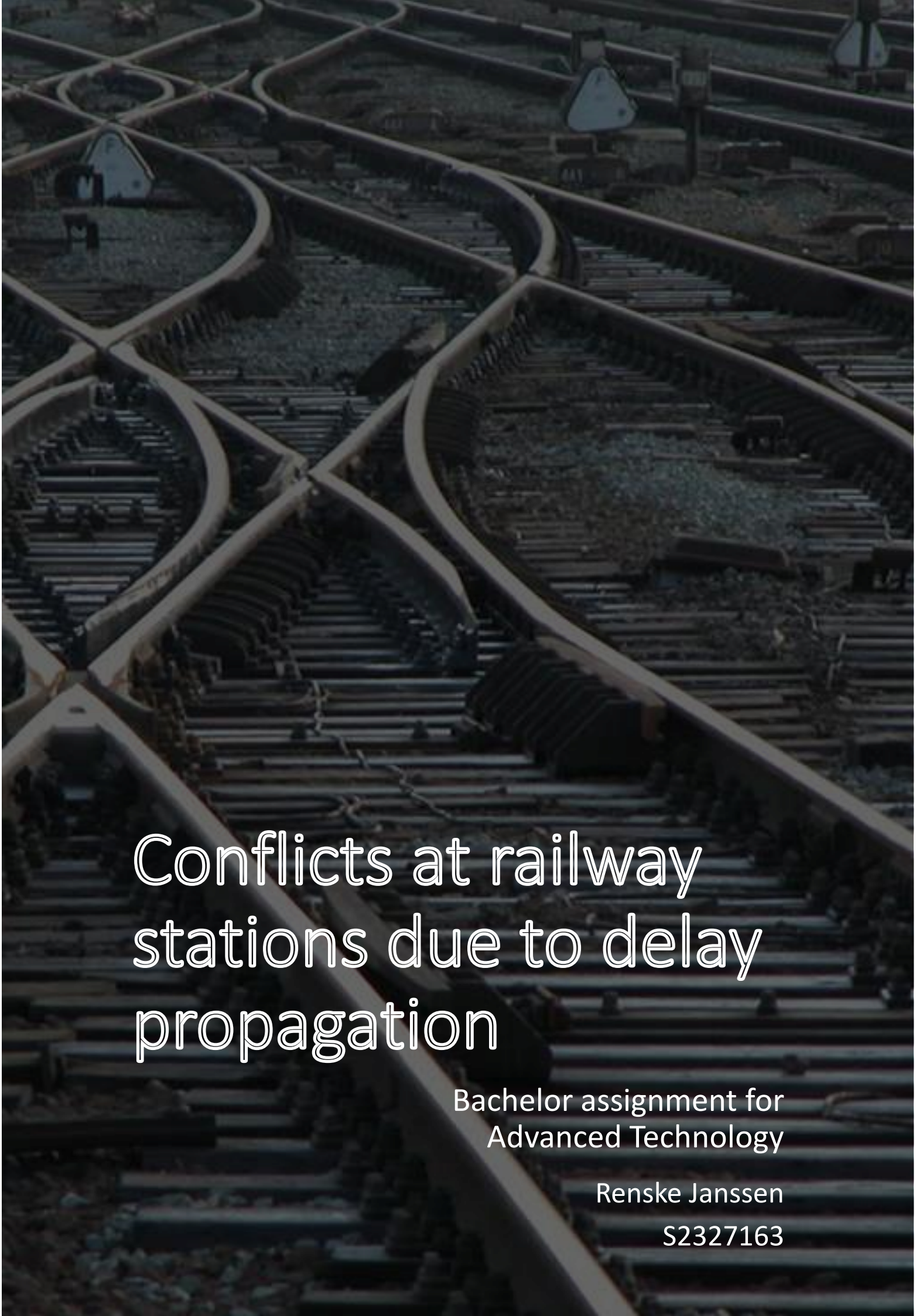

Conflicts at railway stations due to delay propagation

Bachelor assignment for
Advanced Technology

Renske Janssen

S2327163

Conflicts at railway stations due to delay propagation

an evaluation of Amersfoort C. and Arnhem C.

Renske Janssen
(s2327163)

Supervisors:
Prof. Dr. Nico van Dijk
Prof. Dr. Richard Boucherie
Dr. Konstantinos Gkiotsalitis

Research group: SOR

UNIVERSITY
OF TWENTE.

July 2022

Abstract

In railway systems, secondary delays occur when a delayed train causes delays for other trains as well. In the Dutch Railway Network, for passengers this seems to happen regularly, but it is not tracked how often it occurs. To find out, this report proposes a mathematical model based on queuing theory from communication systems, resulting in an acceptance probability for the train lines arriving at the station. Next to the model, a simulation is made with the same assumptions. The results for both are compared. Also, insensitivity was proven for the model and tested in the simulation, resulting in a more accurate representation of reality. Both were applied to a simple made-up station and the Dutch station Arnhem C. and Amersfoort C. For the latter, with input variables based on data, a general acceptance rate of 46% and 69% was found, respectively, but in both cases it highly depends on the different arriving train lines. As multiple assumptions have been made for the model, this will be an overestimation of reality and it can be seen as an upper limit. Improvements for the model have been proposed to attain more realistic results.

Contents

1 Introduction

Climate change makes us reconsider travel choices. To reduce travel pollution, switching from car and plane to train travel could be a part of the solution. However, the delays at Dutch Railways (NS) and their uncertainties are one of the main reasons for dissatisfaction when traveling by train [1]. Delays in the railway system can occur due to different reasons, such as weather conditions, system malfunction, occupancy, etc. Such train-specific delays are called primary delays. However, a major part of train delays in the Dutch Railway Network are those caused by other delayed trains[2], the so-called knock-on or secondary delays. To improve on a train timetable and make it more robust, it will be interesting to know where and how this type of delay occurs.

The Dutch Railway Network is a complex system of stations, signaling lights, train switches, and other (infra)elements. Making a robust train schedule is not an easy task. The utilization of the capacity should be high to serve all, yearly increasing number of customers, mainly during peak hours. On the other hand, it should provide enough space between trains so that fewer secondary delays occur and the overall punctuality increases. The punctuality of the NS in 2019 was, according to their own research, 92.6%[3]. However, delays within 5 minutes are also considered by the NS to be on time, but these short delays can result in a missing connection for travelers. Until 2013 a 3-minute punctuality was also measured, at that time it was 87.4% and 5-minute punctuality of 93.6% [4]. Although the numbers are not bad, no improvement over the last few years has been made and it would be better to measure and strive for higher punctuality in shorter time frames.

The capacity can be determined in different ways. The International Union of Railways (UIC) code 406 is the most widely used method [5]. The NS uses its own build simulation programs to evaluate the capacity and also to see the influence of delays in the system. The performance of one of these programs, named RailSys, is analyzed for the Swedish railway network in [6], but it was concluded that it cannot be used to determine node capacities. These current simulation programs do not take into account the relation between factors that affect the delay propagation, such as the behavior of train drivers (faster/slower riding). It turns out that evaluating the delay propagation for a full railway network will most often not give a correct indication due to this large number of factors.

Therefore, delay propagation can be better analyzed for specific and smaller part of the network to get more correct results. The propagation of delays can occur in different ways and at different places. [7] considers the delays due to different running times of different trains, where a faster train can be held up by slower trains on the same track. [8] uses Markov chains to predict delay throughout a line considering arrival, departure and dwell times. A specific type of Markov chain, a quasi-birth-and-death process, is used in [9] to model train operations, failure characteristics, and capacity change.

Whereas the three studies above all consider lines between stations, already in the early days of railway analysis, it was discovered that the main bottlenecks are located at stations. A delayed train can take up a spot on a platform of another train, resulting in a delay for that train as well. Stations are more difficult to model than tracks/lines as there are more options for trains to go to. Therefore, more complicated simulations should be made or more advanced

mathematical tricks used. The stations can be modeled via both linear programming and queueing theory. So far, linear programming has not given correct results as it is difficult to set up an objective function, e.g. combining the cost for delays and deviations of trains to other platforms as in [10]. [11] also uses linear programming to analytically estimate the propagation of knock-on delays, tested this for a part of the Dutch railway station Den Haag Holland Spoor and used it to optimize capacity utilization.

Markov chains and queueing theory can also be applied for station modeling. This mathematical modeling theory originates from communication systems, but when the incoming rail is seen as the incoming line and a platform as the server, the equivalence can be seen. One of the most used queues, also called the Erlang queue, describes a queue with an arrival rate according to a Poisson distribution, an expected serving time of $E[B]$, c number of servers and a maximum of c trains waiting. A queuing model tries to represent reality, but assumptions often have to be made. How an Erlang queuing model approximates reality and what its error bounds are is evaluated in [12].

Different assumptions can be made, and different combinations of types of queues can be used when modeling a station and incoming and outcoming trains. For example, a standard Erlang queue describes the whole station in [13] after which it is numerically applied to a freight station in Russia and a marshaling yard in the USA. In [14], sets of infra-elements are treated separately, each having its own Erlang queue with one server and an infinite waiting room. An analysis of the literature of multiple proposed methods is performed in [15], and the methods evaluated are applied to stations in China.

Although much research has been done on delay propagation at the railway station, not many studies have been performed on the Dutch Railway system, which differs quite a lot from the railways in USA, Russia and China because the system is denser and trains follow each other up more quickly. Therefore, in this report, an analytic model based on birth-death queuing and the queuing theory typically used for communication systems is proposed for a railway station. This type of queue is a queue with 1 server and no waiting room. The main goal is to analyze the blocking probability for the arriving trains and where in the station these blockages occur, the infra-elements that withhold a train to arrive at the station because of another train that occupies that infra-element. Next, a simulation model is made, with the goal is to visualizing and locating the conflict points. The analytic model and simulation model are compared and the validity of both is tested with data from reality. Next to a simple made-up station, the Dutch stations that are modeled and simulated are Arnhem Centraal and Amersfoort Centraal.

In the second chapter of this article, an overview of the theory used in the report will be given. The third and fourth chapter consists of an explanation of the analytic model and simulation respectively. The comparison and validation for the three evaluated stations are written in the fourth, fifth, and sixth chapters, after which a discussion and conclusion follow in the seventh chapter.

Part I
Mathematical model

2 Theory

The model proposed in section 3 is based upon continuous time Markov chains, specifically birth-death queues. To better explain the model, first an introduction to the theory used in the model is given based upon [16] and [17].

2.1 Continuous Time Markov chains

Consider a system that can be described by various states, depending on the situation the system is in, e.g. the number of people in a queue, the position of a robot, etc. The total of all admissible states of such a system is described by the discrete state space C. The system can jump between a state i and a state j with a probability of p_{ij}, called the transient probability and which should be independent of the duration that the system is in state i.

In such a system, we can also define an infinitesimal transition rate as how often the system switches from a state i to state j in a very small, infinitesimal time frame or:

$$q_{ij} = \lim_{\Delta t \downarrow 0} \frac{1}{\Delta t}[P(X(t+\Delta t) = j | X(t) = i)] \tag{2.1}$$

With this transition rate, probability that the system switches from state i to state j in the upcoming (very small) time interval Δt is given by:

$$P(X(t+\Delta t) = j | X(t) = i) = q_{ij}\Delta t + o(\Delta t) \qquad \forall i, j \in C$$

And, in the same way, for the probability that systems stays in state i with this upcoming small time interval

$$P(X(t+\Delta t) = i | X(t) = i) = 1 - \sum_{j \neq i} q_{ij}$$

From these definitions, we can define a variable v_i such that

$$\nu_i = \sum_{j \neq i} q_{ij} \text{ and } q_{ij} = \nu_i p_{ij} \qquad \forall i, j \in C \tag{2.2}$$

This value also describes the exponentially distributed time the system is in a state i, with an expected value of $1/\nu_i$.

Now, we can define the random variable $X(t)$ to be:

$$X(t) = \text{state of the system at time t}$$

The stochastic process $\{X(t), t \geq 0\}$ is an continuous time Markov chain (CTMC) as the current state of the system does not influence the next state, nor does does any of the other states before.

Definition 1 (Continuous Time Markov chain)**.** *A process $\{X(t), t \geq 0\}$ with countable or finite state space C is a continuous time Markov chain if*

$$P(X(t+s)=j|X(s)=i, X(u)=x(u), 0\leq u<s) = P(X(t+s)=j|X(s)=i)$$
$$\forall s,t>0, \text{ for every } x(u)=0, 0\leq u<s \text{ and } i,j\in C \tag{2.3}$$

Transient probabilities

As described above the transient probability is the probability that a system switches from a state i to a state j. These can be described by

$$p_{ij}(t) = P(X(t)=j|X(0)=i) \qquad \forall i,j\in C, t\geq 0 \tag{2.4}$$

The matrix $\mathbf{P}(t)$ describes all transient probabilities, for all $i,j\in C$.

Ergodicity

Definition 2 (Ergodic)**.** *A CTMC is ergodic if for any initial state i, the probability that the system described by the Markov chain will eventually reach the state r is 1 and that the expected time to reach this state is finite.*

When a CTMC is ergodic, one can define the limiting probability for a state j to be

$$\pi_j = \lim_{t\to\infty} p_{ij}(t) \tag{2.5}$$

Due to the continuity, in the cases of this report, it will always exist and be independent of state i.

Theorem 2.1. *For every $i\in C$, the following holds:*

$$\frac{dp_{ij}(t)}{dt} = \sum_{k\neq j} p_{kj} - p_{ij}(t)\nu_j \quad \textit{for } j\in C \quad \textit{and } t>0 \tag{2.6}$$

Proof. Here, this theorem is only proven if I is finite, which will be true in case of railway systems. It can be shown that as for a small time-frame, Δt, the following holds:

$$\begin{aligned} p_{ij}(t+\Delta t) &= P(X(t+\Delta t)=j|X(0)=i) \\ &= \sum_{k\in C} P(X(t+\Delta t)=j|X(t)=k, X(0)=i)P(X(t)=k|X(0)=i) \\ &= \sum_{k\in C} P(X(t+\Delta T)=j|X(t)=k)p_{ik}(t) \end{aligned} \tag{2.7}$$

If now the summation is split into $k\neq j$ and $k=j$, the following it obtained:

$$\frac{p_{ij}(t+\Delta t)-p_{ij}(t)}{\Delta t} = \sum_{k\neq j} q_{kj}p_{ik}(t) - \nu_j p_{ij}(t) + \frac{o(\Delta t)}{\Delta t} \tag{2.8}$$

If Δt goes to zero, it meets the desired result. □

Theorem 2.2. *If a Markov chain is ergodic, the limiting probabilities π_j for all $j \in C$ from an unique solution to the global balance equations (2.9) and normalization equation (2.10).*

$$v_j \pi_j = \sum_{k \neq j} \pi_k q_{kj} \qquad \forall j \in C, \tag{2.9}$$

$$\sum_{j \in C} \pi_j = 1 \tag{2.10}$$

Proof. The complete theory will not be proven here, only for I is finite it will be shown that it holds. It is known that for $t \to \infty$, $p_{ij}(t) \to \pi_j$ and, by theorem 2.1, $p'_{ij} \to 0$. So, in the case that time goes to infity, $\sum_{k \neq j} = \pi_k q_{kj} - \pi_j \nu_j = 0$ for all $j \in C$. The normalization condition is also met, as for $t \to \infty$, $\sum_{j \in C} p_{ij}(t) = 1$. □

Long run time fraction

Define indicator $1_{\{A\}}$ as:

$$1_{\{A\}}(m) = \begin{cases} 1 & \text{if A is true at time m} \\ 0 & \text{if A is false at time m} \end{cases}$$

Assume a time interval $[1, n]$ and let $1_{\{j\}}(m)$ be an indicator for a system at time m to be in state j. Then the expected proportion of the time that the system is in this state is:

$$\begin{gathered} E\big(\frac{1}{n} \sum_{m=1}^{n} 1_{\{j\}}(m)\big) = \\ \frac{1}{n} \sum_{m=1}^{n} P(X(m) = j | X_0 = i) = \frac{1}{n} \sum_{m=1}^{n} p_{ij}(m) \end{gathered} \tag{2.11}$$

Because of (2.5), this is equal to π_j. So, the limiting probability π_j can also be considered as the actual long run fraction of the time that the system is in state j.

With the property PASTA property (*Poisson Arrivals See Time Averages"*, it can be proven that the probability of that the system is in a certain state for an outside observer is the same for an arriving customer, provided the arrival process is a Poisson process.

Balance equations

Assume the system is in a state k and knowing the limiting probability of π_k, the long-run expected number of transition per unit from this state k to state j can described as $\pi_k q_{kj}$. So total expected number of transition per unit time from any state to state j being $\sum_{j \neq k} \pi_k q_{kj}$

At the same time, the long-run expected number of transition per unit time from state j to any other state can be described as $\pi_j v_j = \sum_{j \neq k} \pi_j q_{jk}$. And in the long run the input in state j should be the same as the output of state j, therefore the global balance equations can also be written as:

$$\sum_{j \neq k} \pi_j q_{jk} (\text{rate out}) = \sum_{j \neq k} \pi_k q_{kj} (\text{rate in}) \tag{2.12}$$

Reversibility

Definition 3 (Reversibility)**.** *A Markov chain with steady states $\pi_j \; \forall j \in C$ is reversible if and only of*

$$\pi_j q_{jk} = \pi_j = k q_{kj} \; \forall i, j \in C \tag{2.13}$$

These reversibility equations satisfy the balance equations (2.12) and are the most detailed form in which the global balance equations (2.12) can be written. They are therefore also called the *detailed balance equations.*

2.2 Birth-death queues

A special type of continuous Markov chains is the so-called birth and death process. Let a system be described by a continuous Markov chain with a state space $C = \{0, 1, 2, ...\}$. The system transitions from a state j to a state $j + 1$ with an exponential rate λ_j (a birth) and from a state j to a state $j - 1$ with an exponential rate of μ_j (a death). Assume that $\lambda_j > 0$ and $\mu_j > 0$ for all j. Now the Markov chain can be described by

$$\begin{aligned} v_j &= \lambda_j + \mu_j && \text{for } j > 0 \\ v_o &= \lambda_j \end{aligned}$$

And the infinitesimal transition rates are

$$q_{ij} = \begin{cases} \lambda_j : \text{if } k = j + 1, i \geq 0 \\ \mu_j : \text{if } k = j - 1, i \geq 1 \\ \nu_j : \text{if } k = j, j \geq 0 \end{cases}$$

According to global balance equations (2.9), the global balance equations can now be written as:

$$\begin{aligned} \lambda_0 \pi_0 &= \mu_1 \pi_1 && (j = 0) \\ \lambda_j \pi_j + \mu_j \pi_j &= \lambda_{j-1} \pi_{j-1} + \mu_{j+1} \pi_{j+1} && (j \geq 1) \end{aligned}$$

Combining these two equations (or according to the detailed balance (2.13)), it can be concluded that:

$$\lambda_j \pi_j = \mu_{j+1} \pi_{j+1} \tag{2.14}$$

or equivalently

$$\lambda_{j-1} \pi_{j-1} = \mu_j \pi_j \tag{2.15}$$

The latter also has a physical rate in versus rate out. As these equations can be written this, the reversibility condition as in definition 3 is met.

