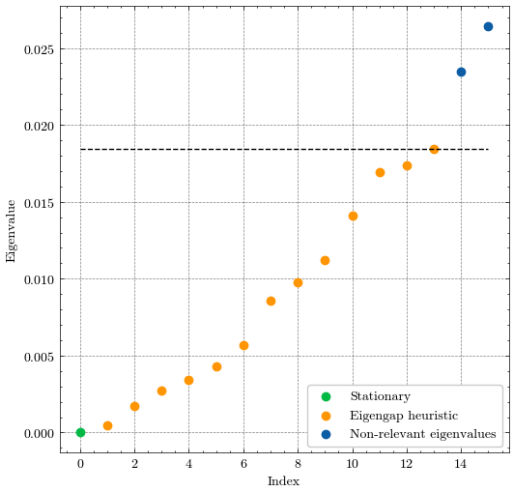
To support the earlier described findings, we now provide a plot of the eigenspectrum (with the eigengap heuristic) for the models described earlier in this section (see figure 5.21). Notice how using the blocking model did virtually no difference to the eigenspectrum, whereas the traffic-flow model slightly changed the values so that the eigengap heuristic resulted in much fewer clusters. This is a perfect example of how this plot can be used to verify model impact.



(a) Topology model



(b) Traffic-flow model

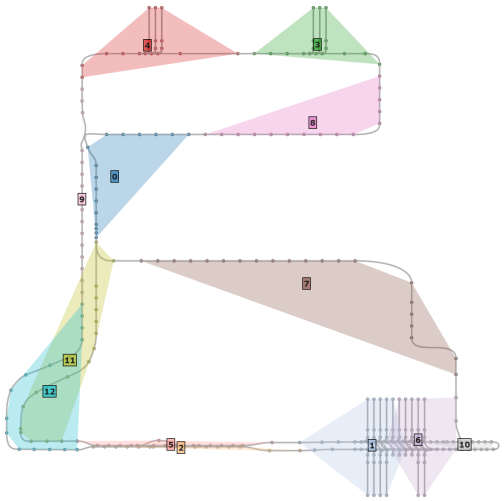


(c) Blocking model

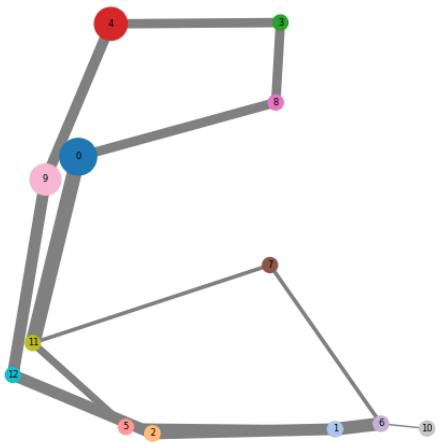
Figure 5.21: Eigenspectrum with eigengap heuristic result for different models. Upper panel shows the topology model and the lower panel shows the traffic-flow mode on the left and the blocking model on the right. Main finding is that the traffic-flow model produced a much smaller k than the two other models.

5.2.5 Congestion System

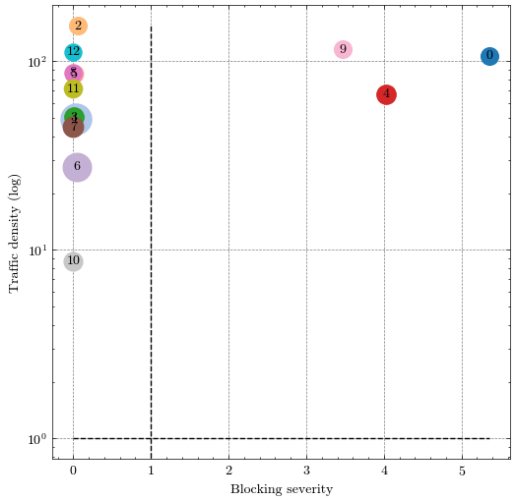
The *congestion system* is designed to have traffic queues forming in a specific intersection. Now, some spectral clustering results of this system will be shown.
For the topology model, the results are shown in figure 5.22. In the abstract view, traffic flows in a clear loop, with little-to-none traffic through cluster 7 (brown) and cluster 10 (gray). Using the cluster metrics, we can see that cluster 0 (darkblue), 4 (red) and 9 (lightpink) have the highest blocking severity, with the rest of the clusters having virtually zero blockings in comparison. Traffic-density-wise, cluster 2 (beige) is the highest.
In figure 5.23, the reader can observe the delay model spectral clustering results. Interestingly enough, there are now only two clusters which are pointed out as significant, rather than three (as in the topology model). These ones are cluster 3 (green) and 0 (darkblue). This model finds two tiny clusters, 7 (lightbrown) and 11 (lightblue). Finally, similar to the previous model, the areas corresponding to cluster 6 (darkbrown) and 9 (gray) have the most traffic density.
Similar to the congestion model, the blocking model (in figure 5.24) manages to distinguish two instead of three clusters as significant. In the cluster view, these can be seen: cluster 0 (darkblue) and 4 (lightgreen). Clusters 1 (gray blue), 2 (orange) och 11 (lightgrey) have the most traffic density.
Figure 5.25 contains spectral clustering results for the traffic-flow model. It is interesting that cluster 7 (darkbrown) spans across three to four disjunct areas of the layout.