When iterating this equation, the steady state distributions can be found:

$$\pi_j = \pi_0 \sum_{k=0}^{j-1} \frac{\lambda_k}{\mu_{k+1}} \tag{2.16}$$

Such a system of birth and deaths is also used for the railway station model, only it will be adapted to a multidimensional frame.

2.3 The exponential distribution

In the birth-death queue, it is assumed the birth and death rate follow a Poisson distribution meaning that the time between two births (or two deaths) is exponentially distributed. Where the Poisson distribution describes the number of occurrences within a certain time period, the exponential distribution described the time between two such occurrences. The probability density functions of the exponential distribution is given in (2.17). This distribution has interesting properties.

$$f(x, \lambda) = \lambda e^{-\lambda x} \tag{2.17}$$

Definition 4 (Memoryless property)**.** *A random variable X has the memoryless property if its distribution satisfies*

$$P(X > a + b | X \geq a) = P(x > b) \qquad \forall a, b \geq 0 \in$$

Theorem 2.3. *The exponential distribution has the memoryless property.*

Proof. Consider the random variable X with has an exponential distribution with mean $1/\lambda$ and $\forall a, b > 0$ then

$$\begin{aligned} P(X > a + b | X \geq a) = \frac{P(X > a + b, A \geq a)}{P(A > a)} = \frac{P(X > a + b)}{P(X \geq a} \\ = \frac{e^{\lambda(a+b)}}{e^{\lambda a}} = e^{\lambda a} = P(X > h) \end{aligned} \tag{2.18}$$

□

As the exponential distribution describes the time between two events of a Poisson process, it is also possible to describes the distribution of the sum of a number (integer) k Poisson events each with mean $1/\lambda$. This distribution is called the Erlang-(k, λ) distribution, described by the following probability density as in (2.19).

$$f(x, k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \tag{2.19}$$

The mean value of an Erlang distribution is k/λ and the variance is k/λ^2. Therefore, with increasing both k and λ with the same rate, the mean will stay the same while the variance decreases, being able to approach a discrete situation. An example can be seen in figure 2.19. Here the mean values for both figures is 2, but the left function has variance 0.05, while the right one has variance 0.005, due to the increase λ and k. Therefore, the peak is smaller and steeper. Increasing both values even more, with the same rate, will result in an even smaller variance.

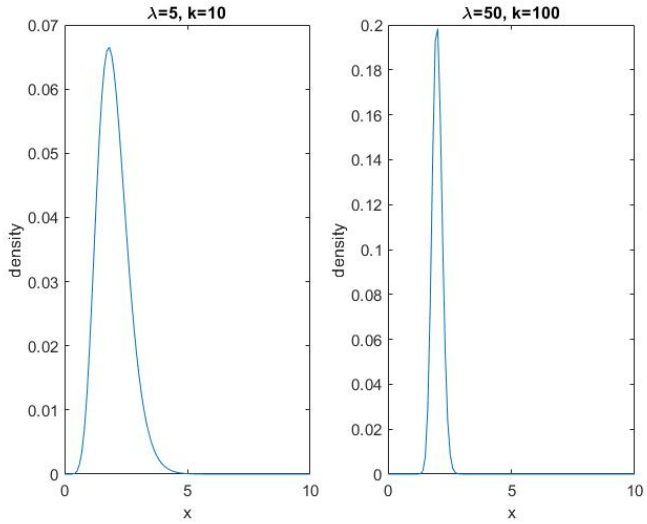

Figure 2.1: *Probability density function for an Erlang distribution with mean 2, but with different input variables*

3 Mathematical model for a railway station

Using the theory of continuous time Markov chains and birth-death processes, a simple model of train arrivals at a station is made. This model is based upon communication networks. The model only takes into account the arrival at a station and assumes either that a train arrives, is handled and after that removed or that a train is blocked and vanishes.

Take a station that that S different arriving train lines. One could say that each of these form a different source s, which if busy generates a train for that line. Such a source has:

$$\begin{cases} \lambda_s : \text{an exponential scheduling rate} \\ \mu_s : \text{an exponential transmission rate} \end{cases}$$

Let state $B = \{s_1, s_2, ..., s_n\}$ denote the set of busy sources, and let space C be the state space of admissible states, indicating which sources can be busy at the same moment. Then $A(s|B)$ can be introduced as the access function, where we can assume:

$$\begin{aligned} A(s|B) &= 1 \text{ for } B + s \in C \\ A(s|B) &= 0 \text{ for } B + s \notin C \end{aligned} \tag{3.1}$$

Or in words, a source can become busy if its addition to the current busy sources is possible by the set of admissible combinations.

Furthermore, assume a certain state B, having a combination of sources that are busy, is possible. Then any state $B - s$, where one of the busy sources in state B has become idle, has also to be possible. This is because less arriving trains is for sure possible if more arriving trains is possible. So blocking can only occur when a source becomes busy, not when becoming idle. Mathematically, we can write:

$$B \in C \Rightarrow B - s \in C \; \forall s \in B \tag{3.2}$$

This property is called *coordinate convex blocking*, and due to it the system is reversible.

3.1 Product form

An idle source $s \notin B$ generates a train with an rate

$$\lambda_s A(s|B) \tag{3.3}$$

From this the following transition rates follow

$$\begin{aligned} q_{B,B+s} &= \lambda_s A(s|B) \quad (s \notin B) \\ q_{B,B-s} &= \mu_s \qquad\qquad\quad (s \in B) \end{aligned} \tag{3.4}$$

The remaining transition rates will be zero: $q(B, B') = 0$. Now, according to equation(2.12), the balance equations can be written as:

$$\sum_{B'} \pi(B) q(B, B') = \sum_{B'} \pi(B') q(B', B) \tag{3.5}$$

With filling in the transition rates (3.4) into (3.5).

$$\begin{aligned} &\sum_{s\in B}\pi(B)\mu_s + \sum_{s\notin B}\pi(B)\lambda_s A(s|B)1_{\{B+s\in C\}} \\ =&\sum_{s\in B}\pi(B-s)\lambda_s A(s|B-s) + \sum_{s\notin B}\pi(B+s)\mu_s 1_{\{B+s\in C\}} \end{aligned} \tag{3.6}$$

as we assume (3.1) and imply the condition that

Theorem 3.1. *Define the following product to be*

$$P(s_1, s_2, ..., s_n) = \prod_{k=1}^{n} A(s_{i_k}|s_{i_1}, ..., s_{i_{k-1}}) \tag{3.7}$$

With this product $P(s_1, ..., s_n)$ the source balance (3.5) can be written as

$$\pi(B) = cP(B)\prod_{s\in B}(\frac{\lambda_s}{\mu_s}) \tag{3.8}$$

Proof. Any permutation $(i_1, ..., i_n) \in (1, ..., n)$ does not change the outcome of P, since this product will be

$$P(B) = 1 \quad B \in C \tag{3.9}$$

$$P(B) = 0 \quad B \notin C \tag{3.10}$$

Therefore, the detailed balance equation as described in (2.9) can be used, filling this in the following is obtained. (This is can also be done via (3.6) as for a generating source s for any $B \in C$ and $s \in B$ the second part of both sides fall out):

$$\pi(B)\mu_s = \pi(B-s)\lambda_s A(s|B-s) \tag{3.11}$$

Now, by using equations (3.8) and (3.7), the detailed balance equations (3.11) are verified as follows:

$$\frac{\pi(B)}{\pi(B-s)} = \frac{cP(B)\prod_{a\in B}(\frac{\lambda_a}{\mu_a})}{cP(B-s)\prod_{a\in B-s}(\frac{\lambda_a}{\mu_a})} = \frac{\lambda_s}{\mu_s}\frac{P(B)}{P(B-s)} = A(s|B-s)\frac{\lambda_s}{\mu_s} \tag{3.12}$$

□

Expression (3.8) is called the *product form*. With this equation, the steady state distributions for all state of arriving trains can be determined. For each state, its steady distributions is, by equation (2.11), equal to the long run time fraction a system in that state. With this for each source and using the PASTA property (section 2), two important variables can be determined:

- the fraction of the time a source is busy (by summing over all states that in which that source s is busy):

$$\text{time fraction busy source } s\text{: } \sum_{B\in C}\pi(B)1_{\{s\in B\}} \tag{3.13}$$

- the accepting probability by summing over all states to which the source s can be added and that state is still in the set of admissible states:

$$\text{accepting probability source } s\text{: } \sum_{B\in C}\pi(B)1_{\{s\notin B, B+s\in C\}} \tag{3.14}$$

3.2 Insensitivity

The above model is based upon exponentially distributed arrival and handling rates, see section 2.3. However, in real life the arrival and handling rates for train arrivals at railway stations are not necessary exponentially distributed. To be able to validate the use of the model in practical situations, it is useful to prove that the model has the *insensitivity property*, meaning that these assumptions cannot be loosened.

To prove the insensitivity, assume that every source is described by a two-station queue. Each status can either can be at station 1, and so being idle, $l_s = 1$ or at station, being busy $l_s = 2$. See the schematic overview in figure 3.1.

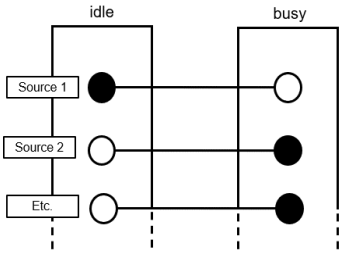

Figure 3.1: *Schematic overview of two-station idle-busy sources. Black dots indicated in which state a source is.*

Then the sources can be described by

$$B = \{s_1, ..., s_n\} \tag{3.15}$$

$$L = \{l_s; s = 1, ..., S\} \tag{3.16}$$

$$[L, A] = \{(l_s, a_s); s = 1, ..., S\} \tag{3.17}$$

Where B is the set of busy sources, as described already above. L is the set describing the state of each of the sources. a_s exponentially describes when the sources will become active (in case $_s = 1$) or when the source switches off (in case $l_s = 2$). This system of [L, A] can also be described by a Markov chain.

One can describe the time the first case (source is now idle and will become busy, l=1) and the second case (the time that the source is still busy until becoming idle, l=2) by

$$G_l^s = \sum_{m=1}^{\infty} q_l^s(m) E(m, \nu_l^s) \qquad (l = 1, 2) \tag{3.18}$$

Here, $q_l^s(m)$ describes the probability that the change of status takes m time phases. These phases are described by an $E(m, \nu_l^s)$ Erlang distribution, as is described in section 2.3. ν_l^s is the phase parameter for a source s in status l.

As in the latter, the expected mean of an (k, ν)-Erlang distribution is k/ν, the mean duration of each of the cases can be described by

$$\tau_l^s = \sum_{m=1}^{\infty} q_l^s(m) \frac{m}{\nu_l^s} \tag{3.19}$$

Let $H_l^s(a)$ be the steady-state probability that a source s is in state l for more than a time phases.

Theorem 3.2. *The solutions for the global balance equation for the $[L, A]$ Markov chain is given by*

$$H_l^s(a) = \frac{1}{\nu_l^s \tau_l^s} \sum_{k=a}^{\infty} q_l^s(k) \tag{3.20}$$

Proof. The rate into a state a is described by

$$H_l^s(a)\nu_l^s \tag{3.21}$$

while the rate out of the state is

$$H_l^s(a+1)\nu_l^s + H_l^s(1)\nu_l^s q_l^s(a) \tag{3.22}$$

where the first part considers the transition to $a+1$ and the second part the transition back to $a = 1$. The relation between the rate in and out is

$$H_l^s(a) = H_l^s(a+1) + H(1)q_l^s \tag{3.23}$$

which is satisfied by equation (3.20).
Furthermore, the sum of the solution equals to 1:

$$\sum_{a=1}^{\infty} H_l^s(a) = \frac{1}{\nu_l^s \tau_l^s} \sum_{a=1}^{\infty} \sum_{k=a}^{\infty} q(k) = \frac{1}{\tau_l^s} \sum_{k=1}^{\infty} \frac{k}{\nu_l^s} = 1 \tag{3.24}$$

□

Theorem 3.3. *The steady-state distribution of the Markov chain of the system $[L, A]$ under the invariance conditions (3.7-3.10), with normalizing constant $\bar{c}$ and with $P(B)$ as in (3.7).*

$$\pi(l_s, a_s) = \bar{c}P(B)\Big[\prod_{s \notin B} \tau_1^s H_1^s(a_s)\Big]\Big[\prod_{s \in B} \tau_2^s H_2^s(a_s)\Big] \qquad fors = 1, ..., S \tag{3.25}$$

Proof. To prove the equation above, it must be proven that for all sources in the $[L, A]$ Markov chain rate into a state of the source is equal to the rate out of this state. As the source in the Markov chain $[L, A]$ can only be either in state 1 or in state 2, and only two cases need to be considered, which are valid for all sources.
(i) Source h is idle
When the source is idle, the rate out is given by

$$\pi([L, A])\nu_1^s \tag{3.26}$$

while the rate into the idle state is the sum of the rates that the system moves one time interval further while source stays idle $[L, A] - (1, a) + (1, a+1)$ and that the source is not accepted to be become busy, but time interval start anyway $[L, A] - (1, a) + (1, 1)$.

$$\begin{aligned} \pi\big([L, A] - (1, a) + (1, a+1)\big)\nu_1^s + \\ \pi([L, A] - (1, a) + (1, a+1))[1 - A(s|B)]\nu_1^h q_1^s(a) \end{aligned} \tag{3.27}$$

Now note that by means of (3.20):

$$H_l^s(1) = 1/\nu_l^s \tau_l^s$$

that by (3.12), it is known that

$$\frac{P(B+s)}{P(B)} = A(s|B) \tag{3.28}$$

and that by (3.25),

$$\pi([L,A]) - (l_s, a_s)_s + (l'_s, a'_s)_s = \pi([L,A]) \frac{H_{l'_s}^s(a'_s)}{H_{l_s}^s(a_s)} \frac{\tau_{l'_s}^s}{\tau_{l_s}^s} \left[\frac{P(B+s)}{P(B)} \right]_s \tag{3.29}$$

Therefore, the rate into can also be written as:

$$\pi([L,A]) \left(H_1^s(a+1) + A(s|B) \frac{q_1^s(a)}{\tau_1^s \nu_1^s} + [a - A(s|B)] \frac{q_1^s(a)}{\tau_1^s \nu_1^s} \right) / H_1^s(a) \tag{3.30}$$

which satisfies the detailed balance equation if

$$H_l^s(a) = H_l^s(a+1) + \frac{q_l^s(a)}{\tau_l^s \nu_l^s} \tag{3.31}$$

which is true by (3.20).
(ii) Source s is busy The same way of proving can be applied here. It needs to be proven that the rate in

$$\pi([L,A])\nu_2^s \tag{3.32}$$

is equal to the rate out

$$\begin{aligned} &\pi([L,A]) - (2,a)_s + (2,a+1)_s)\nu_2^s \\ +&\pi([L,A]) - (2,a)_s + (1,1)_s)\nu_1^s A(s|B-s) q_2^s(a) \\ &= \pi([L,A])\nu_2^s \left(H_2^s(a+1) + \frac{q_2^s(a)}{\tau_2^s \nu_2^s} \right) / H_2^s(a) \end{aligned} \tag{3.33}$$

So the rate in is equal to the rate out of all states for all sources $s \in C$ by (3.20). □

Theorem 3.4 (Insensitivity of the model)**.** *Assuming the invariance conditions and P in (3.7 until 3.2) and with $\overline{c}$ and c as normalization constants, the product can be written as*

$$\pi(B) = \overline{c} P(B) \prod_s \tau_1^s \prod_s \tau_2^s = c P(B) \prod \frac{\lambda_s}{\mu_s} \tag{3.34}$$

Proof. See that

$$\sum_{s=1}^{S} \sum_{a_s=1}^{\infty} \prod_{s=1}^{S} H_l^s(a_s) = \prod_{s=1}^{S} \Big(\sum_{a_s=1}^{\infty} H_l^s(a_s) \Big) \tag{3.35}$$

With equations (3.35) and (3.24), the steady states can be described as

$$\begin{aligned}\pi(B) =& \sum_{s=1}^{S} \sum_{a_s=1}^{\infty} \pi(l_s, a_s) \text{ for } s = 1, ..., S \\ =& \overline{c} P(B) \Big[\prod_{s=1}^{S} \tau_l^s \Big] \Big[\prod_{s=1}^{S} \sum_{a_s=1}^{\infty} H_l^s(a_s) \Big] \\ =& \overline{c} P(B) \prod_{s=1}^{S} \tau_l^s \\ =& \overline{c} P(B) \prod_{s} \tau_1^s \prod_{s} \tau_2^s \end{aligned} \tag{3.36}$$

with $\lambda_s = \frac{1}{\tau_1^s}$, $\mu_s = \frac{1}{\tau_2^s}$ and $\overline{c} = c[\lambda_1 \lambda_2 ... \lambda_S]$, (3.34) is proven. □

By theorem 3.4, the model is insensitive for different distribution of handling times. Meaning that product-form results given in (3.8) are also valid for non-exponential distribution and only depends on the mean values.

3.3 Discussion

The main advantage of the model described above is that is it (easily) solvable. However, it does not display the reality. In the model, two main assumptions are made that are not met in reality.

Firstly, it is assumed that an arriving train uses this route for all time that the train is in the station. However, in reality, the train only uses the route upon arrival, then waits at the track next to its platform, before departing. During the wait, other trains can make use of the route the waiting train took before except from the track next to the platform.

Secondly, the model assumes no waiting room and let a train "vanish" if it is blocked to arrive at the station. Of course, in real life, this is not possibly and a train will have wait until they can enter the station. Therefore, a waiting room should be assumed.

Another point, not unimportant, is that only arrival are taken into account. Typically, in Dutch Railways network, tracks are used both ways and the delayed departures can make incoming arrivals delayed as well.

One could say to implement these parts to get a model that better fits reality, however implementing this will make the model more difficult and it might become unsolvable. Possible directions to improve on the model are given in the appendix part 11.

Part II
Simulation model

4 Explanation of the simulation model

In addition to the analytical model, also simulation model has been made. The two can be compared. In both cases, the same assumptions are made. But in the model, a product form is filled in, immediately showing the result. In the simulation, trains are randomly arriving at the station and the (non)blocking is tracked to see what happens throughout the time. The two can be compared to see what difference arise.

The simulation for this report is done in Python. Python suits well for Object Oriented programming and has multiple packages which make simulating easier. For this simulation , among others, the Simpy package is used. This package accommodates to simulate for a given amount of time and having multiple active functions at the same time, used to simulate the different sources.

The simulation is made in such way that it suits any station, given a range of input variables. In this way, it is easier to also evaluate other stations. The simulations assumes multiple sources that generate trains to arrive at the station, just as in the model. The input variables for the simulation are lists where each elements indicates a certain input variable for a sources. These are lists for arrival and handling rates, both expected to be exponentially distributed. Furthermore, a list for the serial numbers of a source and a list having a route indicating which tracks a train from a certain source uses to arrive at its platform. Next, also the total number of tracks in the station, number of sources, and the total time the simulation will simulate (the simulation itself will take much shorter than this given time) should be given.

In the simulation, each source generates randomly, by the given arrival rates, train objects. Upon generation, it is checked if the train can "arrive", that is if all tracks of the route are free. If so, the train takes in the space of the tracks and wait for an random time, by the handling rate. If not, the train is added to the block list and "vanishes" from the system. This is done for the indicated simulation time, after which the total blocking numbers are returned such that accepting/blocking rates can be calculated.

The code for the simulation function is given in the Appendix part 12.

Part III
Application

5 A simple station

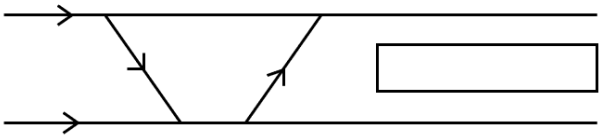

Figure 5.1: *A simple made-up station*

Think of a simple station with two platforms and four switches, as shown above in figure 5.1. Assume that there are four train lines arriving at this station, evaluated as four different sources in the model. This station can also be depicted as figure 5.2, showing the sources and the numbered tracks. Tracks are numbered between intersections, the numbering itself can be done randomly.

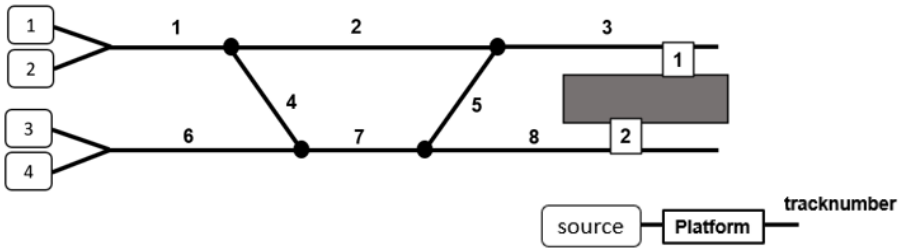

Figure 5.2: *Schematic overview of a simple station with sources and numbering of tracks.*

Each source takes a different route upon arriving at the station. The routes take the following line segments:

$$\begin{aligned} \text{source } 1 &: 1-2-3 \\ \text{source } 2 &: 1-4-7-8 \\ \text{source } 3 &: 6-7-5-3 \\ \text{source } 4 &: 6-7-8 \end{aligned} \tag{5.1}$$

5.1 Application of the analytical model

This simple station can be parameterized by the model proposed in section 3. Following (3.1), the access functions for the four sources are:

$$\begin{aligned} A(s_1|H) &= 1_{\{M_1=0, M_2=0, M_3=0\}} \\ A(s_2|H) &= 1_{\{M_1=0, M_4=0, M_7=0, M_8=0\}} \\ A(s_3|H) &= 1_{\{M_7=0, M_5=0, M_3=0\}} \\ A(s_4|H) &= 1_{\{M_6=0, M_7=0, M_8=0\}} \end{aligned} \tag{5.2}$$

and as only the source s_1 and s_4 can be generating at the same time, the state space C is given by:

$$C = \{\emptyset, \{s_1\}, \{s_2\}, \{s_3\}, \{s_4\}, \{s_1, s_4\}\} \tag{5.3}$$

From (3.8), the steady state probabilities for the simple station example become

$$\begin{aligned} \pi(\emptyset) &= c \\ \pi(\{s_1\}) &= c\frac{\lambda_{s_1}}{\mu_{s_1}} \\ \pi(\{s_2\}) &= c\frac{\lambda_{s_2}}{\mu_{s_2}} \\ \pi(\{s_3\}) &= c\frac{\lambda_{s_3}}{\mu_{s_3}} \\ \pi(\{s_4\}) &= c\frac{\lambda_{s_4}}{\mu_{s_4}} \\ \pi(\{s_1, s_4\}) &= c\frac{\lambda_{s_1}}{\mu_{s_1}}\frac{\lambda_{s_4}}{\mu_{s_4}} \end{aligned} \tag{5.4}$$

where $c = 1/(1 + \frac{\lambda_{s_1}}{\mu_{s_1}} + \frac{\lambda_{s_2}}{\mu_{s_2}} + \frac{\lambda_{s_3}}{\mu_{s_3}} + \frac{\lambda_{s_4}}{\mu_{s_4}} + \frac{\lambda_{s_1}}{\mu_{s_1}}\frac{\lambda_{s_4}}{\mu_{s_4}})$ to meet the normalization condition $\sum_{H \in C} \pi(H) = 1$.

These steady state distribution also describe the amount of the time the system is in a certain state, as is explained in section 2.1. This can be recalculated to the fraction of the time that a source is busy/idle. For sources s_2 and s_3, the fraction of time to be busy is equal to steady state distribution of the state there are busy. For sources s_1 and s_4, the fraction of time that both these sources are active can be added to the state there are alone active. Also, the acceptance probability for each source can be calculated, being the chance that a train originated from that source is acceptable.

For example, assume arrival times that follow a Poisson process, with means of 15, 20, 20 and 15 minutes and exponentially distributed handling times averaging 2, 1, 1.5 and 1 minutes for sources s_1, s_2, s_3 and s_4 respectively. Using (3.8), (3.13) and (3.14), the fraction of the time that a source is busy and the acceptance probability can be determined. The outcomes are shown in table 1. The fraction of the time that no trains are present in the system, and therefore also the general acceptance probability, is equal to $\pi(\emptyset) = 0.75$.

Table 1: *Input variables and corresponding results using the model for the simple station.*

source	arrival time	handling times	fraction busy	acceptance probability
1	15	2	0.107	0.80
2	20	1	0.037	0.75
3	20	1.5	0.056	0.75
4	15	1	0.056	0.85

5.2 Application of simulation model

With the simulation model in section 3, this simple station is also simulated. Interesting is to see if there are any differences between the model and simulation, and particularity whether the insensitivity also holds when using a simulation.

Assume arrival rates and handling rates as proposed in the last paragraph of the previous subsection for the sources s_1, s_2, s_3 and s_4 respectively. The total number of tracks is 8 and the routes that the trains follow in as in (5.1). For each cycle a total time of 1440 minutes is given, to match the number of minutes in a day. Although, the train schedules differ during night and weekends, this is not taken into account. This "day"-cycle is repeated for 100 times, such that a good representation is acquainted of the simulation results.

In tables 2 and 3, the results for both the analytical model as the simulation model are shown. For the simulation model, the mean value for each source is determined. In the first of the table, the time fraction that the sources are busy can be seen. To show statistically, that the mean of the simulation results is equal to the product form result, a students T-test is performed for each of the sources, with a null hypothesis for all sources that the mean of the simulation results is equal to the result of the product form with an alpha level of 5%. The results, shown in table 2, show that this null hypothesis cannot be rejected. The same is done for the acceptance probability, the results for this are shown in table 3. Here, also the null hypothesis that the model and mean of the simulation results are the same could in none of the sources be rejected.

Table 2: *Fraction of the time busy for model and mean of simulation results and P-value for students T-test to compare model and simulation results for the simple station with inputs as in table 1.*

source	Fraction of time busy analytic model	Fraction of time busy simulation model mean	p-value t-test
1	0.107	0.106	0.13
2	0.038	0.037	0.24
3	0.057	0.056	0.18
4	0.057	0.056	0.10

In figure 5.3, for each source an histogram for the 1000 simulation runs of the acceptance rate is shown. As can be seen, the acceptance rate varies highly between each simulation run, due to the randomization of the arrival and handling rate, but the mean is close to the product form result.

Table 3: *Results for the acceptance probability of the simulation for the simple station, p-values to compare the model and simulation results and to test the insensitivity property.*

source	acceptance prob. analytic model	acceptance prob. mean simulation model	acceptance prop. mean simulation model mixed Erlang handling	p-value	p-value insensitivy test
1	0.80	0.802	0.805	0.67	0.22
2	0.75	0.750	0.759	0.59	0.62
3	0.75	0.747	0.757	0.13	0.69
4	0.85	0.851	0.857	0.17	0.07

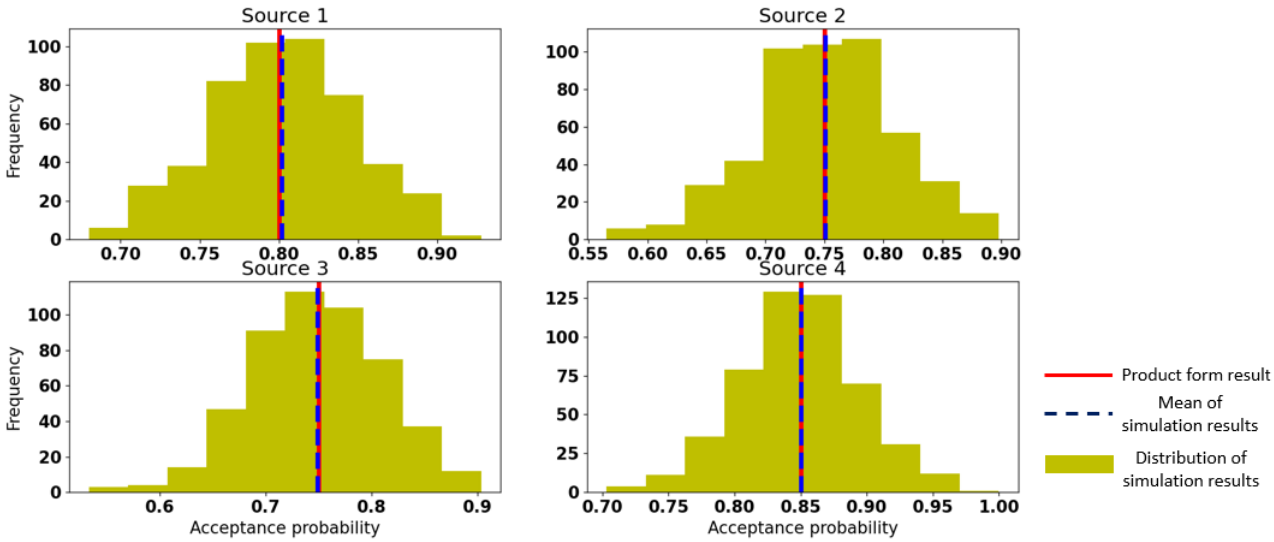

Figure 5.3: *Distribution of the acceptance probability for each of the sources in the simple station with input variable as in table 1.*

Insensitivity

The analytical model is build upon exponentially distributed inter-arrival and handling times. However, in section 3.2 it was proven that the analytical model has the insensitivity property, and therefore handling times that follow any mixture of Erlang distributions (see section 2.3) but with the same mean value should attain the same result.

In the following example, the exponentially distributed handling times in the simulation model are changed into a mixture of Erlang distribution with random number of phases (between 1 and 10). This results in the acceptance probability given in table 3. With a student T-test it tested if the distribution of the initial simulation results are statistically in line with to the distribution of the mixed Erlang handling times simulation.

Indication of busy places

It can be stated that the (easily to be calculated) product form acquaints the same results as (more difficult to perform and longer taking) simulation. However, a reason why one would still implement a simulation is to see at which tracks of the station the congestion occurs the most, such that other routing strategies can more easily be determined.

When looking where the trains are blocked, the simulation can also calculate the average blocking probability for each individual track, being the number of trains that wanted to ride that track but where blocked divided by the total number of trains generated that wanted track. An overview of the mean blocking rates of each of the tracks is given in table 4. It can be concluded that on tracks 1, 3 and 7 occur the most blockages as can be seen in figure 5.4.

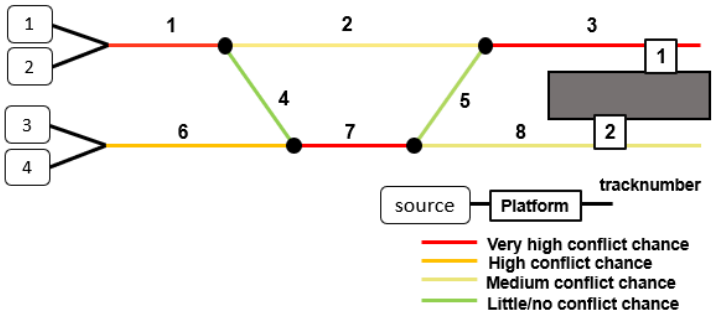

Figure 5.4: *Indication of tracks where blockage occurs the most)*

Table 4: *Blocking rates and decrease in utilization from a theoretical no block scenario, per track for the simple station. Color indicate high (red) to low (green) blocking rates and decreased percentage*

track	blocking rate	theoretical utilization	percentage decreased utilization
1	0.156	0.183	22.3
2	0.116	0.133	21.0
3	0.168	0.208	13.8
4	0.045	0.05	26.0
5	0.065	0.075	1.3
6	0.122	0.147	8.5
7	0.158	0.192	13.1
8	0.104	0.117	20.6

In the same table 4, the theoretical utilization and the decrease in utilization after the blocking model(s) are applied are shown. The theoretical utilization is the time fraction that a track will be occupied in case that no blocking. The other value shown in the table is the percentage is the decrease from this theoretic utilization to the utilization from the analytical and simulation model. It shows that an high blocking rate, meaning that many trains are blocked because

that track is already occupied, does not necessary means that the utilization of that track has also decreased the most. This is because the number of trains that make use of the tracks and how fast they travel over that track (here determined by the handling time). The indication of the most busy tracks and the decrease of utilization can be used to redesign the route that trains take trough a station. To give a good indication of what a better rerouting would be, it is important to take into account which train line and so tracks have preference to be less blocked and/or a less decrease in utilization. An example how to determine the qualities for a possible rerouting is given in the section of the evaluation of Amersfoort C.

Conclusion

To sum up, the analytical model can give first insight in the probability of conflicts at a station. The, more detailed, simulation model results are statistically verified as by the analytic product form results. The same applies for the insensitivity statements. With the simulation model, an indication of the busy tracks can be made. Later in this report, for the more difficult railway station of Amersfoort C., this will be used to determine possible rerouting through the station.

6 Arnhem Centraal

Railway station Arnhem Centraal is a relative big sized railway station in the Netherlands. Build in middle of the nineteenth century, the last major, 20 year during, reconstruction was finished in 2015, to accommodate for more passengers and more trains. After this, the station attained the predicate "Centraal" as one of the eight stations in the Netherlands [1].

A simplified overview including arrival routes can be seen in figure 6.1. This is based upon the map designed by [18]. In the appendix, table 17, an overview of the arrival times to the station during a typical weekday is shown.

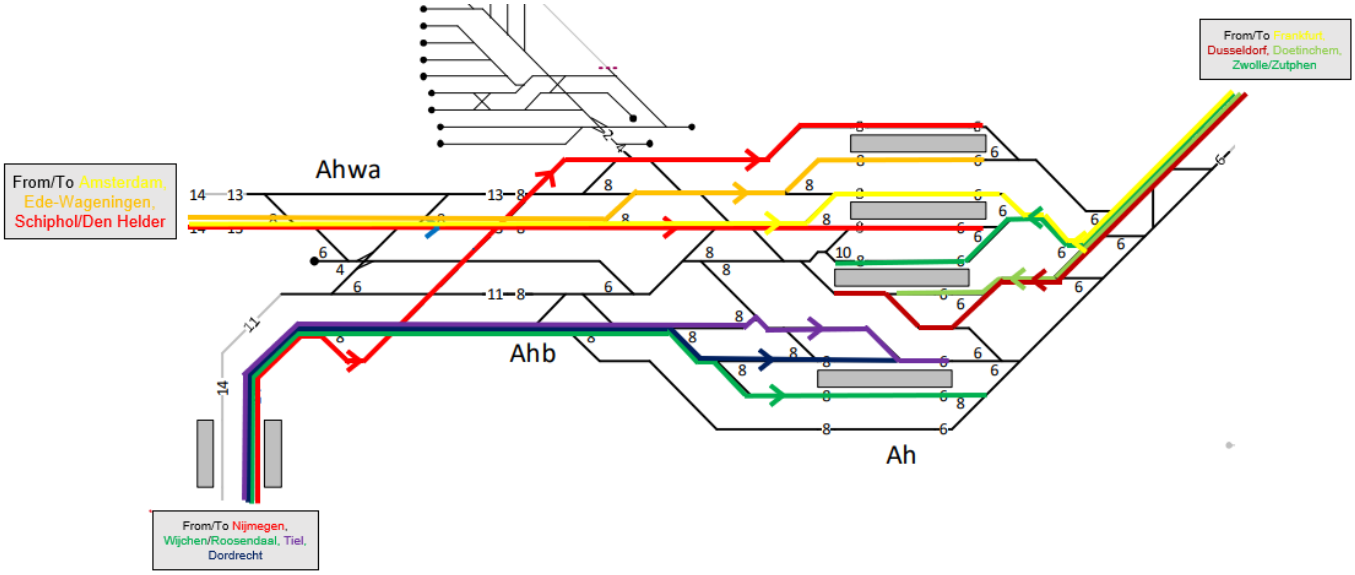

Figure 6.1: *Arrival routes to Arnhem Centraal station based upon [18].*

6.1 Data Analysis

The data regarding all real time trains is public available via an API provided by the NS. The people behind OVData have been saving this data since 2013 publicly, however they only start taking into account arrival times since 2018. As the data from 2020 are modified due to the COVID pandemic, it is chosen to evaluated all data over 2018 and 2019.