(a) Physical view

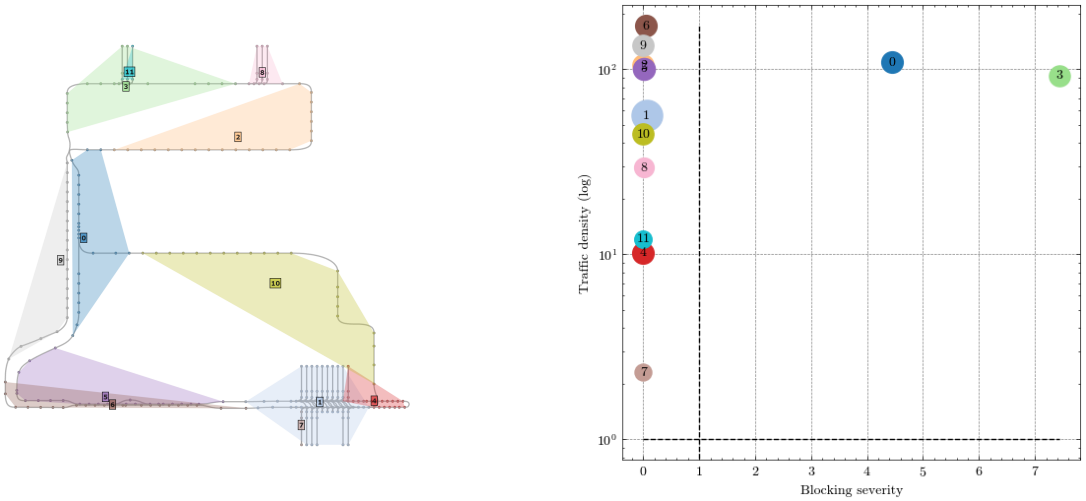


(b) Abstract view



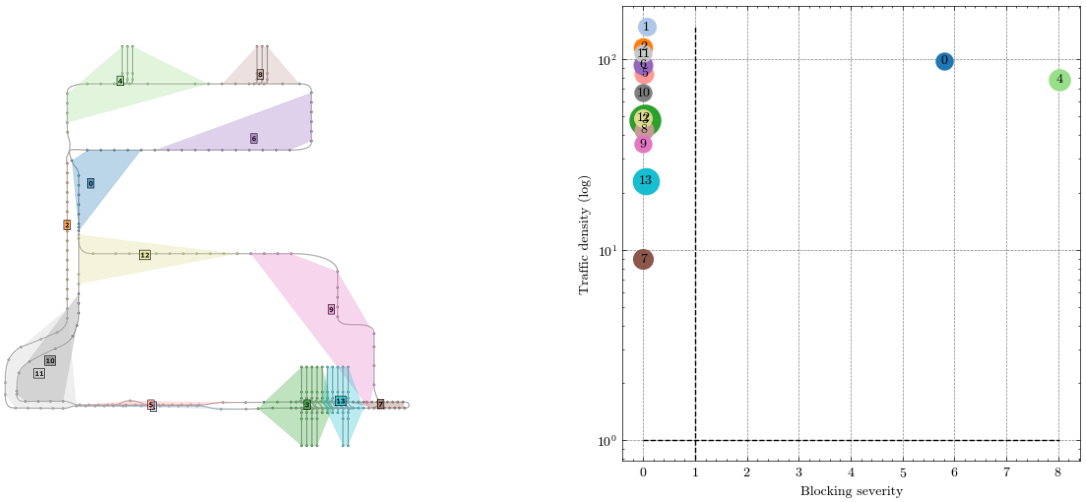
(c) Cluster metrics

Figure 5.22: Topology model spectral clustering results for the congestion system. When running the eigengap heuristic, $k_{max} = 15$ yielded 13 clusters. What to note here, is that the areas corresponding to clusters 0, 4 and 9 have the highest blocking severity, whereas clusters 2, 9 and 12 have the highest traffic density.



(a) Physical view **(b)** Cluster metrics

Figure 5.23: Spectral clustering results for the delay model, congestion system. Eigengap heuristic uses $k_{max} = 15$, which results in 12 clusters. The two key findings are that the delay model finds two very small clusters in the station areas, as well as two significant clusters instead of three (compared to baseline topology model).



(a) Physical view **(b)** Cluster metrics

Figure 5.24: Blocking model spectral clustering results for the congestion system. From the eigengap heuristic (with $k_{max} = 15$), resulting in 14 clusters. Only two clusters are pointed out as significant (cluster 0 and 4).