Arrival delay

The NS only denotes a train to be officially delayed at a delay of 5 minutes or higher and so does not registered for any smaller delays nor what the cause is. Therefore, it is hard to analyze with the data what the fraction of secondary delays is, the analytical model and simulation model can give answer to this. However, what can be analyzed is the arrival delay of all trains at Arnhem C. Taking into account all arrivals at Arnhem centraal in 2018 and 2019, the mean arrival delay is 56 seconds with a standard deviation of 184 seconds. However, the arrival delay depends highly on the train line. The mean arrival delay split for the different lines arriving at Arnhem C. are given in table 5. As the line 3200 is not recorded in the OVDATA, but is considered to be under line 3100, therefore, these numbers are the same.

Table 5: *Mean arrival delay in seconds for lines arriving at Arnhem C., including and excluding arriving trains with no delay, showing the number of trains on time. The names of the abbreviations can be found in the table 16.*

Line number	Origin	Mean	Mean excl. zeros	Non zero delay fraction
3000	NM	50	124	0.402
3000	HDR	50	121	0.413
3100	NM	52	131	0.398
3100	SHL	54	92	0.592
3600	RSD	52	52	0.267
3600	ZL	36	36	0.346
7600	WC	47	113	0.416
7600	ZP	31	114	0.271
6600		157	330	0.476
31100		101	231	0.438
7500		100	150	0.67
30700		85	218	0.391
20000		352	815	0.432
100	AM	175	276	0.633
100	FFFM	280	484	0.578

Handling times

To be able to use both models, the expected handling time for each line entering Arnhem must also be determined. With the data given, this is done for the lines that did not have Arnhem as the final station, as this made it able to determine

the time between arrival and departure. The values for the handling times for the different handling times can be seen in table 6.

Table 6: *Mean handling times for the train arriving at Arnhem C. in seconds and in minutes per different lines and origins. Meaning for the origin abbreviations can be found in appendix section 9.*

Line number	Origin	Mean (s)	Mean (min)
3000	NM	348	5.8
3000	HDR	296	4.9
3100	NM	327	5.5
3100	SHL	362	6.0
3600	RSD	278	4.6
3600	ZL	210	3.5
7600	WC	328	5.5
7600	ZP	244	4.1
100	AM	129.6	2.2
100	FFFM	130.4	2.2

Lines 6600, 31100, 30900, 7500, 30700 and 20000 could not be analysed as these have Arnhem as final station.

6.2 Application of the analytical model

As can be seen in figure 6.1, every line arriving at Arnhem has its own route and platform. The model does not assume scheduling, meaning that it does not take into account rescheduled arriving times. Regularly, 17 different train lines arrive in Arnhem and so 17 sources are assumed in the model.

For every source a fixed route through the station is determined, as also shown in table 7. In figure 6.2, the numbering of the tracks between infra-elements is given.

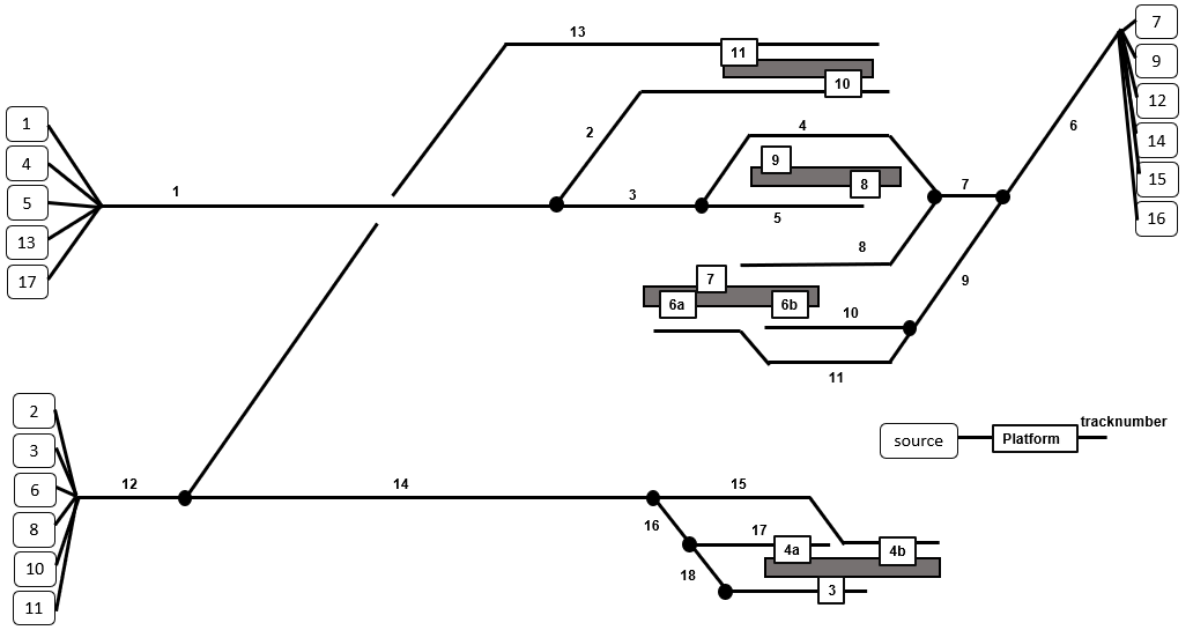

Figure 6.2: *Abstract view of Arnhem C. with numbered arrival tracks and platform. Tracks that not used are left out.*

The set of possible combinations of sources has a size of $\sum_{k=0}^{n} \binom{17}{k} = 2^{17} = 131072$ and will therefore not be given here. Based upon figure 6.1, the sources can be grouped by direction.

Direction A: $s_1, s_3, s_6, s_8 s_{11}, s_{13}$

Direction B: $s_2, s_4, s_5, s_{10}, s_{15}, s_{17}$

Direction C: $s_7, s_9, s_{12}, s_{14}, s_{16}$

Two sources from the same direction cannot be busy at the same time. Furthermore, s_{16} and s_{17} use the same platform and cannot be busy at the same time. Therefore, the total set of admissible states has a size of $2 \times 6 + 1 \times 5 + 6 \times 6 + 2 \times 5 \times 6 + 6^3 + 1 - 1 = 293$. In the table 7, the numbering of sources is given with the determined handling times. For the arrival times, the expected difference between two following trains is used. For the turning trains, a handling of 3 minutes is used, as it cannot be determined how long

these trains stay at the station, so only the time to arrive until the platform is taken into account.

Table 7: *Overview of sources with arrival and handling rates in trains per minute.*

source nr	from	serial number	arrival times	handling times	route
1	Nijmegen	3000	30	5.3	12-13
2	Den Helder	3000	30	5.8	1-3-5
3	Nijmegen	3100	30	5.5	12-13
4	Schiphol Airport	3100	30	6	1-3-5
5	Rotterdam C.	3200	30	6	1-3-4
6	Roosendaal	3600	30	4.6	12-14-16-18
7	Zwolle	3600	30	3.5	6-7-8
8	Wijchen	7600	30	5.5	12-14-16-18
9	Zutphen	7600	30	4.1	6-7-8
10	Dordrecht	6600	30	3	12-14-16-17
11	Tiel	31100	30	3	12-14-15
12	Winterswijk	30900	30	3	6-9-10
13	Ede-Wagingen	7500	30	3	1-2
14	Doetinchem	30700	30	3	6-9-10
15	Dusseldorf	20000	60	3	6-9-11
16	Frankfurt	100	60	2.2	6-7-4
17	Amsterdam C.	100	60	2.2	1-3-4

Knowing the admissible states and the arrival and handling rates, the steady state distribution can be calculated using equation 3.8. Due to the size of set C, this is done using a self-written Python code shown in the Appendix section 12. The steady state distribution can now be used to show the amount of time that each of the sources is busy, these result in the numbers shown in table 8.

Taking into account the amount of train each source generates hourly, the general acceptance probability can be calculated, being 46%. Here it is not taken into that the train schedule differs during night and during the weekends.

6.3 Application of the simulation model

With the simulation made, only the input variables of the arrival and handling times, see table 7, the total number of sources (17) and the number of tracks and routes the trains. The latter can be seen in table 7.

The simulation with these entries was first run for 1000 times, each simulating about 8 hours. The mean of the results are taken and shown in table 8. The simulation results are in most cases higher than the model results, except from source 16.

Scheduling

As in the model, the initial simulation does not take into account the current scheduling, rather than only the time between arrivals. Interesting is to see if the same results are acquired when taking the scheduling into account. This is

Table 8: *Results of the model and the simulation for Arnhem C. with the input variables given in table 7.*

source nr	origin	serial number	fraction busy model	fraction busy simulation mean	accept prob. model	accept prob. simulation mean
1	NM	3000	0.068	0.076	0.423	0.426
2	HDR	3000	0.082	0.098	0.470	0.514
3	NM	3100	0.070	0.077	0.423	0.425
4	SHL	3100	0.085	0.101	0.470	0.512
5	RTD	3200	0.085	0.100	0.470	0.508
6	RSD	3600	0.059	0.064	0.423	0.422
7	ZL	3600	0.054	0.054	0.511	0.478
8	WC	7600	0.070	0.077	0.423	0.432
9	ZP	7600	0.063	0.065	0.470	0.476
10	DDR	6600	0.142	0.142	0.423	0.435
11	TI	31100	0.128	0.141	0.423	0.427
12	WW	30900	0.155	0.157	0.511	0.473
13	ED	7500	0.128	0.171	0.423	0.518
14	DTC	30700	0.155	0.158	0.511	0.476
15	QDU	20000	0.071	0.080	0.470	0.471
16	FFFM	100	0.017	0.015	0.507	0.423
17	ASD	100	0.016	0.018	0.466	0.507

done by modifying the simulation slightly. For each source, the generation starts at a different moment, given by the schedule as can be seen in table 17. The new code can be seen in the appendix, 12. After 1000 simulations, the simulation with scheduling returns the results given in table 8. The null hypothesis that the mean values of the outcome with scheduling and without are the same could not be rejected for all sources.

Insensitively

As is proven in section 3.2, the model made is also valid for non-exponentially distributed handling rates. The simulation for Arnhem C. is also run for mixed Erlang distributed (see section 2.17) handling rates. The statistically same time fraction that sources are busy and acceptance probabilities are found with this input.

Indication of conflicting tracks

A simulation, unlike the product form, makes it possible to get an indication where in the station the conflicts takes place. For the input variables used above, this results in figure 6.3 and table 9. Especially the entrance tracks, 1, 6 and 12 do have a high blocking rate, as trains use this track quickly after one and other.

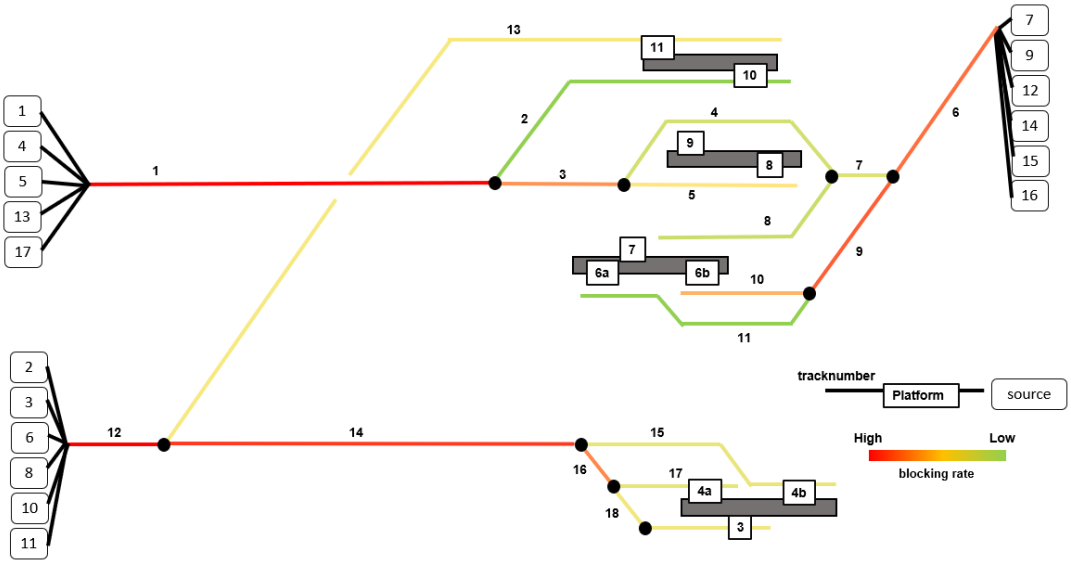

Figure 6.3: *Indication of conflict chance for different tracks in Arnhem C.*

Table 9: *Blocking rates and decreased utilization compared to a no blocking scenario, per track for Arnhem C., colors indicate high blocking rate / percentage (red) to low blocking rate /percentage (green).*

track	blocking rate	percentage decreased utilization
1	0.487	-58.9
2	0.237	-61.6
3	0.382	-57.5
4	0.21	-57.3
5	0.275	-57.5
6	0.431	-54.4
7	0.223	-53.8
8	0.197	-53.8
9	0.453	-54.3
10	0.34	-53.5
11	0.127	-57.4
12	0.575	-60.6
13	0.255	-61.7
14	0.499	-60.2
15	0.239	-61.6
16	0.397	-59.6
17	0.235	-57.4
18	0.249	-61.7

7 Amersfoort Centraal

Build in the middle of the nineteenth century, railway station Amersfoort also received the predicate "Centraal" in 2019[19]. The station has about 40.000 boarding/getting off a train each year[20], being one of the main station that connect the West of the Netherlands (the Randstad) with the North-East.

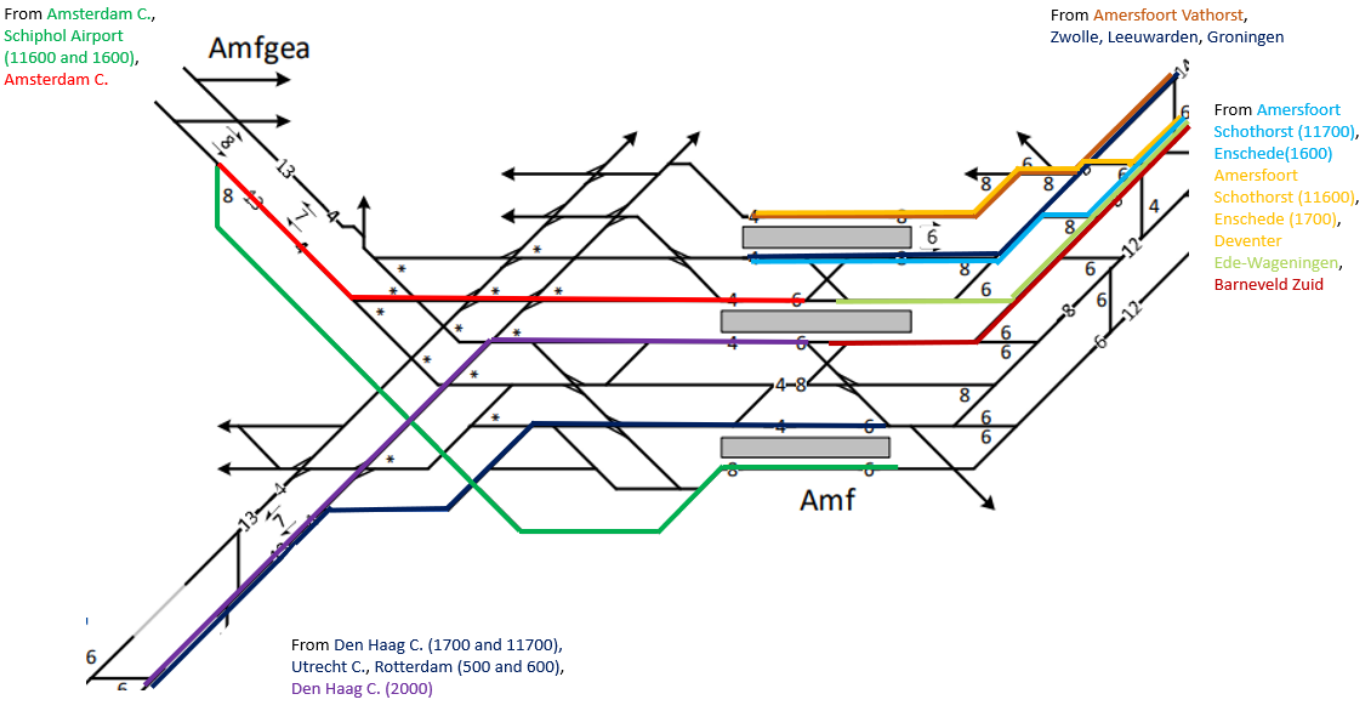

Figure 7.1: *Overview of arrival routes at Amersfoort Centraal*

A simplified overview that includes arrival routes can be seen in figure 7.1. In the appendix, table 18, an overview of the arrival times to the station during a typical weekday is shown.

7.1 Data analysis

Just as for Arnhem C., a data analysis was performed to find the handling rate for the different arriving lines at Amersfoort C. For this data of 2018 and 2019 is used, provided by OVData. Also, the mean arrival delay is analysed.