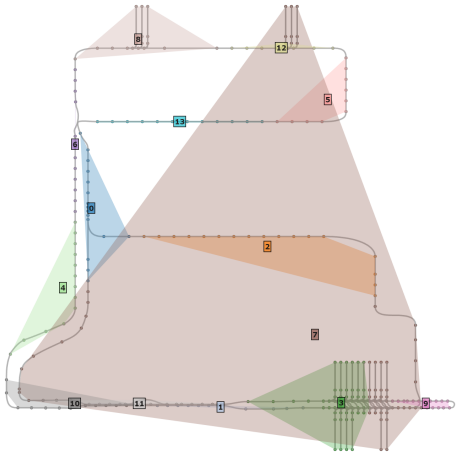


Figure 5.25: Traffic-flow model spectral clustering results, congestion system. Eigengap heuristic was ran with $k_{max} = 15$ and the number of clusters resulted in $k = 14$. Main finding is that the darkbrown cluster (number 7) spans across multiple disjunct areas of the layout.

6
Discussion

6.1 Micro-level Descriptive Analytics

6.1.1 Blocking graphs

The biggest novel contribution of the micro level analysis is the blocking graphs. These model the cause and repercussions of all blockings in the system. The data structure is iterative which enables easy traversal both up and down a graph. Extracting the data and connecting blockings that interact with each other in this way is what enables the more in depth congestion analysis tools to function. After parsing this data structure therefor provides a comprehensive look into how the system behaved from a congestion point of view. This directly helps answer the first research question. By looking at the blocking reasons for graphs it is directly clear what was the initial cause. By iterating to a objects parent until the root is reached a user then is able to see the root cause. This enables clear analysis and provides data regarding which causes are the most significant too.

6.1.2 Metrics

The most interesting traffic dynamic metrics, relating to an individual segment, currently provided are:

- Segment count - Number of traversals per segment.
- Blocking time - Duration of all blockings on segment.
- Occurrence rate - The percentage of traversals on segment that result in blockings.
- Time per blocking - Average time per blocking.
- Blocking depth - Depth of blockings.
- Total waste - Waste of all blockings on segment.

Segment count provides a very good insight into how vehicles moved in the systems. It answers the question of which areas see the most AGV traffic. This is useful to get an understanding for how the system behaves and to see which segments are utilized the most. To get insights into the congestion we have to look into the blocking graphs. Blocking time is the accumulated time of all blockings on a segment. By evaluating segments with this metric it is possible to sort and find which segments are the most problematic in terms of blocking duration. Occurrence rate is calculated per segment as the rate of blocking per traversal. As a quick example a segment that has been traversed ten times and has seen a blocking three times will have an occurrence rate of 30%. This can be seen as the probability of an AGV

getting blocked while traversing the segment. Segments with a high occurrence rate are likely to get blocked which can lead to problematic behaviors if a lot of vehicles are routed via it. Time per blocking is a metric that simply displays the average duration of blockings on a segment. If the number of blockings on a segment are many but the time per blocking is low it may not be problematic. To get a sense for how often a blocking graph propagates backwards there is a metric for seeing the blocking depth per segment. The blocking depth of one blocking graph is determined as the maximum number of propagation layers of the graph. A blocking graph that is just one blocking has a depth of one. If there exists a child to the object and the child in turn has just one child the depth of the graph is three etc. By using this as a segments metric we can locate which segments have propagating blocking graphs. If the depth is high it means that the initial blocking often causes long chains of blockings to occur. Finally there is a metric to display the total "waste" per blocking. This metric is implemented as the duration of wasteful blockings i.e. total time wasted by blockings that hurt the behavior of the system. Since the amount of time was hard to define (as stated in Section 2.1.4) it currently just categorizes blockings of loaded vehicles as more wasteful than those of vehicles without a load. This was one of the factors identified while investigating what constitutes a wasteful blocking but it is far from a complete view of the situation. Hence the waste metric should be taken with a grain of salt in its current form. These are the most important metrics available in the current data. More metrics can be extracted but the result of our findings is that this data is available. The available metrics gives a user different ways of looking at the system and with some exploration of these metrics it should be possible to determine which segments impact system performance the most.

Mission metrics

One important metric that is currently recorded but not visible in the application is the mission data for Automated Guided Vehicles (AGVs). Each mission, defined by a start and end station, is logged whenever an AGV is dispatched. Additionally, any blockings encountered along the route are associated with the respective mission. The AGV is programmed to travel along the shortest path between the start and end stations.
It could be beneficial to display the most frequently used stations and compare this data to the number of missions that encounter significant blockings. Analyzing this station data can provide layout designers with valuable insights into which stations are the most problematic. To further enhance the application's visualization capabilities, a shortest path algorithm could be integrated. This algorithm would determine the shortest path between station pairs and highlight this route in the physical layout view of the tool. This feature would allow designers to see the chosen route and easily identify blocked areas, facilitating specific blocking analysis for individual missions.
Grouping nearby stations into the same station group could provide a more aggregated view of mission behavior. Layouts often designate certain areas as loading or unloading bays, which contain multiple stations. Analyzing these areas collectively and identifying which ones are most frequently interlinked with missions could significantly enhance the spectral clustering overview of the layout. It would be important

that the station area grouping relies on topographical data rather than geographical data, as stations that are geographically close may be separated by long travel distances due to layout design and physical obstacles such as walls.