Arrival Delay

The Dutch Railways does not registered the type of delay if this is caused by other delayed trains. However, to get a view of the mean arrival delay of arriving trains at Amersfoort C., this is still analysed. On average, an arriving trains at Amersfoort C. has a delay of only 45 seconds. Taking only the non-zero delay,

Table 10: *Mean arrival delay in seconds for lines arriving at Amersfoort C., including and excluding arriving trains without delay and showing the fraction of trains arriving without delay. The names of the railway stations can be found in table 9.*

serial number	origin	mean arrival delay	mean arrival delay excl. zeros	fraction zeros
15800	ASD	29	112	0.263
	AVAT	38	105	0.357
11700	AMFS	17	119	0.143
	GVC	24	114	0.213
11600	AMFS	15	126	0.120
	SHL	25	121	0.207
1700	ES	54	147	0.371
	GVC	31	140	0.226
1600	ES	59	162	0.361
	SHL	40	157	0.254
5600	UT	34	136	0.250
	ZL	57	156	0.365
1500	ASD	58	178	0.327
	DV	52	171	0.302
600	RTD	31	153	0.205
	LW	53	160	0.330
500	RTD	33	116	0.196
	GN	59	172	0.340
200	ASD	111	231	0.481
	BHF	366	814	0.450
31300	ED	120	287	0.417
31400	DTC	129	406	0.316
2000	GVC	89	242	0.366

this increases to an average of 147 seconds, so about 2.5 minutes. In table 18, an overview of the arrival delay including and excluding no delays is shown.

Handling times

For all non-turning lines, the mean handling time is determined with data provided by the OV-loket for the years 2018-2019.

Table 11: *Mean handling times for the non-turning lines between arrival and departure from Amersfoort C. in seconds and minutes.*

Serial number	Origin	Mean handling (s)	Mean handling (min)
15800	AVAT	71	1.2
15800	ASD	76	1.3
11700	GVC	184	3.1
11700	AMFS	134	2.2
11600	AMFS	136	2.3
11600	SHL	240	4.0
1700	ES	126	2.1
1700	GVC	138	2.3
1600	ES	127	2.1
1600	SHL	133	2.2
5600	UT	134	2.2
5600	ZL	124	2.1
1500	DV	128	2.1
1500	ASD	130	2.2
500	RTD	180	3.0
500	GN	161	2.7
600	RTD	159	2.7
600	LW	178	3.0
200	ASD	108	1.8
200	BHF	130	2.2

7.2 Application of the analytical model

As is shown in table 12, one can assume 23 sources to generate trains arriving at Amersfoort C. For the handling times of the turning trains (those with final destination Amersfoort C.) are assumed to have an exponentially distributed handling time with mean of 10 minutes. In comparison with Arnhem C., Amersfoort C. has less platforms while more lines arriving at the station. Therefore, more lines make use of the same infra-elements and platforms than in the first. An overview of the numbering of tracks can be seen in figure 7.2.

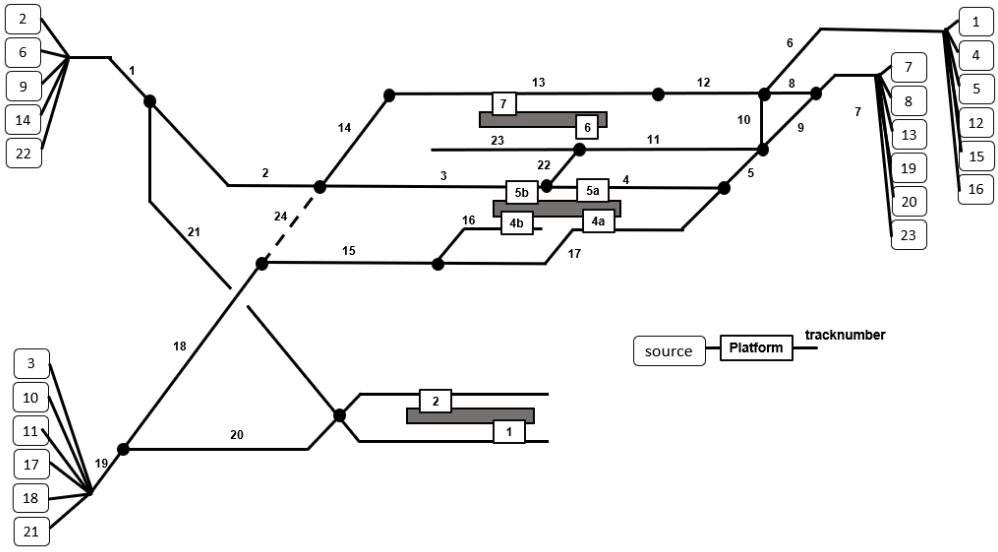

Figure 7.2: *Abstract view of Amersfoort C. with number arrival tracks and platforms.*

Track 24 is later added when looking for possible rerouting options.

For Amersfoort C. the total set of possible busy sources has a size of $2^{23} = 8388608$, and will therefore not be given here. One can define four direction from which trains arrives and so the sources can be grouped in. From each direction, two of more sources cannot be active at the same moment.

Direction A: $s_2, s_6, s_9, s_{14}, s_{22}$

Direction B: $s_3, s_{10}, s_{11}, s_{17}, s_{18}, s_{21}$

Direction C: $s_1, s_4, s_5, s_{12}, s_{15}, s_{16}$

Direction D: $s_7, s_8, s_{13}, s_{19}, s_{20}, s_{23}$

Due to the shared use of platform of trains from different directions, also for the following groups no two or more sources can be active at the same time.

Platform 6: $s_1, s_5, s_7, s_{13}, s_{22}, s_{23}$

Platform 7: $s_4, s_8, s_{12}, s_{15}, s_{16}$

Table 12: *Sources as in the analytical model and simulation model for Amersfoort C. with arrival and handling rates and route through the station.*

Sources	from	serial number	arrival rate	handling rate	platform	routes
1	AVAT	15800	0.033	0.843	7	6,12,13
2	ASD	15800	0.033	0.786	5a	1,2,3
3	GVC	11700	0.017	0.326	2	19,20
4	AMFS	11700	0.017	0.448	6	6,10,11
5	AMFS	11600	0.017	0.442	7	6,12,13
6	SHL	11600	0.017	0.250	1	1,21
7	ESD	1700	0.017	0.477	7	7,8,12,13
8	EC	1600	0.017	0.471	6	7,9,11
9	SHL	1600	0.017	0.451	1	1,21
10	GVC	1700	0.017	0.434	2	19,20
11	UR	5600	0.033	0.447	2	19,20
12	ZL	5600	0.033	0.484	6	6,10,11
13	DEV	1500	0.033	0.470	7	7,8,12,13
14	ASD	1500	0.033	0.461	1	1,21
15	LW	600	0.017	0.338	6	6,10,11
16	GN	500	0.017	0.373	6	6,10,12
17	RTD	500	0.017	0.373	2	19,20
18	RTD	600	0.017	0.377	2	19,20
19	ED	31300	0.033	0.100	5b	7,9,5,4
20	BNZ	31400	0.033	0.100	4b	7,9,5,17
21	GVC	2000	0.033	0.100	4a	19,18,15,16
22	ASD	200	0.008	0.556	7	1,2,14,13
23	BHF	200	0.008	0.460	7	7,8,12,13

Therefore, the total admissible set has a size of 1633. Knowing this, and the arrival and handling rates, the steady state distributions of all admissible states can be computed. To do so, the Python code in Appendix 12 is written and used.

The results can be seen in table 13. The acceptance probability for the different lines is between 50% and 85%. Taking into account the number of trains that each source generates on average, assuming the same schedule all day, the general acceptance probability is 69%.

7.3 Application of the simulation model

With the simulation model function made as given in appendix 12, Amersfoort C. can be simulated. Assume the arrival and handling rate and routes as given in table 12. The total number of tracks, as in figure 7.2, is 23. The simulation model is run for 1000 times, each times simulating about 8 hours. The mean fraction of the time that each source is busy and corresponding acceptance probability is given in table 13. The simulation model results differ only slightly from the product form outcome. Performing a Student's t-test with null hypothesis that the mean of the analytical model is the same as the result from the model, this hypothesis for most sources for both the fraction of the time busy as the acceptance probability could not be rejected.

Table 13: *Analytical model and simulation model results for Amersfoort C. with the input variables as in table 12.*

Sources	name	fraction busy analytical model	fraction busy simulation model	accept prob. analytical model	accept prob. simulation model
1	AVAT	0.029	0.028	0.755	0.738
2	ASD	0.034	0.035	0.810	0.820
3	GVC	0.032	0.031	0.630	0.634
4	AMFS	0.029	0.029	0.778	0.790
5	AMFS	0.029	0.027	0.755	0.727
6	SHL	0.054	0.054	0.811	0.818
7	ESD	0.019	0.017	0.529	0.501
8	EC	0.018	0.017	0.506	0.478
9	SHL	0.03	0.03	0.811	0.807
10	GVC	0.024	0.024	0.629	0.621
11	UR	0.047	0.047	0.629	0.631
12	ZL	0.053	0.054	0.778	0.767
13	DEV	0.038	0.036	0.529	0.502
14	ASD	0.059	0.058	0.811	0.817
15	LW	0.038	0.038	0.778	0.771
16	GN	0.035	0.032	0.778	0.741
17	RTD	0.028	0.028	0.629	0.630
18	RTD	0.028	0.028	0.629	0.620
19	ED	0.183	0.189	0.550	0.563
20	BNZ	0.183	0.182	0.550	0.544
21	GVC	0.211	0.212	0.629	0.634
22	ASD	0.011	0.011	0.766	0.721
23	BHF	0.009	0.009	0.529	0.481

7.4 Indication of conflicting tracks and rerouting of lines

With the simulation model, unlike with the product form, an indication of the tracks where the conflicts occur can be made. In figure 7.2, an overview of the fraction of the trains that is blocked at that tracks. Red indicates a high blocking rate, while green indicates little to no blocking. In table 15, the mean accepting rates are also given. Mainly tracks 5, 9 and 7 and 19 and 20 are highly used and have a high blocking rate. Also, the decrease in utilization of the tracks, compared to a no blocking system is given in this table. Here, often track with a high blocking rate also have a higher decreased utilization. But this does not necessary has to be so.

With the simulation model made, it can easily be seen if a rerouting of trains through the station can have benefits.

For example, the routes for the following sources can be changed: source 1: 6-10-11-23, source 3: 19-18-24-14-13, source 8: 7-8-12-13, source 19: 7-9-11-22-3. In this case, track 24 has been added to the system, as can be seen as the dashed line in figure 7.2. With this it is tried to move the decrease the number of trains over tracks 5,9 and 7 and 20. In the third column of table 14, the new blocking rate on the differences tracks has been shown. The rate for track 5 has been considerably lowered. However, the acceptance probability is, on

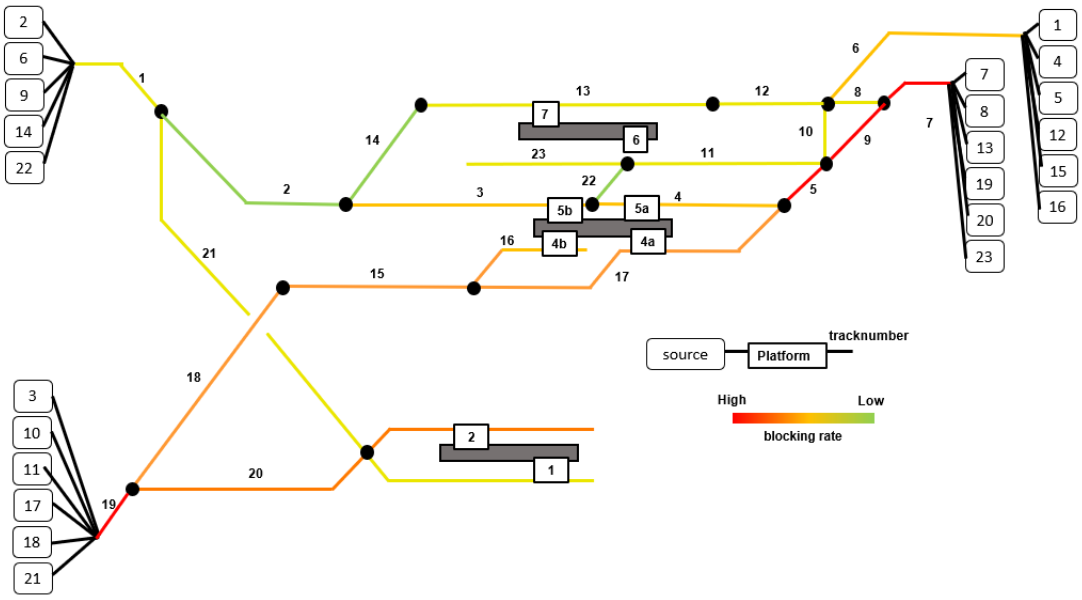

Figure 7.3: *Indication of the conflict chance for different tracks in Amersfoort C.*

average for all sources, lower with this set of changes to the routes through the station. Also, on average the utilization has even further decreased. Therefore, this rerouting scheme will not be beneficial.

With the analytical model and simulation model it becomes more easy to express if a certain redesign of the routes be advantageous. However, before this can be applied, first further research to get an indication of the prioritized trains should be made, being the highest number of travelers, most beneficial train line, usable platforms for a train line etc. Then possible redesigns of the routes can be considered.

Table 14: *Indication of blocking rate for each track for Amersfoort C. with input of table 7. The color ranges from green, indicating little to no blocking, to red, very high blocking rate.*

track number	blocking rate initial	blocking rate with rerouting	percentage decreased utilization	percentage decreased utilization after rerouting
1	0.191	0.193	-19.4	-19.4
2	0.054	0.053	-21.6	-21.6
3	0.25	0.235	-19.8	-19.8
4	0.24	0	-45.1	-45.1
5	0.396	0.244	-45.1	-45.1
6	0.217	0.215	-23.2	-23.2
7	0.455	0.446	-45.5	-45.5
8	0.108	0.135	-46.2	-47.3
9	0.414	0.396	-45.3	-45.1
10	0.164	0.191	-22.6	-23.2
11	0.19	0.33	-26.5	-36.0
12	0.161	0.162	-41.9	-47.3
13	0.172	0.205	-37.4	-40.7
14	0.016	0.056	-26.6	-34.9
15	0.247	0.245	-36.7	-36.7
16	0.247	0.245	-36.7	-36.7
17	0.247	0.244	-45.1	-45.1
18	0.247	0.279	-36.7	-36.8
19	0.369	0.369	-36.9	-36.9
20	0.2	0.167	-37.1	-37.1
21	0.147	0.149	-18.8	-19.0
22	0	0.244	-	-45.1
23	0.19	0.191	-26.5	-23.2
24	-	0.044	-	-37.3

Table 15: *Acceptance probabilities before and after the rerouting. The orange cells indicate a change in route. The red cells indicate a lower acceptance probability, green cells a higher acceptance probability.*

Sources	name	routes	different routes	accept probability old	accept probability new
1	AVAT	6,12,13	6,10,11,23	0.755	0.723
2	ASD	1,2,3	1,2,3	0.810	0.807
3	GVC	19,20	19,18,24,14,13	0.630	0.577
4	AMFS	6,10,11	6,10,11,23	0.778	0.723
5	AMFS	6,12,13	6,12,13	0.755	0.747
6	SHL	1,21	1,21	0.811	0.807
7	ESD	7,8,12,13	7,8,12,13	0.529	0.536
8	EC	7,9,11	7,8,12,13	0.506	0.536
9	SHL	1,21	1,21	0.811	0.807
10	GVC	19,20	19,20	0.629	0.624
11	UR	19,20	19,20	0.629	0.624
12	ZL	6,10,11	6,10,11,23	0.778	0.723
13	DEV	7,8,12,13	7,8,12,13	0.529	0.536
14	ASD	1,21	1,21	0.811	0.807
15	LW	6,10,11	6,10,11,23	0.778	0.723
16	GN	6,10,12	6,10,11,23	0.778	0.723
17	RTD	19,20	19,20	0.629	0.624
18	RTD	19,20	19,20	0.629	0.624
19	ED	7,9,5,4	7,9,11,22,3	0.550	0.505
20	BNZ	7,9,5,17	7,9,5,17	0.550	0.559
21	GVC	19,18,15,16	19,18,15,16	0.629	0.624
22	ASD	1,2,14,13	1,2,14,13	0.766	0.751
23	BHF	7,8,12,13	7,8,12,13	0.529	0.536

8 Conclusion and Discussion

The Dutch Railway network is a busy and complex system of trains, station and infra elements. It occurs that trains are blocked and have to wait to arrive at a station due to other delayed trains. The actual amount that this happens is not registered. To get better insight in the probability that such conflicts happens, both a mathematical model and a simulation model are proposed in this report. The mathematical model for trains entering a railway station, is based upon queuing theory for communication systems. This results in a so-called product form expression for the steady state distribution as is given in (3.8). The corresponding simulation mode for the arrival of trains at a railway station is made, written in Python. Both models can be applied to any station, given the right input variables.

It was shown that this analytical model and simulation model attain similar results. Also, insensitivity was proven for the mathematical model and was also tested and approved for the simulation model. This makes both more accurate for real-life purposes.

The analytical and simulation models were first applied to a simple fictitious station. Next, also two real-life Dutch railway stations are evaluated: Arnhem Centraal and Amersfoort Centraal. For the last two, a data-analysis was performed to attain the real-life waiting time at platforms to be used in both model. Also, with this data analysis it was found that trains arriving at Arnhem C. and Amersfoort C. have on average a delay of 56 seconds and 45 seconds, respectively. However, it is unknown which part of these delays are secondary delays (caused by other delayed trains), as the Dutch Railways does not register this.

Applying the analytical model and simulation model to the two stations, an accepting probability, the chance that a train can arrive at the station without being blocked by other trains, is 40-55% and 50-85% for Arnhem C. and Amersfoort C. respectively. However, this probability highly differs between the different train lines arriving at the two stations. Detailed results can be found in tables 8 and 13. It is noted that international trains are not taken into account here.

With the analytical model and the simulation model, an indication of the busy tracks in a station can be made, where the most blockage of trains occurs. Next, an indication of the amount of utilization that is lost due to the blocking has been made for the three stations. Both factors can help to create different routes through a station, such that the acceptance probability can be increased. This was tried for a couple of new routes for Amersfoort C., but a better result was not attained. To improve on this, more combinations of new routes should be tested. Importantly, it was noted that in choosing other routes a lot of factors should be considered. It is hard to increase all acceptance rates and at the same time satisfy all requirements and wishes.