Enhanced metrics

The current tool features several important metrics that are collected and calculated. For a comprehensive analysis of the collected data, the metrics provided should be extensive, allowing users to investigate any data they desire. It may be beneficial to enable custom types of metrics, created through simple combinations of the provided statistics. Further testing is necessary to fully understand which metrics are the most important and useful for analysis. While we expect this to vary between layouts, some metrics should prove to be universally useful and should therefore be prioritized and displayed first to facilitate efficient analysis.

6.1.3 Visualization Tool

The current tool is however just a prototype. But with a multitude of visualization options it gives the user many ways of interpreting and understanding the data. We believe the ease of use is the most important factor of our app. It lets a user interpret and process the extracted data to gain actionable insights in a fast way. A wide section of available metrics enables a problem to be analyzed from different perspectives. It is especially useful that the graphs are interactive. By choosing a metric it is instantly displayed as a colored gradient across the layout as seen in figure 5.2. Additionally the bar chart is updated to display the segments with the highest value of the selected metric as seen on the right side. A segment identified as interesting in one graph leads to a deep dive into the segment metrics with just one click. By selecting a segment the bar chart tab is changed to display information about the segment. An example of this is seen in figure 5.5a. This gives a complete overview of the segments KPIs and let's a user understand the congestion behavior in full. The full list of blocking graphs including the chosen segment is also displayed allowing for easy maneuvering to the graph explorer. To visualize the blocking graphs a tree like structure was chosen as seen in figure 5.3. There are two different blocking graph views, the *Tree Explorer* and the *Tree Analyzer*. The Tree Explorer enables exploration of the blocking graphs from a segments point of view. It shows an aggregated graph were all graphs featuring the selected segment are included. By navigating to other segments it is possible to move to change the segment in focus. This tool gives a good understanding as to which segments impacts which as it aggregates blocking information from the graphs visually. The Tree Analyzer instead focuses on just one blocking graph at a time. It provides comprehensive information about each blocking object in the graph by simply hovering over each object, represented as a node in the graph. Both these tools are clearly just a prototype to show what data exists. None of them have been polished enough to provide a very intuitive user experience so this is something that has to be explored further in future work.We believe that this aspect of the visualization tool has significant potential for improvement through a UX-design makeover. While the current version effectively displays the most critical information related to the

graphs, enhancing its design could further highlight the insights derived from the analysis. This is essential for unlocking the full potential of the blocking graphs and to demonstrate how congestion in the system behaves.

6.2 Macro-level Spectral Clustering

6.2.1 Cluster Creation

Many spectral clustering models were tested with the system. Here we will discuss some of the most interesting results found regarding to the cluster creation and model decision. For a comprehensive collection of the results please see Appendix I.

Blocking time model

Looking at the propagating blocking system we can see that there are two areas with significant blockings. The unweighted topology model clusters the system in a way that splits both these areas in two. If we instead focus on the blocking time model we can see that it indeed keeps the intense areas inside of one cluster. This makes sense since adjacent segments with significant blocking times does create a strongly connected area in this model. From a blocking perspective it is very beneficial to keep these areas within the same cluster as they most likely are depending on each other. This makes sure the overview given by the clusters focuses its attention on areas with a lot of coherent blockings.
Looking at the areas were few blockings occurred we see that the clustering is very similar to the topology model in general. This is expected as no blockings leads to an unchanged adjacency matrix. Focusing on the cluster plots of the overlapping model we can see a clear example of this. Looking at the blocking overlay for this model shows that there are only a few blockings occurring and that they are spread out across the network. No intense areas of blockings means that the blocking time weights won't make a difference to the Laplacian connectedness of the graph. As seen, the cluster plots are almost identical both with or without taking blockings into account. This is expected behavior but it should be noted that there needs to be a significant amount of blockings for it to impact the clustering a noticeable amount.

Delay Propagation Model

When trying to apply Dekker's delay propagation model to the AGV domain we need to create a pseudo-timetable. Each new departure needs an initial delay per mission. Since no real timetable exists the base value was chosen to be zero seconds as it made the most intuitive sense for a vehicle not being delayed at the start of a new mission. However this creates some problems in the calculation of β. Since β relies on a division of the average initial delay we encounter a division by zero for each segment were no vehicle has been delayed. This will be the case for many segments, especially the ones directly after a home station. Segments where both the initial and resulting delay is zero on average can be handled by just defining this case to

give $\beta = 1$ i.e. delay doesn't change. But the bigger problems arise when looking at the first segment along the route were blockings are occurring. When dividing a real value with zero it is impossible to determine a result without making an arbitrary decision as to what the result should be. We decided to implement a maximum value of beta possible, and in the cases were a value greater than zero is divided by zero we simply assign this value instead. The problems with this is that a lot of information is lost. Two segments, both with previous delays of zero but differing delays after, will both get a beta value of maximum beta. This is true no matter the proportion between the segments resulting delays, which is a loss of information. So why was the initial delay chosen as zero if it has so many disadvantages? Well, no matter the initial value the problem will persist. If we decide the initial delay to be a very small value ϵ the division will end up creating a beta of enormous value. Here we again probably would want to limit the upper value of beta, which creates arbitrariness. This will also be true for other values of initial delay, they will be arbitrary. And another problem that consists with our pseudo-timetable is the fact that a mission delay is never shortened. Since the segment will never be run faster than specified there will never be cases were beta is below one. This means that the mission delay will only stay the same or increase over the mission. Such behavior means that a similar added delay for two following segments will have a higher β-value for the first segment. As an example, let's imagine a vehicle is entering segment 1 with a 1 second delay. The traversal of segment 1 leads to a delay of 9 extra seconds in this case. Calculating the beta for segment 1 we then have $\beta_1 = \frac{10}{1} = 10$. If the segment for the following segment 2 also catches 9 seconds of extra delay the beta will be $\beta_2 = \frac{19}{10} = 1.9$. As shown the beta values can vary a lot even for segments with similar added delay. Is this the parameter behavior we want to model? A symptom of this behavior is also that for a mission that accumulated a lot of delay early on, the following delays won't in comparison amount to a significant change in total delay and thus won't create significant betas. These problems aren't as apparent in a system with a timetable such as a railway system. With a provided timetable none of these arbitrary decisions are necessary. The data would only be gathered by how late a train is on arrival compared to the provided time in the timetable. Also, a train that is late can often make up for some time since a small amount of buffer time is often added to a timetable. Buffer time can be skipped when running late in some cases, resulting in a delay that is decreasing on average. Another problem that usually won't be the case in a real timetable controlled system is the division by zero. There exists no national railway system were each departure is always on time [39]. The departure and arrivals are at least a few minutes late on average. So no need to provide a maximum beta to, in some cases, avoid astronomically large values of beta which would skew the model.

We also see some problems with the parameter α. Since α was used to represent the proportion of delay distributed along each edge, it was defined based on the segment traversal frequency. In our case we don't have a total delay at station, we only measure changes in edge delays. So the delay propagation intuition is lost in our case, and what remains is just a percentage of traffic headed out per segment. Now the problem is that this percentage only is relative per station. Two segments with the same α value, but differing start nodes, can have wildly different actual

traversal frequencies, since the value only relies on proportion to the start node.

Eigengap heuristic

The eigengap heuristic gives a spectral suggestion of an appropriate value of k. This is useful as the problem of choosing a good value for k can often prove to be tricky in clustering models. While the heuristic is very helpful it however has some question marks regarding the application to this domain however. One problem is that the heuristic still needs a value for the maximum k. This leads to a arbitrary decision still having to be made. This means it isn't fully automated which can introduce human variance into the results. Another problem we have found when analyzing the results is that the behavior of the heuristic can be unstable. If we look at figure 5.9 it is seen that the eigenspectrum is very similar for both models. There exists two distinguishable gaps in both spectra. Due to minor differences in the models the topology model here picks the right gap while the blocking model picks the left gap. This isn't necessarily a problem, but it should be noted that while the models provided two almost identical eigenspectrums the chosen amount of clusters is very different resulting in major clustering differences. In general it should be noted that the eigengap heuristic can showcase some variance when the eigenspectrum includes two gaps of similar size. This is undesired behavior. Let's also look at a case were different models provide clear changes to the amount of clusters. Looking at figure 5.12 we can see two different gaps highlighted by the models. Since the eigenspectrum for both models looks completely different we can infer that there are significant differences to the model that the heuristic notices and as a result a different k is chosen. This is a situation were the different amount of clusters chosen for the two models makes a lot more sense.

AGV system data

One thing that can be seen by analyzing the data and looking at the weighted adjacency matrices created for the models is that the blocking data is often very sparsely spread out across the network. It is rare for many adjacent segments to all have blockings in them. And in the cases were this seems to be the case it instead is such a densely populated network that the blocking data is apparent in many areas, and just acts like a sort of noise spread across the adjacency matrix. Let's examine why both of these are problematic for the clustering algorithm. If the data is sparse most weights in the network will be unchanged, i.e. one across the network. This means that the impact the areas with blockings have on the clustering is fairly small, as most of the network is dominated by a uniform set of weights, and thus it can be seen as just a slightly modified version of the topology model. In this case it may simply be the case that clustering isn't useful. Perhaps the direct segment metrics provided by the web app are enough to identify problematic areas.
The other scenario was that there appears blockings all across the network. This creates a sort of noise applied to the adjacency matrix. It will be difficult for the spectral clustering algorithm to clearly identify areas that have significant and coherent blocking behaviors.