The analytical model and simulation model can give a good indication of the accepting rate for the different trains arriving at stations. However, its makes certain assumptions which are generally not true in real-life. An arriving train is assumed to block all tracks it uses in the station (from entry until platform), in real life a train only rides on the first tracks and then takes the space of the tracks next to the platform. In the mean time, other tracks can be used again. For the proposed model, this results in an overestimation of reality. Second, in both models, when a train is blocked, it vanishes from the system.

In reality, the train will wait until the route to the platform is free or a different platform is chosen. This mean that either a waiting room in the models should be created, and/or a routing strategy to a different, free, platform should be applied. Lastly, departures from the station are not taken into account. In the models, after handling a train, it is removed from the system without using any tracks. When departing, often the same tracks are used, which can result in more delay propagation, meaning that the models can also underestimate reality.

To better suits for this assumptions, the model should be improved. In section 11 in the appendix, possibilities for further research are given. These are based upon the model proposed in this report, but loose some of the assumptions made. However, whether the proposed direction is solvable needs to be researched.

References

[1] J. Aalberts and E. Tol, "Modal choice criteria in rail transport Assessment of modal choice criteria in various rail transport market segments," CE Delft, Tech. Rep., 2018. [Online]. Available: www.cedelft.eu.

[2] D. Roßler, J. Reisch, F. Hauck, and N. Kliewer, "Discerning Primary and Secondary Delays in Railway Networks using Explainable AI," *Transportation Research Procedia*, vol. 52, pp. 171–178, 2021, ISSN: 23521465. DOI: 10.1016/J.TRPRO.2021.01.018. [Online]. Available: www.sciencedirect.comwww.sciencedirect.comwww.elsevier.com/locate/procedia.

[3] NS, *NS Jaarverslag 2019 - Punctualiteit op hoofdrailnet*, 2019. [Online]. Available: https://2019.nsjaarverslag.nl/FbContent.ashx/pub_1000/downloads/v200227115042/NS-Jaarverslag-2019.pdf.

[4] ——, *NS Jaarverslag 2013 - Reizigerspunctualiteit blijft achter bij norm*, 2013. [Online]. Available: https://2015.nsjaarverslag.nl/FbContent.ashx/pub_1000/downloads/NS_Jaarverslag2013.pdf.

[5] T. Lindner, "Applicability of the analytical UIC Code 406 compression method for evaluating line and station capacity," *Journal of Rail Transport Planning & Management*, vol. 1, no. 1, pp. 49–57, Nov. 2011, ISSN: 2210-9706. DOI: 10.1016/J.JRTPM.2011.09.002.

[6] A. Lindfeldt and Kungl. Tekniska högskolan. Skolan för arkitektur och samhällsbyggnad., *Railway capacity analysis : methods for simulation and evaluation of timetables, delays and infrastructure*, ISBN: 9789187353659.

[7] T. Huisman and R. J. Boucherie, "Running times on railway sections with heterogeneous train traffic," *Transportation Research Part B: Methodological*, vol. 35, no. 3, pp. 271–292, Mar. 2001, ISSN: 0191-2615. DOI: 10.1016/S0191-2615(99)00051-X.

[8] B. Buchel, T. Spanninger, and F. Corman, "Modeling Evolutionary Dynamics of Railway Delays with Markov Chains," in *2021 7th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, IEEE, Jun. 2021, pp. 1–6, ISBN: 978-1-7281-8995-6. DOI: 10.1109/MT-ITS49943.2021.9529263. [Online]. Available: https://ieeexplore.ieee.org/document/9529263/.

[9] N. Weik and N. Nießen, "A quasi-birth-and-death process approach for integrated capacity and reliability modeling of railway systems," *Journal of Rail Transport Planning and Management*, vol. 7, no. 3, pp. 114–126, Dec. 2017, ISSN: 22109706. DOI: 10.1016/J.JRTPM.2017.06.001. [Online]. Available: http://dx.doi.org/10.1016/j.jrtpm.2017.06.001.

[10] M. Carey and S. Carville, "Scheduling and platforming trains at busy complex stations," *Transportation Research Part A: Policy and Practice*, vol. 37, no. 3, pp. 195–224, Mar. 2003, ISSN: 0965-8564. DOI: 10.1016/S0965-8564(02)00012-5.

[11] J. Yuan and I. A. Hansen, "Optimizing capacity utilization of stations by estimating knock-on train delays," *Transportation Research Part B: Methodological*, vol. 41, no. 2, pp. 202–217, Feb. 2007, ISSN: 0191-2615. DOI: 10.1016/J.TRB.2006.02.004.

[12] R. J. Boucherie, N. M. Van Dijk, R. J. Boucherie, and N. M. Van Dijk, "Monotonicity and error bounds for networks of Erlang loss queues," vol. 62, pp. 159–193, 2009. DOI: 10.1007/s11134-009-9118-9.

[13] I. Bychkov, A. Kazakov, A. Lempert, and M. Zharkov, "Modeling of railway stations based on queuing networks," *Applied Sciences (Switzerland)*, vol. 11, no. 5, Mar. 2021, ISSN: 20763417. DOI: 10.3390/APP11052425.

[14] A. de Kort, B. Heidergott, R. van Egmond, and G. Hooghiemstra, *Train movement analysis at railway stations: Procedures & Evaluation of Wakob 's Approach*, TRIAL, P. Bovy, Ed. Delft University Press, 1999. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3A9a1beba3-4383-4fda-bda3-af97bbfa6746.

[15] Z. Li, P. Huang, C. Wen, X. Jiang, and F. Rodrigues, "Prediction of train arrival delays considering route conflicts at multi-line stations," *Transportation Research Part C: Emerging Technologies*, vol. 138, p. 103 606, May 2022, ISSN: 0968-090X. DOI: 10.1016/J.TRC.2022.103606. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0968090X22000523.

[16] N. M. v. (M. Dijk 1956-, *Queueing networks and product forms : a systems approach LK - https://ut.on.worldcat.org/oclc/782135508*, English. Chichester SE - XIX, 270 p. : illustrations ; 24 cm.: John Wiley, 1993, ISBN: 0471928488 9780471928485. [Online]. Available: http://bvbr.bib-bvb.de:8991/F?func=service&doc_library=BVB01&doc_number=005976880&line_number=0001&func_code=DB_RECORDS&service_type=MEDIA.

[17] R. J. (J. Boucherie 1964-, A. Braaksma, and H. C. Tijms, *Operations research : introduction to models and methods LK - https://ut.on.worldcat.org/oclc/1252850312*, English. Singapore ; SE - xii, 499 pages : illustrations ; 25 cm: World Scientific Publishing Co. Pte. Ltd, 2022, ISBN: 9789811239342 9811239347 9789811239816 9811239819.

[18] treinreiziger.nl, *Hoofdspoorweginfrastructuur in Nederland*, 2021. [Online]. Available: https://www.treinreiziger.nl/wp-content/uploads/2021/01/Infra-NL-210107.pdf.

[19] NOS, *Stations Amersfoort en Eindhoven mogen zich Centraal Station noemen*, Sep. 2019. [Online]. Available: https://nos.nl/artikel/2300168-stations-amersfoort-en-eindhoven-mogen-zich-centraal-station-noemen.

[20] NS, "Grootste, kleinste en snelst groeiende stations 2018," Jul. 2019. [Online]. Available: https://nieuws.ns.nl/download/731822/nsin-enuitstappers2018-409296.pdf.

Part IV
Appendix

9 Station names

Table 16: *Names and abbreviations of the stations stated in this report*

Abbreviation	Station name
AM	Amsterdam C.
AMFS	Amsersfoort Schothorst
AVAT	Amersfoort Vathorst
BHF	Berlin Ostbahnhof
DDR	Dordrecht
DTC	Doetinchem
DV	Deventer
ED	Ede-Wagingen
ES	Enschede
FFFM	Frankfurt Main Hbf.
GN	Groningen
GVC	Den Haag C.
HDR	Den Helder
LW	Leeuwarden
NM	Nijmegen
QDU	Dusseldorg Hbf.
RSD	Roosendaal
RTD	Rotterdam C.
SHL	Schiphol Airport
TI	Tiel
UT	Utrecht C.
WC	Wijchen
WW	Winterswijk
ZL	Zwolle
ZP	Zutphen

10 Arrival schedules

Arnhem

Table 17: *Arrival schedule for Arnhem Central on a workday. The nightjet are not taken into account in this report*

Arrivals from	Final Stop	Arrival	Platform	Serial number	Type
Nijmegen	Schiphol Airport/ Rotterdam	.40/.10	11	3100	NS Intercity
Nijmegen	Den Helder	.56/.26	11	3000	NS Intercity
Den Helder	Nijmegen	.59/.29	8	3000	NS Intercity
Schiphol Airport	Nijmegen	.15/.45	8	3100	NS Intercity
Rotterdam Centraal	Nijmegen	.08/.38	9	3200	NS Intercity
Roosendaal	Zwolle	.37/.07	3	3600	NS Intercity
Zwolle	Roosendaal	.49/.19	7	3600	NS Intercity
Wijchen	Zutphen	.19/.49	3	7600	NS Sprinter
Zutphen	Wijchen	.36/.06	7	7600	NS Sprinter
Last station					
Dordrecht	Arnhem	.02/.32	4a	6600	NS Sprinter
Tiel	Arnhem	.23/.53	4b	31100	Arriva
Winterswijk	Arnhem	.25/.55	6b	30900	Arriva
Ede-Wagingen	Arnhem	.26/.56	10	7500	Arriva
Doetinchem	Arnhem	.10/.40	6b	30700	Breng
Dusseldorf	Arnhem	.13	6a	20000	VIAS
International trains					
Frankfurt	Amsterdam Centraal	.27	9	100	ICE International
Amsterdam Centraal	Frankfurt	.35	9	100	ICE International
Daily trains					
Basel SBS	Amsterdam Centraal	20.57	9		Nightjet
Frankfurt	Amsterdam Centraal	7.51	10		Nightjet
Nurnberg	Amsterdam Centraal	8.51	10		Nightjet
Amsterdam Centraal	Innsbruck	19.48	9		Nightjet

Amersfoort Centraal

Table 18: *Arrival schedule for Amersfoort Centraal on workday around the middle of the day (no rush hours or during night times)*

Arrivals from	Final Stop	Arrival at	Platform	Number	Type
Amersfoort Vathorst	Amsterdam C	.00/.30	7	15800	NS Sprinter
Amsterdam C.	Amersfoort Vathorst	.28/.58	1	15800	NS Sprinter
Den Haag C.	Amersfoort Schothorst	.20	2	11700	NS Intercity
Amersfoort Schothorst	Den Haag C.	.38	6	11700	NS Intercity
Amersfoort Schothorst	Schiphol Airport	.08	7	11600	NS Intercity
Schiphol Airport	Amersfoort Schothorst	.49	1	11600	NS Intercity
Enschede	Den Haag C.	.08	7	1700	NS Intercity
Enschede	Schiphol Airport	.38	6	1600	NS Intercity
Schiphol Airport	Enschede	.18	1	1600	NS Intercity
Den Haag C.	Enschede	.50	2	1700	NS Intercity
Utrecht C.	Zwolle	.10/.40	2	5600	NS Sprinter
Zwolle	Utrecht C.	.17/.47	6	5600	NS Sprinter
Deventer	Amsterdam C	.24/.54	7	1500	NS Intercity
Amsterdam C	Deventer	.04/.34	5a	1500	NS Intercity
Leeuwarden	Rotterdam C.	.25	6	600	NS Intercity
Groningen	Rotterdam C.	.54	6	500	NS Intercity
Rotterdam C	Groningen	.01	2	500	NS Intercity
Rotterdam C	Leeuwarden	.31	2	600	NS Intercity
Last station					
Ede-Wageningen	Amersfoort C.	.01/.31	5b	31300	Stoptrein Valleilijn
Barneveld Zuid	Amersfoort C.	.17/.47	4b	31400	Stoptrein Valleilijn
Den Haag C.	Amersfoort C.	.04/.34	4a	2000	NS Intercity
International trains					
Amsterdam	Berlin Ost	.34 (bihourly)	7	200	Intercity Int.
Berlin Ostbahnhof	Amsterdam	.24 (bihourly)	7	200	Intercity Int.

11 Possible further research

In the model made in this report, it is assumed that a train takes over all tracks on its route in the station. In reality, a train only uses the tracks for riding to the platform and it will wait there for a longer time. Below, a way of modeling this is given.

A train station can be divided into N separate modeling parts, each acting as their own queue. Such parts can for example be between the numbered tracks indicated in in figures 6.2 and 7.2. Let every line have its own train type t, with T being the set of all train types. Let λ_t be the external arrival intensity of a train type t. And let μ_{it} be handling at tracks i for train type t. Then one could write $\mu_{0,t} = \lambda_t$. For the handling rate, this could be hard to analyze from real life data, as it is not publicly recorded at what time the trains arrive at certain infra-elements. However, a basic distance/speed should give a rough indication.

A state $\bar{n}$ includes the number of trains at each of the stations. Let I be the set of admissible states. When a train of type t moves from modeling track i to track j, one can write $\bar{n} - e_{i,t} + e_{j,t}$. However, since such a transition is not always possible, introduce the following :

$b_{ij}(\bar{n})$: probability that a transition $\bar{n} + e_{i,t} \to \bar{n} + e_{j,t}$ for a train t is possible

Often, just as in equation (5.2), this can be described as $b_{ij} = 1_{\{n_j < N_J\}}$

Now one can parameterize the following

$$\begin{cases} p_{ijt}(\bar{n}) = p_{ijt} b_{ij}(\bar{n}) & (j \neq i) \\ p_{iit}(\bar{n}) = 1 - p_{iot} b_{iot}(\bar{n}) - \sum_{j \neq i} p_{ijt} b_{ij}(\bar{n}) \end{cases}$$

where in the latter, it is assumed that a train is blocked and does the handling time for that modelling track i again.

Or, when assuming that a train which cannot move on to the next modeling station is removed out of the system,

$$\begin{cases} p_{ijt}(\bar{n}) = p_{ijt} b_{ij}(\bar{n}) & (j \neq i) \\ p_{i0t}(\bar{n}) = 1 - \sum_{j=1}^{N} p_{ijt} b_{ij}(\bar{n}) \end{cases}$$

Now, following the balance and normalization equations as in (2.9) and (2.10), the steady state distribution $\pi(\bar{n})$ at the set of admissible states I is given by

$$\begin{aligned} \pi(\bar{n}) \sum_{j=0}^{N} \sum_{t \in T} \mu_{jt} \sum_{i} p_{jit}(\bar{n}) = \\ \sum_{j=0}^{N} \sum_{i} \sum_{t \in T} \pi(\bar{n} - e_{jt} + e_{it}) \mu_{it} p_{ijt}(\bar{n} - e_{jt} + e_{it}) \end{aligned} \tag{11.1}$$

And just as in the proposed model, also here a detailed balance equations can be made, this will be written as:

$$\begin{aligned} \pi(\bar{m} + e_j) \mu_j \sum_{t \in T} \sum_{i} p_{ji}(\bar{m} + e_j) = \\ \sum_{t \in T} \sum_{i} \pi(\bar{m} + e_i) \mu_i p_{ij}(\bar{m} + e_i) \end{aligned} \tag{11.2}$$

Which for all $j = 0, 1, ..., N$ wiht $\bar{n} = \bar{m} + e_j$, indeed verifies the global balance equation (11.1).

Now one continue with the same reasoning as in the model in section 3, and propose a function $H(\bar{n})$ such that

$$\begin{aligned} H(\bar{m} + e_j) \sum_{t \in T} \sum_{i} p_{ji}(\bar{m} + e_j) = \\ \sum_{t \in T} \sum_{i} H(\bar{m} + e_i) p_{ij}(\bar{m} + e_i) \end{aligned} \tag{11.3}$$

and so

$$\pi(\bar{n} = cH(\bar{n}) \prod_{i=1}^{N} (\frac{1}{\mu_i})^{n_i} \tag{11.4}$$

However, unless in the model proposed, this equations does not necessary have a solution and can therefore not be so easily used as the problem proposed. Therefore, further research need to be done in order to expand the model in this way. Next to better model the reality in term of track use, this type of model will also make it easier to implement departure from the railway station.

12 Code

Model Arnhem

Python function to be used to calculate the model results for Arnhem Centraal railway station

```
#inport packages
from unicodedata import name
from numpy import equal
import pandas as pd
from openpyxl import load_workbook
import itertools


class TrainSource(): #to keep track of the train sources in the
    system
    def __init__(self, name, serialnr, arrival, handling):
        self.name = name
        self.serialnr = serialnr
        self.arrival = arrival
        self.handling = handling

    def __str__(self):
        about = f"Source␣{self.name}␣with␣arrival␣{self.arrival}␣
            and␣handling␣rate␣{self.handling}"
        return about

class ActiveSources():
    def __init__(self, listOfSources):
        name = ""
        for i in listOfSources:
            name = name + i.name
        self.name = name
        self.sources = listOfSources

def PFArnhemFunction():
    df = pd.read_excel('model/sources.xlsx', index_col=0)
    sources = []

    possible =[]

    #combinations that are not allowed
    prohibitedCombi = [['','s1', 's3', 's6', 's8', 's11' , 's13'],
         ['','s2', 's4', 's5', 's10', 's15', 's17'], ['','s7', 's9
        ', 's12', 's14', 's16']]
    for i in range(1, df.__len__()+1): #make source list
        thisSource = TrainSource("s"+str(i),df.loc[i]['serial␣
            number'], df.loc[i]['arrival␣rate'], df.loc[i]['
            handling␣rate'] )
        sources.append(thisSource)
```

```
#get list of possible combitions of source names
possibleNames = []
for i in prohibitedCombi[0]:
    for i2 in prohibitedCombi[1]:
        for i3 in prohibitedCombi[2]:
            combi = [i, i2, i3]
            combi2 = [i for i in combi if not i==’’]
            if combi2==[’s17’, ’s16’]: #this combinations makes
                 use of the same platform
                break
            possibleNames.append(combi2)

#get list of possible combinations of sources
for i in possibleNames:
    sc = []
    if i==[’s17’, ’s16’]:
        break
    for name in i:
        for s in sources:
            if s.name.__eq__(name):

                sc.append(s)
    possible.append(sc)

steadyStates =[]
sourcesBusy =[]
accept = []
for i in range(0, len(sources)+1):
    sourcesBusy.append(0)
    accept.append(0)
#calculate steady state distributions with product form
for p in possible:
    pi = 1
    for s in p:
        pi = pi*s.arrival/s.handling
    steadyStates.append(pi)
    for s in p:
        nameint = int(s.name[1:])
        sourcesBusy[nameint] += pi
#apply normalisation condition
sumSS = sum(steadyStates)
steadyStatesNorm=[x/sumSS for x in steadyStates]
sourcesBusyNorm =[x/sumSS for x in sourcesBusy]

#calculate acceptance rate for every source
count = 1
for source in sources:
    countp = 0
    for p in possibleNames:
```

```
            p2= p.copy()
            p2.append(source.name)
            lists = itertools.permutations(p2, len(p2))
            for lis in lists:
                listTest = list(lis)
                if list(lis) in possibleNames:
                    accept[count] += steadyStatesNorm[countp]
            countp+=1
        count +=1

    return sourcesBusyNorm, accept
```

Model Amersfoort

Python function to be used to calculate the model results for Amersfoort Centraal railway station. It includes a general function, for the routing how it is done now and an function to opt for rerouting.

```
#import packages
from unicodedata import name
from numpy import equal
import pandas as pd
from openpyxl import load_workbook
import itertools
import inspect

class TrainSource(): #to keep track of the train sources in the
    system
    def __init__(self, name, serialnr, arrival, handling):
        self.name = name
        self.serialnr = serialnr
        self.arrival = arrival
        self.handling = handling

    def __str__(self):
        about = f"Source␣{self.name}␣with␣arrival␣{self.arrival}␣
            and␣handling␣rate␣{self.handling}"
        return about

class ActiveSources():
    def __init__(self, listOfSources):
        name = ""
        for i in listOfSources:
            name = name + i.name
        self.name = name
        self.sources = listOfSources

def productFormAmersfoort(): #function to calculate model results
    #to get input variables
    df = pd.read_excel('model/sources.xlsx', index_col=0,
        sheet_name="Amersfoort")
    sources = []
    possible =[]

    #combinations that are not allowed due to same entrance tracks
    prohibitedCombi = [['','s2', 's6', 's9', 's14', 's22'], ['','
        s21', 's17', 's18', 's10', 's11', 's3'], ['','s1', 's4', '
        s5', 's12', 's15', 's16'], ['','s7', 's8', 's13', 's19', '
        s20', 's23']]
    for i in range(1, 24):
        thisSource = TrainSource("s"+str(i),df.loc[i]['serial␣
            number'], df.loc[i]['arrival␣rate'], df.loc[i]['
            handling␣rate'] )
```

```
    sources.append(thisSource)

pp=[]
#other not allowed combinations due to same platform use
prohibitedPlatform = [['s23', 's22', 's13', 's7', 's5', 's1'],
     ['s4', 's8', 's12', 's15', 's16']]
for k in prohibitedPlatform[0]:
    for l in prohibitedPlatform[0]:
        pp.append([k, l])
for k in prohibitedPlatform[1]:
    for l in prohibitedPlatform[1]:
        pp.append([k, l])
#create list of possible sources combinations names list
possibleNames = []
for i in prohibitedCombi[0]:
    for i2 in prohibitedCombi[1]:
        for i3 in prohibitedCombi[2]:
            for i4 in prohibitedCombi[3]:
                combi = [i, i2, i3, i4]
                combi2 = [i for i in combi if not i=='']
                for p in pp:
                    if combi2==p:
                        break
                else: possibleNames.append(combi2)

#create list of possible combinations of sources
for i in possibleNames:
    sc = []
    for name in i:
        for s in sources:
            if s.name.__eq__(name):
                sc.append(s)
    possible.append(sc)

#calculate steady state distributions with product form
steadyStates =[]
sourcesBusy =[]
accept = []
for i in range(0, len(sources)+1):
    sourcesBusy.append(0)
    accept.append(0)
for p in possible:
    pi = 1
    for s in p:
        pi = pi*s.arrival/s.handling
    steadyStates.append(pi)
    for s in p:
        nameint = int(s.name[1:])
        sourcesBusy[nameint] += pi
#apply normalisation condition
```

```
    sumSS = sum(steadyStates)
    steadyStatesNorm=[x/sumSS for x in steadyStates]
    sourcesBusyNorm =[x/sumSS for x in sourcesBusy]

    #calculate acceptance rate
    count = 1
    for source in sources:
        countp = 0
        for p in possibleNames:
            p2= p.copy()
            p2.append(source.name)
            lists = itertools.permutations(p2, len(p2))
            for lis in lists:
                listTest = list(lis)
                if list(lis) in possibleNames:
                    accept[count] += steadyStatesNorm[countp]
            countp+=1
        count +=1

    return sourcesBusyNorm, accept

#same function as above, but with the input variables from the
    rerouting.
#The routing can easily be adapted by changes the route in the
    excel sheet
def productFormAmersfoortREROUTING():
    df = pd.read_excel('model/sources.xlsx', index_col=0,
        sheet_name="Amersfoort")

    sources = []

    possible =[]

    prohibitedCombi = [['','s2', 's6', 's9', 's14', 's22'], ['','
        s21', 's17', 's18', 's10', 's11', 's3'], ['','s1', 's4', '
        s5', 's12', 's15', 's16'], ['','s7', 's8', 's13', 's19', '
        s20', 's23']]
    for i in range(1, 24):
        thisSource = TrainSource("s"+str(i),df.loc[i]['serial␣
            number'], df.loc[i]['arrival␣rate'], df.loc[i]['
            handling␣rate'] )
        sources.append(thisSource)

    pp=[]
    prohibitedPlatform = [['s23', 's22', 's13', 's7', 's8','s5', '
        s3'], ['s4', 's1', 's12', 's15', 's16', 's19']]
    for k in prohibitedPlatform[0]:
        for l in prohibitedPlatform[0]:
            pp.append([k, l])
    for k in prohibitedPlatform[1]:
```

```
    for l in prohibitedPlatform[1]:
        pp.append([k, l])
possibleNames = []
for i in prohibitedCombi[0]:
    for i2 in prohibitedCombi[1]:
        for i3 in prohibitedCombi[2]:
            for i4 in prohibitedCombi[3]:
                combi = [i, i2, i3, i4]
                combi2 = [i for i in combi if not i==’’]
                for p in pp:
                    if combi2==p:
                        break
                else: possibleNames.append(combi2)

for i in possibleNames:
    sc = []
    for name in i:
        for s in sources:
            if s.name.__eq__(name):
                sc.append(s)
    possible.append(sc)

steadyStates =[]
sourcesBusy =[]
accept = []
for i in range(0, len(sources)+1):
    sourcesBusy.append(0)
    accept.append(0)
for p in possible:
    pi = 1
    for s in p:
        pi = pi*s.arrival/s.handling
    steadyStates.append(pi)
    for s in p:
        nameint = int(s.name[1:])
        sourcesBusy[nameint] += pi

sumSS = sum(steadyStates)
steadyStatesNorm=[x/sumSS for x in steadyStates]
sourcesBusyNorm =[x/sumSS for x in sourcesBusy]

count = 1
for source in sources:
    countp = 0
    for p in possibleNames:
        p2= p.copy()
        p2.append(source.name)
        lists = itertools.permutations(p2, len(p2))
        for lis in lists:
            listTest = list(lis)
```

```
            if list(lis) in possibleNames:
                accept[count] += steadyStatesNorm[countp]
        countp+=1
    count +=1

return sourcesBusyNorm, accept
```

Simulation function

The functions in these code can be used to attain simulation results. The first function assumes exponentially distributed handling rate, the second one has an mixed Erlang distribution. The function makes use of randomization, so results will differ each run.

```
from re import S
import simpy
import random
import bisect
import cv2
import numpy as np
from itertools import chain
from collections import Counter


def simulationStationArrivals(totTime, arrivalRates,
    handlingRates, nrSources, nrTracks, lines, routes):
    NUM_TRACKS = nrTracks #number of set tracks in station
    NUM_SOURCES = nrSources #number of sources generating trains
        for the station
    TOT_TIME = totTime #total time to simulate

    #make sure input variables are (more or less) correct
    if not len(arrivalRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")
    if not len(handlingRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")

    class Source():
        #setting up the Source
        def __init__(self, name, line, arrivalRate, handlingRate,
            enterTracks):
            self.name = name
            self.line = line
            lines.append(line)
            self.arrivalRate = arrivalRate
            self.handlingRate = handlingRate
            self.enterTracks = enterTracks
            self.generated = []
            self.busy = 0
            self.blocked = 0

        #activate function to start generating trains
        def generate(self, env):
            self.count = 0 #to keep track of amount of trains
                generated
            while True:
                interArrival = getArrivalTime(self.arrivalRate)
                yield env.timeout(interArrival) #block source for
```

```
            self.count += 1
            trainName = "Train"+self.line+"."+str(self.count)
            t = Train(trainName, self.line, self.handlingRate,
                self) #make train
            env.process(t.arrivalProcess(self.enterTracks, env)
                ) #start process of train arriving

    def getCount(self):
        return self.count

    def addBusy(self, time):
        self.busy += time

    def getBlocking(self): #to get blocking rate
        if self.count ==0:
            return 0
        else: return self.blocked/self.count

class Train():
    def __init__(self, name, line, handlingRate, source):
        self.name = name
        self.line = line
        self.handlingRate = handlingRate
        self.handlingTime = getHandlingTime(self.handlingRate)
        self.source = source
    def arrivalProcess(self, enterTracks, env):
        if not checkFreeTracks(enterTracks): #train is blocked
             and is removed from system
            occupied = getOccupiedTrack(enterTracks)
            blockedTracks.append(occupied)
            blockedLines.append(self.line)
            self.source.blocked +=1
        else: #train can arrive at station

            goTracks(enterTracks, self.name)
            self.source.addBusy(self.handlingTime)
            yield env.timeout(self.handlingTime)
            leaveTracks(enterTracks, self.name)

def getArrivalTime(arrivalRate):
    return random.expovariate(arrivalRate) #exponentially
        distributed arrival rate

def getHandlingTime(handlingRate):
    return random.expovariate(handlingRate) #exponentially
        distributed handling rate

def goTracks(enterTracks, trainNumber): #set train to the
    given tracks
    if trainNumber.__eq__("0"):
```

```
        print("trainNumber cannot be zero")
        raise Exception("train number is zero")
    if not checkFreeTracks(enterTracks):
        raise Exception("Tracks already occupied")
    for enter in enterTracks:
        tracks[enter].enterTrain(trainNumber)

def checkFreeTracks(enterTracks): #to check if given tracks
    are free
    for enter in enterTracks:
        if not tracks[enter].train.__eq__('0'):
            return False
    else:
        return True

def getOccupiedTrack(enterTracks): #in case tracks are not
    free, this function return the tracks that are blocked
    occupied = []
    for enter in enterTracks:
        if not tracks[enter].train.__eq__('0'):
            occupied.append(enter)
    return occupied

def leaveTracks(enterTracks, trainNumber): #train is removed
    from tracks
    for enter in enterTracks:
        if not tracks[enter].train.__eq__(trainNumber):
            raise Exception("train not on this track")
        tracks[enter].removeTrain(trainNumber)

class Track(): #track object to keep track of train that are
    on it
    def __init__(self, name):
        self.name= name
        self.train = "0"
        self.previousTrains = []
        self.nrtrains = 0
    def enterTrain(self, trainNumber):
        self.train = trainNumber
    def removeTrain(self, trainNumber):
        self.previousTrains.append(trainNumber)
        self.nrtrains += 1
        self.train = "0"
    def getNrTrains(self):
        return self.nrtrains

#list for the track and tracknames
tracks = []
trackList = [i for i in range(0, NUM_TRACKS+1)]
#to keep track of which tracks and trainlines are blocked
```

```
blockedTracks = []
blockedLines = []
sources = []

for t in trackList:
    tracks.append(Track(name = t))

#setting up all the sources
for i in range(0, NUM_SOURCES):
    thisSource = Source("s"+str(i+1), lines[i], arrivalRates[i
        ], handlingRates[i], routes[i])
    sources.append(thisSource)

#create simulation environment
env = simpy.Environment()

#starting up the source to generate trains
for s in sources:
    env.process(s.generate(env))

#run simulation
env.run(until=TOT_TIME)

#determine for each track how often it is blocking
tracksBlockedCounter = []
def countList(list, x):
    return Counter(chain.from_iterable(set(i) for i in list))[
        x]
for i in range(0, NUM_TRACKS+1):
    tracksBlockedCounter.append(countList(blockedTracks, i))

#calculate blocking rate for tracks
count = 0
conflictTracks = []
for tr in tracks:
    counter = tracksBlockedCounter[count]
    if counter == 0:
        conflictTracks.append(0)
        count+=1
    else:
        percentageConflict = counter/(tr.getNrTrains()+counter
            )
        conflictTracks.append(percentageConflict)
        count +=1

#calculate results for the differen sources
result = []
percentageBlocked = []
for t in sources:
    result.append(t.busy/TOT_TIME)
```

```
        v= t.getBlocking()
        percentageBlocked.append(v)

    return result, blockedTracks, percentageBlocked,
        conflictTracks

#same function as above but instead of exponentially dsitributed
    handling rate, here a mixed Erlang distribution is used
def simulationStationArrivalsERLANG(totTime, arrivalRates,
    handlingRates, nrSources, nrTracks, lines, routes):
    NUM_TRACKS = nrTracks #number of set tracks in station
    NUM_SOURCES = nrSources #number of sources generating trains
        for the station
    TOT_TIME = totTime #total time to simulate

    #make sure input variables are (more or less) correct
    if not len(arrivalRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")
    if not len(handlingRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")

    class Source():
        #setting up the Source
        def __init__(self, name, line, arrivalRate, handlingRate,
            enterTracks):
            self.name = name
            self.line = line
            lines.append(line)
            self.arrivalRate = arrivalRate
            self.handlingRate = handlingRate
            self.enterTracks = enterTracks
            self.generated = []
            self.busy = 0
            self.blocked = 0

        #activate function to start generating trains
        def generate(self, env):
            self.count = 0 #to keep track of amount of trains
                generated
            while True:
                interArrival = getArrivalTime(self.arrivalRate)
                yield env.timeout(interArrival) #block source for
                self.count += 1
                trainName = "Train"+self.line+"."+str(self.count)
                t = Train(trainName, self.line, self.handlingRate,
                    self) #make train
                env.process(t.arrivalProcess(self.enterTracks, env)
                    ) #start process of train arriving

        def getCount(self):
```

```
        return self.count

    def addBusy(self, time):
        self.busy += time

    def getBlocking(self): #to get blocking rate
        if self.count ==0:
            return 0
        else: return self.blocked/self.count

class Train():
    def __init__(self, name, line, handlingRate, source):
        self.name = name
        self.line = line
        self.handlingRate = handlingRate
        self.handlingTime = getHandlingTime(self.handlingRate)
        self.source = source
    def arrivalProcess(self, enterTracks, env):
        if not checkFreeTracks(enterTracks): #train is blocked
             and is removed from system
            occupied = getOccupiedTrack(enterTracks)
            blockedTracks.append(occupied)
            blockedLines.append(self.line)
            self.source.blocked +=1
        else: #train can arrive at station

            goTracks(enterTracks, self.name)
            self.source.addBusy(self.handlingTime)
            yield env.timeout(self.handlingTime)
            leaveTracks(enterTracks, self.name)

def getArrivalTime(arrivalRate):
    return random.expovariate(arrivalRate) #exponentially
        distributed arrival rate

def getHandlingTime(handlingRate):
    k=np.random.randint(1, 100)
    #k=100
    theta = 1/(handlingRate*k)
    return np.random.default_rng().gamma(k, theta) #
        exponentially distributed handling rate

def goTracks(enterTracks, trainNumber):
    if trainNumber.__eq__("0"):
        print("trainNumber␣cannot␣be␣zero")
        raise Exception("train␣number␣is␣zero")
    if not checkFreeTracks(enterTracks):
        raise Exception("Tracks␣already␣occupied")
    for enter in enterTracks:
        tracks[enter].enterTrain(trainNumber)
```

```
def checkFreeTracks(enterTracks):
    for enter in enterTracks:
        if not tracks[enter].train.__eq__('0'):
            return False
    else:
        return True

def getOccupiedTrack(enterTracks):
    occupied = []
    for enter in enterTracks:
        if not tracks[enter].train.__eq__('0'):
            occupied.append(enter)
    return occupied

def leaveTracks(enterTracks, trainNumber):
    for enter in enterTracks:
        if not tracks[enter].train.__eq__(trainNumber):
            raise Exception("train␣not␣on␣this␣track")
        tracks[enter].removeTrain(trainNumber)

class Track():
    def __init__(self, name):
        self.name= name
        self.train = "0"
        self.previousTrains = []
        self.nrtrains = 0
    def enterTrain(self, trainNumber):
        self.train = trainNumber
    def removeTrain(self, trainNumber):
        self.previousTrains.append(trainNumber)
        self.nrtrains += 1
        self.train = "0"
    def getNrTrains(self):
        return self.nrtrains


tracks = []
trackList = [i for i in range(0, NUM_TRACKS+1)]
blockedTracks = []
blockedLines = []
sources = []

for t in trackList:
    tracks.append(Track(name = t))

for i in range(0, NUM_SOURCES):
    thisSource = Source("s"+str(i+1), lines[i], arrivalRates[i
        ], handlingRates[i], routes[i])
    sources.append(thisSource)
```

```
    env = simpy.Environment()

    for s in sources:
        env.process(s.generate(env))

    env.run(until=TOT_TIME)

    tracksBlockedCounter = []
    def countList(list, x):
        return Counter(chain.from_iterable(set(i) for i in list))[
            x]
    for i in range(0, NUM_TRACKS+1):
        tracksBlockedCounter.append(countList(blockedTracks, i))
    count = 0
    conflictTracks = []
    for tr in tracks:
        counter = tracksBlockedCounter[count]
        if counter == 0:
            conflictTracks.append(0)
            count+=1
        else:
            percentageConflict = counter/(tr.getNrTrains()+counter
                )
            conflictTracks.append(percentageConflict)
            count +=1

    result = []
    blockedLinesCount = []
    percentageBlocked = []
    for t in sources:
        result.append(t.busy/TOT_TIME)
        v= t.getBlocking()
        percentageBlocked.append(v)

    return result, blockedTracks, percentageBlocked,
        conflictTracks
```