Segment weight diffusion

When creating weighted adjacency matrices there are areas of the layout that display a fragmented segment weight behavior. What we mean by this is that there isn't always a continuous change of weights when moving along the graph. In the most prominent model, the blocking time model, there are often segments with no blockings scattered between segments with a lot of blockings. This creates low weights and hence the area is viewed as weakly connected even if it perhaps creates a high amount of blockings and such should be highlighted by a model as a cluster. For this to happen we need to explore some sort of method for spreading out the weights of one segment to nearby connected segments. This can be done with some type of heat kernel approach perhaps, were each segments current weight acts as a start ”temperature” and it then spreads out to some adjacent areas. The benefit of such an approach is that the blocking data will be more continuous, and all segments with or without blockings will get a high weight if they are near heavily blocked segments. This in turn creates a Laplacian matrix that highlights the connectedness in these problematic blocking areas. The result should be a more continuous segment weight distribution that fluctuates based on a sample of nearby segments not only the actual segment. Clustering on the spectral embedding of such a matrix should enable easier detection in these sparsely blocked areas that are prevalent in AGV systems.

6.2.2 Clustering analysis

Analysis of the clusters is integral to acquiring useful results. Just looking at the visual cluster overlay doesn't explain much about the system dynamics. To get the full understanding of the behavior we need to combine the insights from three graphs: the Physical view, the Abstract view and the cluster metrics. Lets examine an example, see figure 5.11. The physical view showcases the geometry of the clusters. This is the view that displays which nodes belong to which cluster. To gain further insight into how these clusters interact we look at the abstracted view. Here insight about how traffic flows between the nodes is gained. As an example, here it can be seen that the clusters with the most traffic between them are cluster 0 and 4. To see which clusters are the most significant from a congestion perspective we need to examine the cluster metrics. This plot provides insights into which clusters are the most trafficked and which have high proportional levels of blockings. It is especially the blocking severity that is of importance to the congestion analysis. Clusters with a blocking severity lower than 1 can probably be discarded as insignificant due to traffic here flowing well. The cluster(s) with more than 1 blocking severity should probably be examined further as these areas are the most problematic. With an overview of these three graphs the user has the necessary tools to investigate the system behavior. Lets examine some scenarios.

Defining what constitutes a useful cluster

A problem with analyzing the resulting clusters is that it is hard to define what would be a ”good” cluster. This is true for any model. Optimally we would want

to divide the network into insightful clusters that instantly give the user valuable information about the layout at hand. Since the data is unlabeled, i.e. no ground truth to the clusters is obtainable, there is no direct way to determine if the clusters are correct. This leads to a situation were clustering is performed to try to avoid a user having to rely on intuition, but the resulting clusters require a healthy dose of intuition to understand. To fully understand and evaluate the potential benefits of the clusters it would be advised to perform a deeper study into what insights layout designers of different experience levels are able to extract from the clusters. This could be a good way to solve the problem that we currently have were we create a clustered network, but don't fully understand what benefits this entails. If there was clear rules or ideas of what a good clustering would look like it would be possible to train models to mimic those behaviors, or directly implement the ideas if well defined.

Custom cluster modification

Another idea that could be interesting is to let a user control how the clusters look like. One could imagine an application that interactively gathers the metrics from each cluster and displays them as graphs. A user can combine, move, split and perform other transformations to the cluster borders to manipulate how they are connected and thus this lets the user fully self configure the clusters. This could either be done from the clusters of a model or by letting the user start clustering from scratch. The benefits to this is that the insights from one step can spark ideas that makes a user want to understand how other areas of the network behave and interact together. This method could be seen as a custom aggregating method based on user defined areas.

Hybrid methods

To further the analysis of a cluster it could be very interesting to combine cluster results with the blocking graphs. Choosing to focus on one cluster would then show all blocking graphs in that cluster, and the associated metrics. This would give a in depth analysis of specific areas, something the application for displaying metrics isn't capable of. This would open lots of new doors for insights, especially with the cluster modification capabilities enabled. Blocking model only takes blocking time into account, another idea is total traversal time

Other options for clustering

One thing that should be explored further is options to spectral clustering. We realized during the later stages of our thesis that the data we have isn't really suited for spectral clustering so it could make sense to try other clustering models for graphs.

7
Conclusion

At the start of this thesis, we set out to assist application engineers in understanding their data better, in turn helping bridge the gap needed to make data-driven decisions when designing layout. In pursuing this goal, we created a data structure, *blocking graphs*, that enables novel insights into blocking behavior. We also explored different ways of aggregating, visualizing and interactively traversing this data structure. This has shown promise in gaining novel insights about congestion in the system. To reach the full potential, more work is needed on turning these insights into suggestions or actions. Additionally, we explored spectral clustering as a tool to provide insights on the large-scale traffic dynamics. In this work, different models of describing system behavior were motivated and applied. Clustering analysis was enabled by providing three different views. These visualizations proved to be interesting as a way to obtain an overview of the system. However, we are not convinced spectral clustering with system behavior modeling is the best way to provide the clusters (which the visualizations are based on).
With the novel tools and insights created, we believe that the layout design process can be more streamlined than before. Especially, by providing concrete insights into the root-cause and context of each blocking in a system. By aggregating blocking graphs over a simulation, we can provide a layout designer with the symptoms and causes of congestion. This allows the designer to identify not only what specific segment is the most problematic, but also why this is the case.
To further increase the value of the results, we propose some future research directions:

- In our thesis we have quantified blockings only by their duration. Since blocking time can be deceiving and not representative of only harmful behavior we propose researching what constitutes a wasteful blocking. If this was clearly defined the blocking graphs could show where the most waste was generated, which would be more reliable than just the blocking duration.
- More effort is needed into how to integrate blockings graphs into a layout designer workflow. We already know that providing a blocking with it's context and reason is incredibly useful, but the most intuitive way to visualize this information is unclear.
- The idea of providing a system overview has proven to be valuable, but the means to obtain it might need reconsideration. We have explored some options, but further studies are required to understand what type of insights would be helpful from an overview.