Simulation function with scheduling

Same function as above, but here an extra input variable for scheduling is provided.

```
from re import S
import simpy
import random
import bisect
import cv2
import numpy as np
from itertools import chain
from collections import Counter
from simpy.util import start_delayed

def simulationStationArrivalsTiming(totTime, arrivalRates,
    handlingRates, nrSources, nrTracks, lines, routes,
    startingTimes):
    NUM_TRACKS = nrTracks
    NUM_SOURCES = nrSources
    TOT_TIME = totTime
    if not len(arrivalRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")
    if not len(handlingRates)==NUM_SOURCES:
        raise Exception("not␣good␣amount␣of␣arrival␣rates")

    class Source():
        #setting up the Source
        def __init__(self, name, line, arrivalRate, handlingRate,
            enterTracks):
            self.name = name
            self.line = line
            self.arrivalRate = arrivalRate
            self.handlingRate = handlingRate
            self.enterTracks = enterTracks
            self.generated = []
            self.busy = 0

        def generate(self, env):
            self.count = 0
            while True:
                interArrival = getArrivalTime(self.arrivalRate)
                yield env.timeout(interArrival)
                self.count += 1
                trainName = "Train"+self.line+"."+str(self.count)
                t = Train(trainName, self.line, self.handlingRate,
                    self)

                env.process(t.arrivalProcess(self.enterTracks, env)
                    )
```

```
    def getCount(self):
        return self.count

    def addBusy(self, time):
        self.busy += time

class Train():
    def __init__(self, name, line, handlingRate, source):
        self.name = name
        self.line = line
        self.handlingRate = handlingRate
        self.handlingTime = getHandlingTime(self.handlingRate)
        self.source = source
    def arrivalProcess(self, enterTracks, env):
        if not checkFreeTracks(enterTracks):
            occupied = getOccupiedTrack(enterTracks)
            blockedTracks.append(occupied)
            blockedLines.append(self.line)
        else:

            goTracks(enterTracks, self.name)
            self.source.addBusy(self.handlingTime)
            yield env.timeout(self.handlingTime)


def getArrivalTime(arrivalRate):
    return random.expovariate(arrivalRate)

def getHandlingTime(handlingRate):
    return random.expovariate(handlingRate)

def goTracks(enterTracks, trainNumber):
    if trainNumber.__eq__("0"):
        print("trainNumber␣cannot␣be␣zero")
        raise Exception("train␣number␣is␣zero")
    if not checkFreeTracks(enterTracks):
        raise Exception("Tracks␣already␣occupied")
    for enter in enterTracks:
        tracks[enter].enterTrain(trainNumber)

def checkFreeTracks(enterTracks):
    for enter in enterTracks:
        if not tracks[enter].train.__eq__('0'):
            return False
    else:
        return True

def getOccupiedTrack(enterTracks):
    occupied = []
    for enter in enterTracks:
```

```
            if not tracks[enter].train.__eq__('0'):
                occupied.append(enter)
        return occupied

    def leaveTracks(enterTracks, trainNumber):
        for enter in enterTracks:
            if not tracks[enter].train.__eq__(trainNumber):
                raise Exception("train␣not␣on␣this␣track")
            tracks[enter].removeTrain(trainNumber)

    class Track():
        def __init__(self, name):
            self.name= name
            self.train = "0"
            self.previousTrains = []
        def enterTrain(self, trainNumber):
            self.train = trainNumber
        def removeTrain(self, trainNumber):
            self.previousTrains.append(trainNumber)
            self.train = "0"


    tracks = []
    trackList = [i for i in range(0, NUM_TRACKS)]
    blockedTracks = []
    blockedLines = []
    sources = []
    for t in trackList:
        tracks.append(Track(name = t))
    for i in range(0, NUM_SOURCES):
        thisSource = Source("s"+str(i+1), lines[i], arrivalRates[i
            ], handlingRates[i], routes[i])
        sources.append(thisSource)

    env = simpy.Environment()
    count = 0
    for s in sources:
        start_delayed(env, s.generate(env), startingTimes[count] )
        count+=1

    env.run(until=TOT_TIME)

    result = []
    for t in sources:
        result.append(t.busy/TOT_TIME)

    return result, blockedTracks
```

Image making

The functions below can be used to make the imaging showing the blocking rate at the different tracks for stations. The function are made for the simple made-up station and Arnhem C. The results are not as clear as if hand made and making such a code takes a lot of time. Therefore, no function for Amersfoort C. has been made.

```
import cv2
import numpy as np
import pandas as pd

green = (0, 255, 0)
red = (52, 52, 235)
orange = (0, 165, 255)
yellow = (10, 219, 254)
white = (255, 255, 255)
black = (0, 0, 0)
def makeImageSimpleStation(blockedTracks):
    first, median, third = np.quantile(blockedTracks, [0.25, 0.5,
        0.75])
    def getColor(number, first, median, third):
        if number>third:
            return red
        elif number>median:
            return orange
        elif number>first:
            return yellow
        elif number>0:
            return green
        else:
            return white
    height = 512
    width = 700
    img = np.zeros((height, width, 3), np.uint8)
    img.fill(255)

    track0 = [(30, 170),(130, 170), getColor(blockedTracks[0],
        first, median, third)]
    cv2.putText(img, "1", (80, 180), fontFace=FONTTHIS, fontScale
        =0.5, color=black,thickness=1)
    track5 = [(30, 340),(230, 340), getColor(blockedTracks[5],
        first, median, third)]
    track1 = [(130, 170),(380, 170), getColor(blockedTracks[1],
        first, median, third)]
    track3= [(130, 170),(230, 340), getColor(blockedTracks[3],
        first, median, third)]
    track4 = [(280, 340),(380, 170), getColor(blockedTracks[4],
        first, median, third)]
    track6= [(230, 340),(280, 340), getColor(blockedTracks[6],
        first, median, third)]
```

```
    track2 = [(380, 170),(450, 170), getColor(blockedTracks[2],
        first, median, third)]
    track7 = [(280, 340),(450, 340), getColor(blockedTracks[7],
        first, median, third)]


    FONTTHIS = cv2.FONT_HERSHEY_DUPLEX
    tracksImage = []
    tracksImage.append(track0)
    tracksImage.append(track1)
    tracksImage.append(track2)
    tracksImage.append(track3)
    tracksImage.append(track4)
    tracksImage.append(track5)
    tracksImage.append(track6)
    tracksImage.append(track7)
    for track in tracksImage:
        cv2.line(img, track[0], track[1], track[2], 3)
    cv2.rectangle(img, (450, 226),(600, 286), color=black,
        thickness=5)
    cv2.putText(img, "platform␣1", (475, 215), fontFace=FONTTHIS,
        fontScale=0.5, color=black,thickness=1)
    cv2.putText(img, "platform␣2", (475, 305), fontFace=FONTTHIS,
        fontScale=0.5, color=black,thickness=1)
    cv2.rectangle(img,(450, 10), (650, 150), color=black,thickness
        =3)
    cv2.line(img, (470, 38), (520, 38), red, 3)
    cv2.putText(img, ">"+str(int(third)), (550, 40),fontFace=
        FONTTHIS, fontScale=0.5, color=black,thickness=1)
    cv2.line(img, (470, 66), (520, 66), orange, 3)
    cv2.putText(img, ">"+str(int(median)), (550, 68),fontFace=
        FONTTHIS, fontScale=0.5, color=black,thickness=1)
    cv2.line(img, (470, 94), (520, 94), yellow, 3)
    cv2.putText(img, ">"+str(int(first)), (550, 96),fontFace=
        FONTTHIS, fontScale=0.5, color=black,thickness=1)
    cv2.line(img, (470, 122), (520, 122), green, 3)
    cv2.putText(img, ">0", (550, 124),fontFace=FONTTHIS, fontScale
        =0.5, color=black,thickness=1)
    #cv2.line(img, pt1, pt2, color)
    cv2.imshow('Binary',img)
    cv2.waitKey(0)

def makeImageArnhem(blockedTracks):

    def getColor(number):
        if number>third:
            return red
        elif number>median:
            return orange
        elif number>first:
```

```
        return yellow
    elif number>0:
        return green
    else:
        return white
height = 512
width = 700
img = np.zeros((height, width, 3), np.uint8)
img.fill(255)
first, median, third = np.quantile(blockedTracks, [0.25, 0.5,
    0.75])
track1 = [(10,170), (110,170), getColor(blockedTracks[1])]
track2 = [(110, 170), (140, 100), getColor(blockedTracks[2])]
track21 = [(140, 100), (170, 100), getColor(blockedTracks[2])]
track3 = [(110, 170), (140, 170), getColor(blockedTracks[3])]
track41 = [(140, 170), (150, 150), getColor(blockedTracks[4])]
track4 = [(150, 150), (180, 150), getColor(blockedTracks[4])]
track5 = [(140, 170), (170, 170), getColor(blockedTracks[5])]
track42 = [(180, 150), (190, 170), getColor(blockedTracks[4])]
track7 = [(190, 170), (200, 170), getColor(blockedTracks[7])]
track6 = [(200, 170) , (240, 90), getColor(blockedTracks[6])]
track8 = [(190, 170), (180, 190), getColor(blockedTracks[8])]
track81 = [(180, 190), (150, 190), getColor(blockedTracks[8])]
track9 = [(200, 170), (180,210 ), getColor(blockedTracks[9])]
track10 = [(180, 210), (150, 210), getColor(blockedTracks[10])
    ]
track111= [(180, 210),(170, 230), getColor(blockedTracks[11])]
track11 = [(170, 230), (140, 230), getColor(blockedTracks[11])
    ]
track151 = [(170, 260), (140, 260), getColor(blockedTracks
    [15])]
track15 = [(140, 260), (135, 250), getColor(blockedTracks[15])
    ]
track152 = [ (135, 250), (110,250), getColor(blockedTracks
    [15])]
track17 = [(132, 260), (115, 260), getColor(blockedTracks[17])
    ]
track16 = [ (115, 260),(110,250), getColor(blockedTracks[16])]
track14 = [(110,250),(50, 250), getColor(blockedTracks[14])]
track181 = [(115, 260), (120, 270), getColor(blockedTracks
    [18])]
track18 = [(170,270), (120, 270), getColor(blockedTracks[18])]
track13 = [(110,80),(50, 250), getColor(blockedTracks[13])]
track131 = [(110,80), (170,80), getColor(blockedTracks[13])]
track12 = [(50, 250), (10, 250), getColor(blockedTracks[12])]
tracksImage = [track1, track2, track21, track3, track4,
    track41, track5, track6, track7, track42, track81, track8,
     track9, track10, track111 , track11, track15, track151,
    track152, track17, track16, track14, track18, track181,
    track13, track131, track12]
```

```
for t in tracksImage:
    cv2.line(img, t[0], t[1], t[2], thickness=2)
cv2.imshow('Binary',img)
cv2.waitKey(0)
```

Attaining and comparing result

The code below can serve as an example how to attain the results using the functions above and to make plots. The code for both Arnhem C, including the image making, and Amersfoort C, which includes taking all variables from all excel sheet. The latter provide for easy change making. Same codes are used for insensitivy and rerouting tests.

Arnhem results

```
from simulationFunction import simulationStationArrivals
from simulationFunction import simulationStationArrivalsERLANG
from itertools import chain
from collections import Counter
import matplotlib.pyplot as plt
import statistics as stats
import pandas as pd
from scipy import stats as st
from PFArnhem import PFArnhemFunction
from imageMaking import makeImageArnhem
NUM_TRACKS = 19
NUM_SOURCES = 17
TOT_TIME = 1000

arrivalRates = []
handlingRates = []
lines = ["3000_NIJ","3000_DH", "3100_NIJ", "3100_SCH", "3200_ROT"
    , "3600_ROO", "3600_ZWO", "7600_WIJ", "7600_ZUT" ,"6600", "
    31100", "30900", "7500", "30700", "20000", "100_FRA", "100
    _AMS"]
routes = [[12, 13], [1, 3, 5], [12,13], [1, 3,5], [1, 3, 4], [12,
     14, 16, 18], [6, 7, 8], [12, 14, 16, 18], [6, 7, 8], [12,
    14, 16, 17], [12, 14, 15], [6, 9, 10], [1, 2], [6, 9, 10],
    [6, 9, 11], [6, 7, 4], [1, 3, 4]]
lenLines = len(lines)
df = pd.read_excel('model/sources.xlsx', index_col=0)
for i in range(1, df.__len__()+1):
    arrivalRates.append(df.loc[i]['arrival␣rate'])
    handlingRates.append(df.loc[i]['handling␣rate'] )

allResults =[]
allBlockedTracks = []
percentageBlocked = []
conflictTracksPercentage = []
for i in range(0, NUM_SOURCES+1):
    allResults.append([])
    percentageBlocked.append([])

for i in range(0, NUM_TRACKS+1):
    conflictTracksPercentage.append([])
```

```
for i in range(0, 1000):
    result, blockedTracks, pb, conflictTracks =
        simulationStationArrivals(TOT_TIME, arrivalRates,
        handlingRates, NUM_SOURCES, NUM_TRACKS, lines, routes)
    count = 1
    for r in result:
        allResults[count].append(r)
        count+=1
    for b in blockedTracks:
        allBlockedTracks.append(b)
    c = 1
    for w in pb:
        percentageBlocked[c].append(w)
        c +=1
    c = 0
    for ct in conflictTracks:
        conflictTracksPercentage[c].append(ct)
        c+=1

def countList(list, x):
     return Counter(chain.from_iterable(set(i) for i in list))[x]

meanConflictTracks = []
tracksBlockedCounter = []
for i in range(0, NUM_TRACKS):
    tracksBlockedCounter.append(countList(allBlockedTracks, i))
    meanConflictTracks.append(stats.mean(conflictTracksPercentage[
        i]))
productForm, acceptModel = PFArnhemFunction()

t = []
p = []
means = []
meanAccept = []
stdevs =[]
for i in range(1, NUM_SOURCES+1):
    meanNow = stats.mean(allResults[i])
    means.append(meanNow)
    stdNow = stats.stdev(allResults[i])
    stdevs.append(stdNow)
    meanAccept.append(1-stats.mean(percentageBlocked[i]))

makeImageArnhem(meanConflictTracks)
print("wait")
```

Amersfoort results

```
from simulationFunction import simulationStationArrivals
from itertools import chain
```

```
from collections import Counter
import matplotlib.pyplot as plt
import statistics as stats
import pandas as pd
from scipy import stats as st
from PFAmersfoort import productFormAmersfoort
import numpy as np

NUM_TRACKS = 23
NUM_SOURCES = 23
TOT_TIME = 1500

arrivalRates = []
handlingRates = []
lines = []
routes =[]
lenLines = len(lines)
df = pd.read_excel('model/sources.xlsx', index_col=0, sheet_name=
    'Amersfoort')
for i in range(1, df.__len__()+1):
    arrivalRates.append(df.loc[i]['arrival␣rate'])
    handlingRates.append(df.loc[i]['handling␣rate'] )
    serialNumber = int(df.loc[i]["serial␣number"])
    thisName = df.loc[i]['name']
    thisLine = str(serialNumber)+"_"+str(thisName)
    thisRoute = [int(i) for i in list(str(df.loc[i]['routes']).
        split(','))]
    lines.append(thisLine)
    routes.append(thisRoute)


result, blockedTracks, pb, conflictTracks =
    simulationStationArrivals(TOT_TIME, arrivalRates,
    handlingRates, NUM_SOURCES, NUM_TRACKS, lines, routes)

allResults =[]
allBlockedTracks = []
percentageBlocked = []
conflictTracksPercentage = []
for i in range(0, NUM_SOURCES+1):
    allResults.append([])
    percentageBlocked.append([])

for i in range(0, NUM_TRACKS+1):
    conflictTracksPercentage.append([])
for i in range(0, 1000):
    result, blockedTracks, pb, conflictTracks =
        simulationStationArrivals(TOT_TIME, arrivalRates,
        handlingRates, NUM_SOURCES, NUM_TRACKS, lines, routes)
    count = 1
```

```
    for r in result:
        allResults[count].append(r)
        count+=1
    for b in blockedTracks:
        allBlockedTracks.append(b)
    c = 1
    for w in pb:
        percentageBlocked[c].append(w)
        c +=1
    c = 0
    for ct in conflictTracks:
        conflictTracksPercentage[c].append(ct)
        c+=1
def countList(list, x):
     return Counter(chain.from_iterable(set(i) for i in list))[x]

tracksBlockedCounter = []
meanConflictTracks = []

for i in range(0, NUM_TRACKS+1):
    tracksBlockedCounter.append(countList(allBlockedTracks, i))
    meanConflictTracks.append(stats.mean(conflictTracksPercentage[
        i]))

productForm, acceptModel = productFormAmersfoort()

first, median, third = np.quantile(tracksBlockedCounter, [0.25,
    0.5, 0.75])
t = []
p = []
pBlock = []
means = []
meanAccept = []

acceptSimu = []
for i in range(1, NUM_SOURCES+1):
    meanNow = stats.mean(allResults[i])
    means.append(meanNow)
    meanAccept.append(1-stats.mean(percentageBlocked[i]))
    tNow, pNow = st.ttest_1samp(a=allResults[i], popmean =
        productForm[i])
    t.append(tNow)
    p.append(pNow)
    t2, p2Now = st.ttest_1samp(percentageBlocked[i], popmean=1-
        acceptModel[i])
    pBlock.append(p2Now)
    plt.figure()
    plt.hist(allResults[i])
    plt.axvline(productForm[i], color='b')
```

```
    plt.axvline((stats.mean(allResults[i])), linestyle='dashed',
        color='b')
    plt.show()
    print("wait")
    plt.close()
sourcesNR = [i for i in range(1, NUM_SOURCES+1)]

print("wait")
```