This thesis explored two methods for describing congestion in AGV systems, aiming to streamline the layout design process and enhance our understanding of these

complex systems. We used descriptive analysis to show how one blocked vehicle can lead to a chain reaction, blocking others and forming causal chains. This approach provided deeper insights into where traffic is most significant and what the primary causes are.
To analyze the system from a broader perspective and identify macro congestion behavior, we not only applied spectral clustering to the AGV system but also adapted it specifically for this domain. Using novel models, we introduced a new way of examining AGV systems through the lens of traffic dynamics. We believe that the results of this thesis lay a strong foundation for further research in the field, offering valuable insights and methodologies for future studies.

Bibliography

[1] Grand View Research. *Logistics Automation Market Size & Trends*. Accessed: 2024-12-09. 2024. URL: https://www.grandviewresearch.com/industry-analysis/logistics-automation-market-report.

[2] Günter Ullrich and Thomas Albrecht. "History of Automated Guided Vehicle Systems". In: *Automated Guided Vehicle Systems: A Guide - With Practical Applications - About The Technology - For Planning*. Wiesbaden: Springer Fachmedien Wiesbaden, 2023, pp. 1–26. ISBN: 978-3-658-35387-2. DOI: 10.1007/978-3-658-35387-2_1. URL: https://doi.org/10.1007/978-3-658-35387-2_1.

[3] Grand View Research. *Global Automated Guided Vehicle Market Size & Outlook*. Accessed: 2024-12-09. 2024. URL: https://www.grandviewresearch.com/horizon/outlook/automated-guided-vehicle-market-size/global.

[4] Béla Bollobás. *Modern Graph Theory*. 1st ed. Graduate Texts in Mathematics. Published: 01 July 1998, eBook Published: 01 December 2013, Springer Book Archive. Springer New York, NY, 1998. ISBN: 978-0-387-98488-9. DOI: 10.1007/978-1-4612-0619-4.

[5] F.R.K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series nr. 92. Conference Board of the Mathematical Sciences, 1997. ISBN: 9780821889367. URL: https://books.google.se/books?id=YUc38_MCuhAC.

[6] Ulrike von Luxburg. "A Tutorial on Spectral Clustering". In: *CoRR* abs/0711.0189 (2007). arXiv: 0711.0189. URL: http://arxiv.org/abs/0711.0189.

[7] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: https://doi.org/10.1016/S0169-7552(98)00110-X. URL: https://www.sciencedirect.com/science/article/pii/S016975529800110X.

[8] Willem H. Haemers Andries E. Brouwer. *Spectra of Graphs*. 1. Springer New York, NY, 2011. DOI: https://doi.org/10.1007/978-1-4614-1939-6.

[9] Daniel A. Spielman. "Spectral Graph Theory and its Applications". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. 2007, pp. 29–38. DOI: 10.1109/FOCS.2007.56.

[10] William L. Hamilton. "Graph Representation Learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 1–159.

[11] F Chung. "Laplacians and the Cheeger Inequality for Directed Graphs". In: *Ann. Comb* 9 (2004), pp. 1–19. DOI: https://doi.org/10.1007/s00026-005-0237-z.

[12] David Austin. *Understanding Linear Algebra*. Department of Mathematics, Grand Valley State University, 2023.

[13] Miroslav Fiedler. "Algebraic connectivity of graphs". eng. In: *Czechoslovak Mathematical Journal* 23.2 (1973), pp. 298–305. URL: http://eudml.org/doc/12723.

[14] Robert A. Beezer. *A First Course in Linear Algebra*. GNU Free Documentation License, 2015. URL: hhttp://linear.pugetsound.edu/html/fcla.html5.

[15] M. Bolla. *Relations Between Spectral and Classification Properties of Multigraphs*. DIMACS technical report. DIMACS, Center for Discrete Mathematics and Theoretical Computer Science, 1991. URL: https://books.google.se/books?id=vRlxHAAACAAJ.

[16] William R. Palmer and Tian Zheng. "Spectral Clustering for Directed Networks". In: (2021). Ed. by Rosa M. Benito et al., pp. 87–99.

[17] Lovász László, L. Lov, and Of Erdos. "Random Walks on Graphs: A Survey". In: *Combinatorica* (Jan. 1996), pp. 1–46.

[18] Hongyuan Zha et al. "Spectral Relaxation for K-means Clustering". In: *Adv. Neural Inf. Process. Syst.* 14 (Apr. 2002).

[19] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979). Accessed 9 Dec. 2024, pp. 100–108. DOI: 10.2307/2346830. URL: https://doi.org/10.2307/2346830.

[20] Aykut Argun, Agnese Callegari, and Giovanni Volpe. *Simulation of Complex Systems*. 2053-2563. IOP Publishing, 2021. ISBN: 978-0-7503-3843-1. DOI: 10.1088/978-0-7503-3843-1. URL: https://dx.doi.org/10.1088/978-0-7503-3843-1.

[21] Cindy E. Hmelo-Silver and Roger Azevedo. "Understanding Complex Systems: Some Core Challenges". In: *The Journal of the Learning Sciences* 15.1 (2006), pp. 53–61. ISSN: 10508406, 15327809. URL: http://www.jstor.org/stable/25473509 (visited on 12/16/2024).

[22] "Frontiers in Complex Systems". In: 1 (2023). ISSN: 2813-6187. DOI: 10.3389/fcpxs.2023.1080801. URL: https://www.frontiersin.org/journals/complex-systems/articles/10.3389/fcpxs.2023.1080801.

[23] Adrian Diaz de Rivera and C. Tyler Dick. "Illustrating the implications of moving blocks on railway traffic flow behavior with fundamental diagrams". In: *Transportation Research Part C: Emerging Technologies* 123 (2021), p. 102982. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2021.102982. URL: https://www.sciencedirect.com/science/article/pii/S0968090X21000188.

[24] Ning Zhao. "Railway Traffic Flow Optimisation with Differing Control Systems". In: (Aug. 2012).

[25] Ludolf E. Meester and Sander Muns. "Stochastic delay propagation in railway networks and phase-type distributions". In: *Transportation Research Part B: Methodological* 41.2 (2007). Advanced Modelling of Train Operations in Stations and Networks, pp. 218–230. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2006.02.007. URL: https://www.sciencedirect.com/science/article/pii/S0191261506000221.

[26] N. Andrienko et al. “Leveraging Spatial Abstraction in Traffic Analysis and Forecasting with Visual Analytics”. In: *Machine Learning and Knowledge Discovery in Databases* 99 (2016), pp. 32–35. DOI: https://doi.org/10.1007/978-3-319-46131-1_7.

[27] R. Besenczi et al. “Large-scale simulation of traffic flow using Markov model”. In: *PLOS ONE* (2021). DOI: https://doi.org/10.1371/journal.pone.0246062.

[28] Nikolas Geroliminis Et al. “Existence of urban-scale macroscopic fundamental diagrams: Some experimental findings”. In: *Transportation Research Part B: Methodological* 42.9 (2008). DOI: https://doi.org/10.1016/j.trc.2021.102982.

[29] K. Almatar et al. “Traffic congestion patterns in the urban road network: (Dammam metropolitan area)”. In: *Ain Shamps Engineering Journal* 14.3 (2023). DOI: https://doi.org/10.1016/j.asej.2022.101886.

[30] S. Salman et al. “Alleviating road network congestion: Traffic pattern optimization using Markov chain traffic assignment”. In: *Computers & Operations Research* 99 (2018), pp. 191–205. DOI: https://doi.org/10.1016/j.cor.2018.06.015.

[31] Mark M. Dekker. “Geographic delay characterization of railway systems”. In: *Scientific Reports* 11.1 (Oct. 2021). ISSN: 2045-2322. DOI: 10.1038/s41598-021-00361-z. URL: http://dx.doi.org/10.1038/s41598-021-00361-z.

[32] Linton C. Freeman. “A Set of Measures of Centrality Based on Betweenness”. In: *Sociometry* 40.1 (1977), pp. 35–41. ISSN: 00380431, 23257938. URL: http://www.jstor.org/stable/3033543 (visited on 03/11/2025).

[33] M. N. S. Thulasiraman K.; Swamy. “Directed Graphs”. In: *Graphs: Theory and Algorithms*. John Wiley & Sons, Ltd, 1992. Chap. 5.7, p. 118. ISBN: 9781118033104. DOI: https://doi.org/10.1002/9781118033104.ch5.

[34] Guido van Rossum and the Python development team. *Python Reference Manual*. Python Software Foundation. 2023. URL: https://docs.python.org/3/reference/.

[35] Pieter J. Swart Aric A. Hagberg Daniel A. Schult. *NetworkX Reference Manual*. NetworkX Developers. 2024. URL: https://networkx.org/documentation/stable/reference/index.html.

[36] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[37] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[38] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[39] Riksdagen. *Punktlighet för persontrafik på järnväg - en uppföljning*. Riksdagstryckeriet, Stockholm 2020. Rapport från riksdagen 2020/21:RFR5. 2020. URL: https://www.riksdagen.se/sv/dokument-och-lagar/dokument/rapport-fran-riksdagen/punktlighet-for-persontrafik-pa-jarnvagen_h80wrfr5/.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY