universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Simulation von Service-Network-Design-Lösungen für die mittlere Meile unter Berücksichtigung stochastischer Einflussfaktoren

verfasst von | submitted by

Johannes Schumann B.A.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von | Supervisor:	emer. o. Univ.-Prof. Dipl.-Ing. Dr. Richard Hartl
Mitbetreut von | Co-Supervisor:	Dr. Yannick Scherr B.Sc. M.Sc.

Abstract

Die zunehmenden Anforderungen an Termintreue und Geschwindigkeit in der Logistik, insbesondere im Kontext von Same-Day Delivery, erfordern innovative Ansätze zur effizienten Gestaltung der mittleren Meile. Ziel dieser Masterarbeit ist die Entwicklung eines stochastischen Simulationsmodells, das Unsicherheiten durch variierende Ablaufzeiten realitätsnah abbildet.

Aufbauend auf einem bestehenden deterministischen Modell wird ein erweitertes stochastisches Simulationsmodell in Python implementiert, das zufallsbedingte Einflussfaktoren sowie gezielte Entscheidungsregeln zur Optimierung logistischer Prozesse integriert.

Im Rahmen der Analyse werden verschiedene Netzwerkvarianten mit unterschiedlichen Konfigurationen simuliert, um belastbare und übertragbare Ergebnisse zu gewinnen. Die Simulationsergebnisse belegen, dass insbesondere die kombinierte Anwendung der entwickelten Entscheidungsmechanismen zu einer signifikanten Steigerung der Liefertreue führt.

Das entworfene Modell liefert damit einen praxisorientierten Beitrag zur flexiblen Planung logistischer Netzwerke unter realitätsnahen Bedingungen und bietet zugleich wertvolle Ansätze für die Weiterentwicklung simulationsbasierter Planungsmethoden.

Inhaltsverzeichnis

Abbildungsverzeichnis V
Tabellenverzeichnis V
Abkürzungsverzeichnis V
1 Einleitung 1
2 Theoretischer Hintergrund 4
2.1 Logistische Netzwerke 5
2.2 Same-Day Delivery 7
2.3 Service Network Design 8
2.3.1 Deterministische Modelle 11
2.3.2 Stochastische Modelle 12
2.3.3 Vergleich der Modelle 14
2.4 Simulation 16
2.4.1 Grundlagen der Simulation 17
2.4.2 Arten von Simulationen 18
3 Methodik 21
3.1 Problembeschreibung 21
3.2 Datenbeschaffung und -aufbereitung 23
3.3 Einsatz von SimPy für die Simulationsdurchführung 25
3.3.1 Grundlagen von SimPy 25
3.3.2 Struktur der Simulation 27
3.3.3 Konzeption der Simulation 29
3.3.4 Ausgabe der Ergebnisse 31
4 Implementierung 32
4.1 Modul zur Datenverarbeitung 32
4.2 Modul zur Simulationsdurchführung 37
4.2.1 Ablauf und Steuerung der Simulationsprozesse 38
4.2.2 Ergebnisaufbereitung und Datenexport 47
4.3 Modul zur Ausführung 51
5 Ergebnisse 53
5.1 Darstellung der Simulationsergebnisse 53
5.2 Auswertung der Simulationsergebnisse 59

6 Analyse 64
6.1 Interpretation der Ergebnisse 64
6.2 Identifikation von Engpässen und Optimierungspotenzialen 66
6.3 Kritische Reflexion des Modells 68
6.4 Bewertung des Modells 70
7 Fazit 71
Literaturverzeichnis 74
Anhang 78

Abbildungsverzeichnis

Abbildung 1: Priorisierung von Lieferfahrzeugen 28
Abbildung 2: Einsatz von Pufferzeiten zur Priorisierung von Lieferfahrzeugen 28

Tabellenverzeichnis

Tabelle 1: Vergleich der Leistungskennzahlen ohne Entscheidungsregeln 54
Tabelle 2: Vergleich mit eingesetzter Pufferzeit von 0,2 Perioden 55
Tabelle 3: Vergleich mit einer zusätzlichen Ladetür 56
Tabelle 4: Vergleich mit zwei zusätzlichen Ladetüren 57
Tabelle 5: Vergleich bei kombiniertem Einsatz der Entscheidungsregeln 58
Tabelle 6: Vergleich im Basismodell unter 30 Minuten Bearbeitungszeit 59
Tabelle 7: Vergleich im Basismodell unter 120 Minuten Bearbeitungszeit 61
Tabelle 8: Vergleich im erweiterten Modell unter 30 Minuten Bearbeitungszeit 62
Tabelle 9: Vergleich im erweiterten Modell unter 120 Minuten Bearbeitungszeit 63

Abkürzungsverzeichnis

BZ ..... Bearbeitungszeit
LZ ..... Lieferzeit
PZ ..... Pufferzeit
WZ ..... Wartezeit
ZT ..... Zusätzliche Tür

1 Einleitung

Die zunehmende Digitalisierung sowie die fortschreitende Globalisierung haben in den vergangenen Jahren den Anspruch an moderne Logistiknetzwerke und Lieferketten grundlegend verändert. Insbesondere die stetig wachsende Erwartungshaltung der Verbraucher hat dazu geführt, dass Konzepte wie Same-Day Delivery nicht mehr als Premiumdienst, sondern zunehmend als Standardoption wahrgenommen werden. Für Logistikunternehmen ergibt sich daraus die Herausforderung, komplexe Anforderungen in Bezug auf Flexibilität, Geschwindigkeit und Zuverlässigkeit zu erfüllen, ohne dabei Abstriche in der Wirtschaftlichkeit und der Effizienz zu verzeichnen.

In Anbetracht dieser Entwicklung gewinnt die effiziente Gestaltung der mittleren Meile, also des Transports von Waren zwischen Distributionszentren und Verteilzentren, zunehmend an Bedeutung. Verzögerungen in diesem Abschnitt wirken sich unmittelbar auf die anschließende Verteilung aus und können zu einer Nichterfüllung der festgelegten Lieferfristen führen, was sich wiederum negativ auf die Kundenzufriedenheit und die Serviceleistung auswirkt. Die optimierte Strukturierung der mittleren Meile stellt somit eine zentrale Voraussetzung für die Sicherstellung termingerechter Lieferungen innerhalb von Logistiknetzwerken dar. Zudem erfordert sie eine hohe Anpassungsfähigkeit an variierende Rahmenbedingungen wie Verkehrsstörungen, Engpässe an Umschlagspunkten oder Nachfrageschwankungen. Vor diesem Hintergrund wird deutlich, dass herkömmliche, rein deterministische Planungsansätze zunehmend an ihre Grenzen stoßen und durch Modelle ergänzt werden müssen, die realitätsnahe Unsicherheiten und Schwankungen berücksichtigen.

Ziel dieser Masterarbeit ist es, die bestehende Lücke zu schließen, indem ein stochastisches Simulationsmodell entwickelt wird, das reale Unsicherheiten und Schwankungen einbezieht. Im Gegensatz zu deterministischen Ansätzen, die auf festen Parametern und Abläufen basieren, sind im stochastischen Modell variierende Transportzeiten und Bearbeitungsdauern an den Umschlagspunkten integriert. In der Realität stellen diese zeitlichen Abweichungen eine zentrale Herausforderung dar, da sie zu Verzögerungen führen und somit die termingerechte Zustellung gefährden. Vor diesem Hintergrund dient die Reduzierung von verspätet eintreffenden Warenlieferungen als wesentliches Bewertungskriterium im entwickelten Modell.

Auf Basis dieser Zielgröße sollen konkrete Optimierungspotenziale identifiziert werden, die zur Steigerung der logistischen Zuverlässigkeit beitragen können. Im Fokus steht dabei die Untersuchung, inwieweit gezielte Anpassungen des Modells eine höhere Liefertreue und eine effizientere Nutzung der verfügbaren logistischen Kapazitäten ermöglichen.

Zur Umsetzung dieser Zielsetzung wird ein umfangreiches Simulationsverfahren als methodischer Ansatz angewendet. Die Grundlage bildet ein bereits vorhandenes deterministisches Modell, das sowohl die strukturellen Gegebenheiten des Netzwerks als auch die Eingangsdaten bereitstellt. Dieses Modell wird systematisch durch stochastische Elemente erweitert, um realitätsnahe Verzögerungen und Unsicherheiten in den Prozess zu integrieren. Für die Implementierung des Simulationsmodells kommt die Programmiersprache Python zum Einsatz, welche sich durch eine umfangreiche Auswahl an geeigneten Bibliotheken sowie durch ihre hohe Flexibilität in der Entwicklungsumgebung auszeichnet.

Innerhalb der Modellierung erfolgen zunächst eine detaillierte Extraktion und Aufbereitung der Eingangsdaten aus dem bestehenden deterministischen Modell. Diese Daten werden im Anschluss an das stochastische Modell übergeben, um die Transport- und Bearbeitungsprozesse unter Berücksichtigung variabler Schwankungen zu simulieren. Aufgrund der zufallsbasierten Abweichungen werden wiederholte Simulationsdurchläufe durchgeführt. Dieses Vorgehen ist erforderlich, um belastbare Durchschnittswerte zu generieren und ein fundiertes Verständnis der Auswirkungen stochastischer Einflüsse auf die Leistungsfähigkeit des Netzwerks zu gewinnen.

Neben dem entwickelten stochastischen Simulationsmodell wird in dieser Arbeit ein erweitertes Modell implementiert, welches zusätzlich zu den stochastischen Einflussfaktoren verschiedene Entscheidungsregeln in den Ablauf integriert. Diese Erweiterung verfolgt das Ziel, den logistischen Prozess gezielt zu optimieren und die termingerechte Zustellung der Waren weiter zu verbessern.

Konkret beinhalten die implementierten Regeln unter anderem eine priorisierte Ressourcenvergabe, eine dynamische Erweiterung der Bearbeitungskapazitäten durch das Öffnen zusätzlicher Ladetüren an den Knotenpunkten sowie die Integration einer definierten Pufferzeit, um verspätet eintreffende Fahrzeuge in der Ressourcenzuteilung zu berücksichtigen. Diese Maßnahmen sollen potenzielle Engpässe im Netzwerk identifizieren und gezielt reduzieren.

Im Rahmen der Analyse werden beide Modellvarianten in unterschiedlichen Logistiknetzwerken unter verschiedenen Szenarien simuliert und miteinander verglichen. Dabei werden die einzelnen Optimierungsmaßnahmen sowohl isoliert als auch in Kombination hinsichtlich ihrer Wirksamkeit untersucht und anschließend bewertet, um deren Einfluss auf definierte Leistungskennzahlen darzustellen.

Die Struktur dieser Masterarbeit gliedert sich in mehrere aufeinander aufbauende Kapitel, die von theoretischen Grundlagen bis hin zur Ergebnisinterpretation und abschließenden Bewertung führen. Nach der Einleitung folgt im zweiten Kapitel der theoretische Hintergrund, in dem die Grundlagen für das Verständnis der Arbeit gelegt werden. In diesem Zusammenhang werden zentrale Begriffe und Konzepte erläutert, darunter logistische Netzwerke, das Konzept der Same-Day Delivery, Service Network Design sowie deterministische und stochastische Modellansätze. Darüber hinaus wird die Simulation näher dargestellt, mit besonderem Augenmerk auf die verschiedenen Simulationsarten und deren spezifische Anwendungsbereiche.

Das dritte Kapitel widmet sich der methodischen Vorgehensweise. Eingangs wird das zugrunde liegende Problem detailliert beschrieben und der Bezug zu praktischen Herausforderungen hergestellt. Im Anschluss erfolgt die Darstellung der eingesetzten Methodik, welche sich aus mehreren funktional getrennten Modulen zusammensetzt. Diese Module werden mit Fokus auf ihre jeweiligen Aufgabenbereiche wie Datenverarbeitung, Simulationssteuerung und Ausführung in ihren grundlegenden Prinzipien erläutert. Ein besonderer Fokus liegt dabei auf der Verwendung der Programmiersprache Python und der verwendeten Simulationsbibliothek, die zur Modellierung des Systems und zur Abbildung der stochastischen Prozesse herangezogen wurde.

Im vierten Kapitel steht die technische Implementierung des entwickelten Simulationsmodells im Mittelpunkt. Dabei wird die Struktur der Programmarchitektur ausführlich erläutert, wobei insbesondere auf die Funktionsweise der einzelnen Module innerhalb der verwendeten Programmierumgebung eingegangen wird. Neben der Darstellung des Codeaufbaus werden ebenso die eingesetzten Funktionen und Logiken beschrieben, die zur Modellierung des Logistiknetzwerks, zur Implementierung stochastischer Einflussfaktoren sowie zur Durchführung der Simulationsabläufe verwendet wurden.

Das fünfte Kapitel beinhaltet die Präsentation der Simulationsergebnisse. In diesem Abschnitt werden die Resultate des stochastischen Basismodells mit den Ergebnissen des

erweiterten Modells gegenübergestellt. Dabei wird zwischen verschiedenen Simulationskonfigurationen unterschieden, um die Auswirkungen der implementierten Entscheidungsregeln sowie deren Kombination auf das logistische Netzwerk zu analysieren. Die Ergebnisse werden dafür sowohl tabellarisch als auch textlich aufbereitet, um eine detaillierte und vergleichende Betrachtung zu ermöglichen.
Abschließend folgen im sechsten Kapitel eine vertiefende Analyse und eine Interpretation der gewonnenen Ergebnisse. Ziel dieses Abschnitts ist es, Engpässe innerhalb des logistischen Netzwerks zu identifizieren und daraus bestehende Optimierungspotenziale aufzuzeigen. Dabei werden neben den Auswirkungen der stochastischen Schwankungen auch die Effekte der eingesetzten Optimierungsmaßnahmen kritisch hinterfragt. Zusätzlich wird überprüft, inwieweit die gesetzten Zielgrößen durch die gewählten Maßnahmen erreicht werden konnten. Somit bildet die Analyse die abschließende Bewertung der Modelle und liefert konkrete und zukünftige Handlungsempfehlungen.

2 Theoretischer Hintergrund

Das vorliegende Kapitel enthält die theoretischen Grundlagen, die für das Verständnis der im weiteren Verlauf dargestellten Untersuchungen relevant sind. Im Fokus steht die Betrachtung logistischer Systeme, die als Grundlage für die Modellierung und Optimierung von Transportnetzwerken von zentraler Bedeutung sind.
Zu Beginn wird das Konzept logistischer Netzwerke erläutert, wobei wesentliche Elemente, Strukturen und deren funktionale Zusammenhänge dargestellt werden. Im Anschluss folgt eine Betrachtung des Same-Day-Delivery-Ansatzes, der durch seine hohen Anforderungen an die Flexibilität und Geschwindigkeit neue Herausforderungen für die logistische Planung mit sich bringt. Darauf aufbauend werden die Grundlagen des Service Network Designs vorgestellt, das sich insbesondere mit der operativen Umsetzung und Steuerung von Transportnetzwerken befasst. In diesem Zusammenhang erfolgt zudem eine vertiefende und vergleichende Auseinandersetzung von deterministischen und stochastischen Modellierungsansätzen, die zur Abbildung logistischer Prozesse unter Berücksichtigung von Unsicherheiten genutzt werden. Abschließend wird das Konzept der Simulation behandelt. Dieses dient dazu, reale Prozesse mithilfe geeigneter Modelle nachzubilden, um komplexe logistische Zusammenhänge analysieren und bewerten zu können.

2.1 Logistische Netzwerke

Im Allgemeinen versteht man unter logistischen Netzwerken ein strukturiertes Datenflusssystem, in dem Ressourcenknoten sowohl hierarchisch als auch geografisch organisiert sind und durch Transport- und Informationsflüsse miteinander in Verbindung stehen. Innerhalb dieser Netzwerke werden Quellen, wie beispielsweise Produktionsstätten, mit den Senken, also Orten des endgültigen Bedarfs an Gütern, verbunden. Das zentrale Ziel besteht darin, wirtschaftlich effiziente und bedarfsgerechte Versorgungsketten zu schaffen. Diese umfassen sowohl die physische Bewegung von Waren als auch die dazugehörigen Informationsprozesse (Tripp, 2021). Darüber hinaus bezeichnet man logistische Netzwerke als Gesamtheit von Transport-, Umschlags- und Lagerprozessen sowie den damit verbundenen Informationsflüssen, die innerhalb einer Raum-Zeit-Umgebung ablaufen (Piontek, 1997).

Die zentralen Elemente logistischer Netzwerke lassen sich in Knoten und Kanten unterteilen. Knoten repräsentieren dabei die logistischen Funktionen wie den Wareneingang, die Lagerhaltung oder den Versand (Tripp, 2021). Sie bilden die Start- und Endpunkte von Warenströmen und umfassen Depots, Terminals und weitere Sammelpunkte (Piontek, 1997). Die Effizienz des gesamten Systems hängt wesentlich von der Positionierung, der Anzahl sowie der Ausstattung dieser Knoten ab. Die Verbindung zwischen den Knoten erfolgt über Kanten, über die materielle und immaterielle Austauschbeziehungen stattfinden. Die effiziente Gestaltung dieser Kanten ist ein maßgeblicher Faktor in der Kostenoptimierung und beeinflusst maßgeblich die Geschwindigkeit der Transportvorgänge (Tripp, 2021).

In der Praxis existieren verschiedene Netzwerkstrukturen. Eine Form stellt das Rastermodell dar, bei dem jeder Knoten direkt mit allen anderen verbunden ist. Eine alternative Struktur ist das Ein-Nabe-System, in dem alle Knoten sternförmig mit einem zentralen Hub verknüpft sind. Während vollständig miteinander verbundene Strukturen eine hohe Flexibilität und schnelle Reaktionszeiten gewährleisten, sind sie mit erheblichen Kosten verbunden. Im Gegensatz dazu ist ein zentraler Knoten kosteneffizienter, jedoch oftmals mit eingeschränkter Flexibilität verbunden. In den meisten Fällen erweist sich eine stufige Netzwerkgestaltung als vorteilhaft, die sowohl lokale und regionale Umschlagspunkte als auch interkontinentale Verbindungen berücksichtigt. Dadurch können Warenströme effizient gebündelt und Depots bedarfsgerecht geplant werden. Wichtige Kriterien für eine optimale Depotplanung sind

unter anderem das prognostizierte Warenaufkommen, die spezifischen Eigenschaften der Güter, die Marktanforderungen sowie die bestehende Verkehrsinfrastruktur (Piontek, 1997).

Die Konfiguration logistischer Netzwerke hängt nicht ausschließlich von den physischen Transportwegen ab, sondern muss gleichermaßen den Informationsfluss berücksichtigen. Optimierungsansätze fokussieren sich häufig auf die Reduktion von Transport-, Lager- und Verwaltungskosten sowie auf die Gewährleistung einer bedarfsgerechten Lieferfähigkeit. Um ein ausgewogenes Verhältnis zwischen maximaler Serviceleistung für den Kunden und minimalen Transportkosten zu erzielen, kommen Quellen-Senken-Analysen zum Einsatz. Dabei werden Aspekte wie zurückgelegte Entfernungen, Transportfrequenzen, Standortbedingungen, transportierte Mengen und einzuhaltende Lieferzeitfenster untersucht, um den optimalen Standort zu ermitteln. Darüber hinaus gewinnen Nachhaltigkeitsaspekte zunehmend an Bedeutung. Der emissionsarme Einsatz von Transportmitteln, die Vermeidung ungenutzter Kapazitäten und die Optimierung von Routen stellen grundlegende Strategien zur Reduktion der Umweltbelastung dar (Tripp, 2021).

Unter Berücksichtigung dieser Faktoren werden moderne Netzwerke verstärkt unter Einhaltung ökologischer Zielsetzungen gestaltet, wie beispielsweise durch Minimierung von Umweltauswirkungen und Kosten. Methoden wie das Multi-Objective Programming ermöglichen es, sowohl ökonomische als auch ökologische Faktoren zu berücksichtigen. Dabei werden Kompromisse zwischen Profitabilität und Umweltbelastungen in mathematischen Modellen systematisch abgewogen, um effiziente Lösungen zu entwickeln. Dennoch treten häufig Zielkonflikte auf, da nachhaltige Konzepte oft mit erhöhten Investitionen verbunden sind und nicht immer wirtschaftlich rentabel erscheinen (Frota Neto et al., 2008).

Besonders vor dem Hintergrund globalisierter Produktions- und Handelsströme und durch digitale Transformationsprozesse steigt die Relevanz logistischer Netzwerke. Globale Wertschöpfungsketten erfordern flexible, widerstandsfähige und häufig mehrstufig strukturierte Netzwerke, die in der Lage sind, auf schwankende Nachfrage und Störungen effizient zu reagieren (Piontek, 1997). Die fortschreitende Digitalisierung bietet hierbei den Vorteil, eine kontinuierliche Erfassung von Echtzeitinformationen zu ermöglichen, wodurch schnellere Reaktionszeiten, präzisere Planungen und teilweise sogar Automatisierungsschritte ermöglicht werden. Neben Wirtschaftlichkeit und Flexibilität rückt dadurch auch die Serviceorientierung gegenüber dem Endkunden

verstärkt in den Fokus, was Unternehmen strategische Wettbewerbsvorteile verschaffen kann (Tripp, 2021).

Zusätzlich erfordert die Gestaltung logistischer Netzwerke robuste Strukturen, um Störungen wie Stromausfällen, Naturkatastrophen oder Streiks entgegenzuwirken. Ziel ist es, die Kosten im Normalbetrieb zu minimieren und parallel eine ausreichende Absicherung gegen Ausfälle durch Redundanzen oder alternative Transportwege zu gewährleisten. Für dieses Vorgehen könnten zusätzliche Umschlagterminals als Backup dienen, um im Falle einer möglichen Störung zumindest einen Teil der Lieferströme aufrechtzuerhalten. Während solche Maßnahmen im regulären Betrieb mit leicht erhöhten Kosten verbunden sind, tragen sie dazu bei, schwerwiegende Folgen bei plötzlichen Unterbrechungen zu vermeiden. In der Gestaltung logistischer Netzwerke stellt somit die Balance zwischen Kostenoptimierung und Risikomanagement eine essenzielle Strategie dar (Peng et al., 2011).

2.2 Same-Day Delivery

Same-Day Delivery beschreibt eine logistische Dienstleistung, bei der Bestellungen noch am Tag der Aufgabe beim Empfänger zugestellt werden (Wu et al., 2023). Somit kann die Wartezeit zwischen der Auftragserteilung durch den Kunden und dem Empfang der Lieferung auf ein Minimum reduziert werden (Dayarian & Savelsbergh, 2020; Voccia et al., 2019). Ziel des Konzepts ist es, eine nahezu unmittelbare Verfügbarkeit von Produkten zu garantieren, indem Aufträge innerhalb eines enorm kurzen Zeitfensters bearbeitet werden (Ulmer, 2020; Voccia et al., 2019). Im Gegensatz zu herkömmlichen Lieferkonzepten, bei denen die Lieferungen über längere Zeiträume hinweg geplant werden, pflegt das der Same-Day Delivery kontinuierlich neue Aufträge ein. Dies erfordert eine besonders dynamische Planung, die in Echtzeit auf dynamische Bestellnachfragen reagieren kann. Zudem müssen variable Kapazitäten der Transportmittel berücksichtigt werden, um eine termingerechte Auslieferung sicherzustellen (Dayarian & Savelsbergh, 2020; Klapp et al., 2018; Voccia et al., 2019).

In der praktischen Umsetzung können verschiedene Same-Day Delivery-Modelle zum Einsatz kommen. Zum einen gibt es die Option der Expresszustellung, bei der Bestellungen innerhalb kurzer Zeiträume nach Auftragseingang ausgeliefert werden. Andererseits existieren terminierte Zustellungen, bei denen Kunden die Möglichkeit erhalten, die Lieferung innerhalb eines vordefinierten Zeitfensters zu planen. Diese

Flexibilität erscheint aus Kundensicht vorteilhaft, stellt jedoch für die Logistikdienstleister erhebliche Herausforderungen bereit, da eine präzise und optimale Planung der Routen und Ressourcen erforderlich ist. Während die kürzesten und sinnvollsten Transportwege ermittelt werden, muss zeitgleich die Wirtschaftlichkeit berücksichtigt werden, um eine möglichst effiziente Nutzung der Transportmittel sicherzustellen (Ulmer, 2020).

Gleichzeitig birgt dieses Konzept erhebliches Potenzial für betriebswirtschaftliche Herausforderungen, insbesondere in Hinsicht auf die entstehenden Kosten. Das Modell der Same-Day Delivery ist auf schnelle Reaktionszeiten, eine ausreichende Kapazität der Transportflotte sowie deren ständige Verfügbarkeit angewiesen, was im Vergleich zu konventionellen Lieferkonzepten mit erheblich höheren Investitionen verbunden ist (Ulmer & Streng, 2019; Ulmer & Thomas, 2018). Aufgrund der dynamischen Besonderheit der Zustellungen kann die Zahl der eingehenden Aufträge im Tagesverlauf erheblichen Schwankungen unterliegen. Daher ist eine kontinuierliche Anpassung der Auslieferungsstrategie erforderlich, um sowohl die Betriebskosten zu optimieren als auch eine termingerechte Zustellung am selben Tag zuverlässig sicherzustellen. Zu diesem Zweck sind Echtzeit-Analysen sowie eine strategische Planungsweise essenziell, um dieses Konzept langfristig und wirtschaftlich effizient umzusetzen (Klapp et al., 2018; Voccia et al., 2019).

2.3 Service Network Design

Unter dem Begriff Service Network Design versteht man die strategische Planung, Modellierung und Optimierung von Transport- und Dienstleistungsnetzwerken in verschiedenen Anwendungsbereichen (Wieberneit, 2007). Insbesondere im logistischen Kontext konzentriert sich das Service Network Design auf die Entwicklung und Steuerung eines Systems, das den Transport von Gütern von einem definierten Ausgangspunkt mittels geeigneter Transportmittel bis hin zum finalen Zielpunkt effizient gestaltet. Dieser Ansatz erfordert komplexe Entscheidungsprozesse die unter anderem die zeitliche Koordination der Transporte, die Erstellung von detaillierten Fahr- und Routenplänen sowie die optimale Zuweisung von Ressourcen wie Fahrzeugen oder Kapazitäten umfassen. Ein besonderer Fokus liegt dabei auf der Maximierung der Effizienz, der Reduktion von Umweltbelastungen und der Verkehrsentlastung (Crainic et al., 2009).

Besonders im Bereich konsolidierungsbasierter Transportnetzwerke ist das Service Network Design von zentraler Bedeutung. Konsolidierung bezeichnet dabei die Bündelung mehrerer einzelner Sendungen zu einer gemeinsamen Transporteinheit, mit dem Ziel, Skaleneffekte zu nutzen und gleichzeitig ein hohes Serviceniveau sicherzustellen. Voraussetzung für eine erfolgreiche Umsetzung ist eine präzise Planung, um Sendungen innerhalb definierter Zeitfenster an zentrale Hubs zu befördern, an denen diese zusammengeführt und gemeinsam weitertransportiert werden können. Anwendungsbereiche konsolidierungsbasierter Netzwerke sind typischerweise unter anderem der Paketdienst sowie der Schienengüterverkehr (Crainic, 2021).

Grundsätzlich ist ein Service Network Design in strategische, taktische und operative Planungsebenen zu unterteilen, die im Bereich von Frachttransportnetzwerken relevant sind. Die strategische Planung konzentriert sich auf langfristige Entscheidungen, die von der obersten Führungsebene getroffen werden. Diese haben speziell Einfluss auf die grundlegende Struktur des Transportnetzwerkes, die Standortauswahl, die Ressourcenverteilung und die Entwicklung der Infrastruktur. Darüber hinaus werden Tarifstrukturen und Kundendienstarten definiert, mit dem Ziel, eine möglichst hohe Servicequalität bei optimierter Kostenstruktur zu gewährleisten. Auf mittelfristiger Ebene erfolgt die taktische Planung, die besonders auf die effiziente Nutzung vorhandener Ressourcen abzielt. Das Transportsystem wird dabei in verschiedene Sektoren unterteilt. Ein Bereich umfasst den regionalen Verkehr, der die Organisation der Abhol- und Zustellregionen steuert. Ein weiterer Bereich konzentriert sich auf die Langstreckenverteilung, die für die übergeordnete Strukturierung des Servicenetzwerks verantwortlich ist. Die gesamte Planung kann jedoch nur mit hoher Sensitivität erfolgen, da sie auf der Analyse historischer Daten oder Nachfrageprognosen basiert. Die operative Planung hingegen befasst sich mit der kurzfristigen und dezentralen Steuerung logistischer Prozesse. Aufgrund der dynamischen Natur des operativen Umfelds können Aufträge kurzfristig eingehen oder Änderungen unterliegen. Zudem können die Zeitfenster für Abholungen und Zustellungen variieren. Daher ist eine kontinuierliche Anpassung der taktischen Planung an die aktuellen Bedingungen erforderlich. Dies umfasst unter anderem die fortlaufende Steuerung von Fahrplänen, die Koordination von Wartungsaktivitäten sowie die Einteilung von Crews, um einen reibungslosen Betriebsablauf zu gewährleisten (Wieberneit, 2007).

Ein besonders relevanter Bestandteil des Service Network Designs ist die sogenannte „mittlere Meile“. Darunter versteht man den Transport von Waren zwischen

Distributionszentren und weiterführenden Lieferstationen. In dieser Phase werden die Waren, sobald eine Bestellung eingegangen ist, durch spezialisierte Lieferfahrzeuge von den zentralen Lagerhäusern zu den dezentralen Verteilstationen transportiert. Daraufhin erfolgt die Verteilung an den Endkunden durch die Fahrzeuge der „letzten Meile“. Das Hauptziel der mittleren Meile besteht darin, eine termingerechte und zuverlässige Lieferung, in der Regel spätestens bis zum Folgetag, sicherzustellen, sodass die nachgelagerten Lieferprozesse der letzten Meile reibungslos ausgeführt werden können. Dieses Vorgehen unterliegt jedoch zahlreichen operativen Einschränkungen, wie beispielsweise den Kapazitätsgrenzen der eingesetzten Fahrzeuge oder variablen Transportzeiten. Zur Bewältigung dieser Herausforderungen im Netzwerk ist der Einsatz fortschrittlicher Optimierungsalgorithmen erforderlich, da diese Art von Planungsproblem typischerweise ein NP-hartes Problem darstellt (Benidis et al., 2023).

Eine wesentliche Vorgehensweise zur Modellierung zeitabhängiger Abläufe in Service-Netzwerken ist die Verwendung sogenannter „Time-Expanded Networks“. Dabei wird der Planungszeitraum in gleichlange und diskrete Intervalle unterteilt, wodurch die Zeitabhängigkeiten explizit in der Netzwerkstruktur berücksichtigt werden. Diese Art der Modellierung erlaubt eine detaillierte Steuerung von Abfahrts- und Ankunftszeiten und ermöglicht somit eine realitätsnahe Planung logistischer Prozesse. Die Wahl der Länge dieser Zeitintervalle hat jedoch erheblichen Einfluss auf die Lösungsgenerierung. Kürzere Intervalle ermöglichen eine höhere Modellgenauigkeit, führen jedoch zu einem starken Anstieg der Modellkomplexität. Gröbere Intervalle hingegen erleichtern die Berechnung, können jedoch zu einer geringeren Qualität der Lösung führen. Ursache hierfür ist insbesondere die eingeschränkte Möglichkeit Sendungen innerhalb enger Zeitfenster zu konsolidieren (Boland et al., 2019).

In der Praxis von Service Network Design-Projekten ist eine präzise Modellierung der Netzwerkstruktur erforderlich, die aus Transportwegen, Hubs und Terminals besteht. Das Ziel ist es, die Gesamtkosten des Transports zu reduzieren und gleichzeitig eine hohe Servicequalität sowie eine hohe Zuverlässigkeit zu gewährleisten. Hierbei kommen häufig mathematische Optimierungsmodelle zum Einsatz, insbesondere gemischt-ganzzahlige Netzwerkdesign-Formulierungen, um die Entscheidungsfindung hinsichtlich der Positionierung von Knotenpunkten, der Kapazitätszuweisung und der effizienten Steuerung von Fahrzeugbewegungen zu unterstützen (Crainic, 2000).

Ein weiteres zentrales Merkmal eines erfolgreichen Service Network Designs ist dessen Ausrichtung auf Robustheit und Flexibilität gegenüber Nachfrageschwankungen sowie

externen Unsicherheiten. Um diesen Herausforderungen zu begegnen und das Netzwerk anpassungsfähig zu gestalten, werden zunehmend stochastische Modelle entwickelt, die variable Parameter in der Modellierung einbeziehen. Der Einsatz solcher Modelle ermöglicht eine flexiblere Netzwerkstruktur, die eine effektivere Konsolidierungsstrategie bietet, als es mit einem deterministischen Modell realisierbar wäre (Lium et al., 2009).

2.3.1 Deterministische Modelle

Deterministische Modelle zeichnen sich durch die grundlegende Eigenschaft aus, dass sie ausschließlich bekannte Parameter und Variablen verwenden, ohne Zufallselemente oder stochastische Komponenten zu beachten (Dinkelbach & Lorscheider, 1994). Diese Modelle beruhen auf festen mathematischen Zusammenhängen zwischen den Modellvariablen, sodass für jede definierte Eingabe ein eindeutiger und reproduzierbarer Ausgangswert erfolgt (Uusitalo et al., 2015).

Bei formalen Entscheidungsproblemen lassen sich deterministische Modelle durch eine definierte Zielfunktion sowie eine Menge möglicher Alternativen beschreiben. Die Ermittlung von optimalen Lösungen erfolgt dabei über eine Extremierungsregel, die je nach spezifischer Problemstellung entweder eine Maximierungs- oder Minimierungsstrategie umfasst (Dinkelbach & Lorscheider, 1994). Dieses Vorgehen wird insbesondere in der Unternehmensforschung genutzt, wie beispielsweise in der Lagerhaltung oder Produktionsplanung. In diesen Bereichen kommen deterministische Modelle zum Einsatz, um den wirtschaftlich optimalen Nachschub- und Produktionszyklus zu ermitteln, wobei von einer bekannten und konstanten Nachfrage ausgegangen wird. Die Modellierung der Szenarien verwendet in der Regel lineare oder nichtlineare Programmierungsansätze, bei denen sämtliche Eingabeparameter, wie beispielsweise Produktionskapazitäten, als sicher und unverändert angenommen werden (Hillier et al., 2002).

Neben klassisch wirtschaftswissenschaftlichen Fragestellungen finden deterministische Modelle auch in der Entscheidungsunterstützung von Ressourcen- und Umweltsystemen Anwendung. Beispielsweise können sie zur Simulation der Auswirkungen unterschiedlicher Managementstrategien im Bereich der nachhaltigen Nutzung natürlicher Ressourcen oder des Erhalts der Biodiversität eingesetzt werden. Dabei werden die Interaktionen von eindeutig definierten Einflussgrößen modelliert. Jedoch

werden dabei keine Unsicherheiten, die durch Messfehler, natürliche Schwankungen oder Modellbeschränkungen entstehen können, berücksichtigt. Aus diesem Grund ist es häufig erforderlich, ergänzende Sensitivitätsanalysen oder externe Unsicherheitsbetrachtungen durchzuführen, um robuste Entscheidungen zu ermöglichen (Uusitalo et al., 2015). Ein weiteres Anwendungsbeispiel deterministischer Modelle ist das Verkehrswesen. In diesem Kontext werden sie zur Untersuchung von Gleichgewichtszuständen und optimalen Lösungen genutzt. Beispielsweise kann der zeitliche Ablauf von Engpasssituationen mathematisch beschrieben werden, indem Parameter wie Straßenkapazität, Anzahl der Verkehrsteilnehmer oder Kosten für Wartezeiten exakt definiert werden. Die daraus resultierenden Modelle ermöglichen es, die Auswirkungen bestimmter Entscheidungen von Akteuren innerhalb der Verkehrssysteme zu simulieren und potenzielle Optimierungsmaßnahmen zu identifizieren (Hendrickson & Kocur, 1981).

Ein wesentliches Merkmal deterministischer Modelle besteht in der klaren und mathematischen Struktur, wodurch sie eine exakte Bestimmung von Ergebnissen ermöglichen, im Gegensatz zu stochastischen Modellen, in denen Wahrscheinlichkeitsverteilungen berücksichtigt werden. Zudem sind diese Modelle besonders für Szenarien geeignet, in denen alle relevanten Parameter stabil und quantifizierbar sind und keinen Zufallsschwankungen unterliegen. Dennoch stellt die fehlende Berücksichtigung von Unsicherheiten eine Herausforderung und Limitation der deterministischen Modellansätze dar. Sobald die definierten Parameter erheblich von der realen Situation abweichen, kann dies in der Praxis zu suboptimalen Ergebnissen führen. Um diese Problematik zu reduzieren, wird oftmals die Durchführung einer Sensitivitätsanalyse empfohlen. Als Alternative können erweiterte Modellvarianten herangezogen werden, in denen explizite Unsicherheitsfaktoren oder realitätsnahe Beschränkungen integriert werden (Kalayci et al., 2019).

2.3.2 Stochastische Modelle

Stochastische Modelle sind mathematische Systeme, in denen Unsicherheiten und Zufälligkeiten durch Wahrscheinlichkeitsverteilungen erfasst werden, anstatt ausschließlich mit deterministischen Eingabedaten zu operieren (Bernardo & Hillston, 2007). Insbesondere zeichnen sich diese Ansätze dadurch aus, dass sie sowohl zufällige Parameter als auch mögliche Messfehler und Modellunsicherheiten innerhalb des

Systems berücksichtigen. Dadurch resultiert nicht lediglich ein einzelnes Ergebnis, sondern eine gesamte Verteilung potenzieller Ergebnisse (Bi et al., 2023). So ist eine realistischere und präzisere Abbildung vieler praktischer Problemstellungen möglich, wie beispielsweise in den Bereichen des Lieferketten- oder Risikomanagements, der Lagerhaltung oder der Leistungsevaluation von Computernetzwerken (Bernardo & Hillston, 2007).

Im Bereich der Modellanpassung besteht ein wesentliches Ziel stochastischer Ansätze darin, Unsicherheiten systematisch zu quantifizieren und gegebenenfalls zu reduzieren. Hierbei werden Wahrscheinlichkeiten genutzt, um beobachtete Daten mit den Ergebnissen des Modells abzugleichen. Im Gegensatz zu deterministischen Optimierungsverfahren, die eine einzige optimale Lösung ausgeben, ermöglichen stochastische Methoden robustere und flexiblere Vorhersagen. Grund dafür ist, dass sie nicht nur physikalische und technische Eigenschaften berücksichtigen, sondern auch epistemische und aleatorische Unsicherheiten in ihre Berechnungen einbinden. Epistemische Unsicherheiten resultieren aus unvollständigem Wissen und können durch zusätzliche Informationen potenziell reduziert werden. Aleatorische Unsicherheiten hingegen beruhen auf zufälligen Schwankungen, die nicht vollständig eliminiert werden können (Bi et al., 2023).

Zu den häufig eingesetzten Modellierungsansätzen in stochastischen Optimierungsverfahren zählen unter anderem das Konzept der sogenannten „Chance-Constraints“ sowie das „Two-Stage-Stochastic-Programming“. Beim Chance-Constraint-Ansatz stellen Wahrscheinlichkeitsbeschränkungen sicher, dass Zustands- und Ausgangsvariablen mit einer vorab definierten Wahrscheinlichkeit im akzeptablen Bereich bleiben. Zudem besteht die Möglichkeit, Unsicherheiten wahrscheinlichkeitsbasiert zu berücksichtigen und einen Ausgleich zwischen der Zielerreichung und der Einhaltung von Einschränkungen zu finden. Allerdings ergeben sich bei der Implementierung des Ansatzes besondere Herausforderungen, da die resultierenden Optimierungsprobleme häufig nicht konvex sind und die korrekte Berücksichtigung der Unsicherheiten eine hohe Komplexität aufweisen kann. Two-Stage-Stochastic-Programming hingegen ist in zwei Phasen unterteilt. In der ersten Phase werden Entscheidungen getroffen, noch bevor Unsicherheiten realisiert werden, während in der zweiten Phase Anpassungen auf Basis der tatsächlich beobachteten Unsicherheiten vorgenommen werden. Durch dieses Vorgehen können verschiedene Unsicherheiten

bereits im Vorfeld durch szenariobasierte Optimierungsalgorithmen berücksichtigt werden (Mesbah, 2016).

Auch in der Statistik finden stochastische Modelle Anwendung, insbesondere bei der Analyse komplexer Systeme mit einem hohen Zufallsanteil. Während statistische Modelle die Zusammenhänge zwischen Variablen durch feste Verteilungen beschreiben, werden in stochastischen Simulationen die Zufallsprozesse direkt in den Ablauf integriert. Dabei werden durch den wiederholten Einsatz von Simulationsabläufen Wahrscheinlichkeitsverteilungen der möglichen Ausgänge erzeugt. Besonders vorteilhaft erweist sich dieses Vorgehen, wenn die Wahrscheinlichkeiten bestimmter Parameter der beobachteten Daten innerhalb des statistischen Modells nicht explizit bekannt sind (Hartig et al., 2011).

Eine weitere Anwendungsmöglichkeit stochastischer Modelle stellt das Management globaler Lieferketten dar. Durch den Einsatz solcher Modelle besteht die Möglichkeit verschiedene Unsicherheitsfaktoren wie Nachfragerisiken, Störungen oder Wechselkursschwankungen zu berücksichtigen. Anhand der stochastischen Programme werden verschiedene Szenarien für Angebot und Nachfrage simuliert, um Entscheidungsregeln abzuleiten. Diese bieten beispielsweise die Möglichkeit, Produktionsmengen, Standorte oder Transportstrategien dynamisch an sich verändernde Rahmenbedingungen anzupassen. Außerdem können Unternehmen durch die Anwendung stochastischer Modelle ihre Risikofaktoren quantifizieren und strategische Handlungsoptionen entwickeln, wodurch sowohl Kosten minimiert als auch Gewinne stabilisiert werden können (Goh et al., 2007).

2.3.3 Vergleich der Modelle

Der wesentliche Unterschied zwischen deterministischen und stochastischen Modellen liegt in ihrem Umgang mit Unsicherheiten und deren Auswirkungen auf die Modellierung realer Systeme. Deterministische Modelle beruhen auf der Annahme, dass alle relevanten Variablen, Parameter und Zusammenhänge exakt bekannt und unveränderlich sind. Dadurch liefert eine bestimmte Eingabe einen eindeutigen und reproduzierbaren Ergebniswert. Da deterministische Modelle klar strukturiert und mathematisch formuliert sind, ermöglichen sie eine präzise Berechnung. Besonders vorteilhaft sind diese Modelle in stabilen Umgebungen, in denen die relevanten Faktoren genau und sicher bestimmt werden können.

Ein wesentliches Problem der Modelle besteht jedoch darin, dass Unsicherheiten, die durch Schwankungen oder Messfehler eintreten können, nicht automatisch berücksichtigt werden. Dies kann dazu führen, dass die festgelegten Parameter in der Praxis von den tatsächlichen Gegebenheiten abweichen, wodurch suboptimale Ergebnisse folgen. Daher werden in realen Anwendungsszenarien zusätzlich Sensitivitätsanalysen oder erweiterte Modelle eingesetzt, die Unsicherheitsfaktoren einbeziehen, um die reale Situation exakter abzubilden.

Stochastische Modelle hingegen integrieren von Beginn an Unsicherheiten und Zufälligkeiten, indem sie Wahrscheinlichkeitsverteilungen anstelle von festgelegten Eingabedaten verwenden. So entsteht nicht ein einzelner und reproduzierbarer Ausgang, sondern eine Verteilung möglicher Ergebnisse mit unterschiedlicher Wahrscheinlichkeit. Dieses Vorgehen trägt dazu bei, dass stochastische Modelle oftmals eine realitätsnähere Darstellung vieler praktischer Problemstellungen ermöglichen, insbesondere wenn bestimmte Parameter zufälligen Schwankungen unterliegen oder nicht mit vollständiger Sicherheit bekannt sind. Sie erlauben beispielsweise eine flexible Anpassung an schwankende Nachfragebedingungen und können somit zu robusteren und flexibleren Entscheidungen beitragen.

Jedoch ist der hohe mathematische und rechnerische Aufwand stochastischer Modelle ein wesentlicher Nachteil. Die Durchführung der komplexen Berechnungen mit Wahrscheinlichkeitsverteilungen erfordert leistungsfähige Simulationsverfahren oder erweiterte analytische Methoden, was die Nutzung in der Praxis erheblich erschwert.

Zusammenfassend lässt sich festhalten, dass die Wahl des passenden Modellierungsansatzes maßgeblich von der jeweiligen Problemstellung abhängt. Während deterministische Modelle insbesondere in stabilen und gut definierten Umgebungen präzise Ergebnisse liefern, sind stochastische Modelle für Anwendungsfälle geeignet, in denen Unsicherheiten und Schwankungen explizit berücksichtigt werden müssen. Aus diesem Grund besitzen die verschiedenen Modelltypen abhängig von den jeweiligen Anforderungen spezifische Stärken und Schwächen.

2.4 Simulation

In wissenschaftlichen und technischen Zusammenhängen bezeichnet der Begriff der Simulation die methodische Nachbildung realer Prozesse oder Systeme, mit dem Ziel, deren Verhalten zu analysieren, zu verstehen und vorherzusagen (Roth, 2018). In der Forschung und Technik dient die Simulation dazu, reale Vorgänge zu untersuchen, ohne diese unmittelbar in der physischen Realität ausführen zu müssen. Besonders für große Systeme ist dies relevant, da sie mithilfe von Simulationen einfach und effizient analysiert werden können, ohne dass physische Untersuchungen erforderlich sind (Bandyopadhyay & Bhattacharya, 2014). Eine Simulation zeichnet sich durch die Fähigkeit aus, ein bestehendes System oder einen spezifischen Prozess durch ein alternatives Modell abbilden oder imitieren zu können. Diese Modelle können beispielsweise durch mathematische Formulierungen oder Computerprogramme umgesetzt werden. Die Wahl einer geeigneten Simulationsmethode hängt dabei von der Verfügbarkeit der analytischen Lösungen ab. Sind diese nicht oder nur unter erheblichem Aufwand verfügbar, werden häufig leistungsfähige Rechenmethoden eingesetzt, um die relevanten Parameter zu berechnen (Schweer et al., 2024).

Das primäre Ziel der Simulation besteht darin, Erkenntnisse über das Verhalten des realen Systems zu gewinnen, um daraus Zusammenhänge zu identifizieren, ohne reale Experimente durchzuführen oder aktiv in die Wirklichkeit einzugreifen (Altiok & Melamed, 2010). Dafür erfolgt eine Simulation in aufeinander abgestimmten Schritten. Zunächst wird ein geeignetes Modell ausgewählt oder neu entwickelt, welches die wesentlichen Eigenschaften des realen Systems beinhaltet. Anschließend wird dieses Modell experimentell untersucht, indem verschiedene Parameterkonfigurationen simuliert und die resultierenden Ergebnisse analysiert werden, um Rückschlüsse auf das reale System zu ermöglichen (Schweer et al., 2024). Dabei erfolgt die Nachbildung des realen Systems in einer kontrollierten Umgebung, wodurch unterschiedliche Einflussfaktoren gezielt variiert und deren Auswirkungen innerhalb des Modells evaluiert werden können, bevor potenzielle Veränderungen in der Praxis umgesetzt werden (Haas, 2020).

Ein wesentlicher Vorteil von Simulationen liegt in der frühzeitigen Identifikation möglicher Entwicklungen, zentraler Dynamiken oder systematischer Fehlerquellen. Dadurch können mögliche Risiken minimiert und effizientere Untersuchungsweisen gewährleistet werden (Haas, 2020). Außerdem ermöglichen Simulationen die Prüfung

von Hypothesen, den Vergleich verschiedener Szenarien und die Herleitung von Vorhersagen, ohne dabei unmittelbar Ressourcen, Zeit oder Sicherheitsaspekte in der realen Welt zu gefährden (Fabricius, 2016).

2.4.1 Grundlagen der Simulation

Simulationen beruhen in der Regel auf mathematischen Gleichungen, die dazu dienen, ein bestimmtes System oder ein spezifisches Phänomen zu beschreiben. Der Prozess umfasst mehrere essenzielle Schritte, wie die Modellbildung, die numerische Implementierung sowie die Diskretisierung (Schweer et al., 2024). Dabei spielt die Entwicklung und Anwendung geeigneter Modelle eine zentrale Rolle, da sie eine essenzielle Voraussetzung für wissenschaftliche Erkenntnisse und die Validierung theoretischer Anwendungen darstellen (Fabricius, 2016). Insbesondere bei realen Problemstellungen, deren praktische Untersuchung aufgrund der Komplexität und der hohen Kosten nicht empfehlenswert ist, bietet die Simulationen eine theoretische Alternative, um unterschiedliche Szenarien abzubilden, zu analysieren und vorhandene Unsicherheiten zu berücksichtigen (Bankhofer, 2022).

Im Mittelpunkt einer Simulation steht die Modellbildung, die darauf abzielt, ein reales System mithilfe einer gezielten Abstraktion und Vereinfachung nachzubilden. Ziel des Prozesses ist es, die Wirklichkeit in einem adäquaten, aber zugleich praktikablen Abbild darzustellen. Dabei wird angestrebt, alle relevanten Eigenschaften möglichst umfassend in die Simulation zu integrieren. Der Detaillierungsgrad eines Modells kann jedoch nicht beliebig erhöht werden, da das sogenannte Bonini-Paradoxon besagt, dass Modelle mit zunehmender Komplexität an Verständlichkeit verlieren. Aus diesem Grund ist eine gezielte Selektion der wesentlichen Eigenschaften erforderlich, um die Modellkomplexität auf ein sinnvolles Maß zu begrenzen. Dies erleichtert die Reduktion überflüssiger Details, wodurch verschiedene Eingangsparameter gezielt experimentell variiert und Berechnungen effizienter durchgeführt werden können, um verschiedene Szenarien zu analysieren (Herzog, 2021).

Für eine grundlegende Verifikation des Simulationsmodells ist es erforderlich, empirische Daten zu erheben, um die relevanten Eingangsparameter zu bestimmen. Sind diese nicht in ausreichendem Umfang verfügbar, können stattdessen Bandbreiten festgelegt werden, innerhalb derer sich potenzielle Werte bewegen. Nach Abschluss der Modellierung erfolgt in der Regel die Implementierung in Form eines Computerprogramms, welches

anschließend einer umfangreichen Testphase unterzogen wird. Dabei werden mehrere Testläufe durchgeführt, um etwaige Modellfehler zu identifizieren und anschließend zu beheben (Altiok & Melamed, 2010).

Ein wesentlicher Vorteil besteht in der Handhabung stochastischer Elemente innerhalb des Simulationsmodells. Insbesondere bei Systemen mit Warteschlangen oder bei Projekten mit unbestimmter Vorgangsdauer ist es erforderlich, zufallsbedingte Variationen realitätsgetreu zu erfassen. Hierzu werden mathematische Verteilungen und Zufallszahlen genutzt, um unterschiedliche Zustände innerhalb der Simulation möglichst realistisch abzubilden (Bankhofer, 2022).

Nach der Durchführung mehrerer Simulationsdurchläufe ist eine fundierte Analyse der Ergebnisse entscheidend. Nur durch eine sorgfältige Auswertung der generierten Daten lassen sich nachvollziehbare und belastbare Schlussfolgerungen ableiten. Dabei ist es besonders wichtig, Optimierungspotenziale zu identifizieren und gegebenenfalls gezielte Anpassungen am Modell vorzunehmen (Altiok & Melamed, 2010). Die grafische Visualisierung der Ergebnisse stellt ein wertvolles Hilfsmittel zur Interpretation der Daten dar. Da sie jedoch die Wahrnehmung der Ergebnisse beeinflussen kann, ist eine objektive und präzise Analyse von zentraler Bedeutung, um verzerrte Einschätzungen und Fehlschlüsse zu vermeiden (Fabricius, 2016).

2.4.2 Arten von Simulationen

Simulationen lassen sich anhand verschiedener Kriterien differenzieren, die sich insbesondere in ihrer Vorgehensweise und ihrem Anwendungsbereich unterscheiden. Grundlegend unterscheidet man zwischen deterministischen und stochastischen Simulationen. Deterministische Simulationen beruhen auf festgelegten mathematischen Regeln, wodurch das Systemverhalten vollständig durch die Eingabedaten bestimmt wird. Dies hat zur Folge, dass bei identischen Ausgangsbedingungen stets dasselbe Ergebnis erzielt wird (Roth, 2018). Solche Simulationen finden beispielsweise in Bereichen Anwendung, in denen exakte Prognosen notwendig sind, wie in der Produktionssteuerung oder der Verkehrsplanung (Herzog, 2021).

Stochastische Simulationen hingegen verwenden Zufallsvariablen und berücksichtigen Unsicherheiten, um reale Prozesse oder Systeme zu modellieren. Sie sind insbesondere in Bereichen relevant, in denen Unsicherheiten oder zufällige Schwankungen eine zentrale Rolle spielen, wie beispielsweise in der Risikobewertung oder in

Warteschlangensystemen. In diesen Modellen werden unter anderem Bedienzeiten und Ankunftszeiten als zufällige Größen interpretiert, um verschiedene Szenarien abzubilden (Herzog, 2021). Eine besonders häufig angewendete Methode innerhalb der stochastischen Simulation ist das Monte-Carlo-Verfahren. Ursprünglich wurde das Verfahren für militärische Anwendungen entwickelt. Heute findet das Monte-Carlo-Verfahren breite Anwendung in Disziplinen wie der Finanzwirtschaft oder den Naturwissenschaften (Schweer et al., 2024). Diese Form nutzt Zufallsgeneratoren und eine Vielzahl an Wiederholungen, um eine Annäherung an die Verteilung möglicher Systemausgänge zu erzielen. Hierbei werden Zufallszahlen durch einen sogenannten „Random Number Generator“ erzeugt, der gleichverteilte Werte im Intervall zwischen null und eins liefert. Diese Zufallszahlen werden anschließend so transformiert, dass sie einer vorgegebenen Verteilung entsprechen. Zur Ableitung statistisch belastbarer Aussagen über das Systemverhalten werden mehrere Simulationsdurchläufe mit unabhängigen Zufallszahlen durchgeführt. Aus den resultierenden Einzelergebnissen wird im Anschluss der Mittelwert gebildet (Altiok & Melamed, 2010).

Eine weitere wesentliche Kategorie stellt die agentenbasierte Simulation dar. Bei diesem Ansatz liegt der Fokus auf einzelnen individuellen Akteuren und deren Interaktionen. Die sogenannten Agenten agieren autonom innerhalb des simulierten Systems und passen ihr Verhalten dynamisch an Veränderungen in ihrer Umgebung an (Schweer et al., 2024). Diese Methode wird insbesondere in Wirtschaftssimulationen eingesetzt, um Marktmechanismen und Entscheidungsprozesse zu analysieren. Darüber hinaus findet sie im Bereich des Umweltschutzes Anwendung, um beispielsweise Untersuchungen des Verhaltens von Individuen oder Unternehmen in Bezug auf Ressourcennutzung und Umweltpolitik durchzuführen (Bandyopadhyay & Bhattacharya, 2014).

Eine weitere Unterscheidung erfolgt zwischen diskreten und kontinuierlichen Simulationsmodellen. Diskrete Ereignissimulationen modellieren Systeme, deren Zustand sich nur zu bestimmten, exakt definierten Zeitpunkten ändert. Dies ist beispielsweise der Fall, wenn ein neuer Kunde eine Warteschlange betritt oder verlässt (Haas, 2020). Die Zustandsänderungen innerhalb des Systems erfolgen hier nicht kontinuierlich, sondern in zeitlich getrennten Sprüngen zwischen den aufeinanderfolgenden Ereignissen. Aufgrund der Fähigkeit, logistische Abläufe detailliert darzustellen, werden diskrete Simulationen insbesondere für Produktionsprozesse oder Logistiksysteme eingesetzt (Herzog, 2021). Kontinuierliche Simulationen hingegen bilden Zustandsänderungen mithilfe von Differentialgleichungen ab. Solche Modelle

eignen sich insbesondere für dynamische Systeme, in denen sich Variablen kontinuierlich verändern, wie zum Beispiel Geschwindigkeit oder Temperatur (Bandyopadhyay & Bhattacharya, 2014). Aufgrund dessen finden sie oft in Bereichen wie der Flugdynamik oder der Thermodynamik Anwendung, in denen physikalische Prozesse mithilfe mathematischer Modelle laufend weiterberechnet werden (Haas, 2020). Darüber hinaus existieren hybride Simulationsansätze, die sowohl diskrete als auch kontinuierliche Komponenten in einem gemeinsamen Modell integrieren. Solche Modelle werden beispielsweise in Produktionssystemen eingesetzt, in denen eine kontinuierliche Erfassung von Lagerbeständen mit diskreten Ereignissen, wie Wareneingängen oder Transportvorgängen, kombiniert wird (Bandyopadhyay & Bhattacharya, 2014).

Zusätzlich lassen sich Simulationsmodelle in lokale und verteilte Simulationen untergliedern. Lokale Simulationen werden auf einem einzelnen System ausgeführt, wobei sämtliche Prozesse und Berechnungen zentral auf einem einzigen Rechner stattfinden. Verteilte Simulationen hingegen nutzen mehrere miteinander verbundene Systeme, um einzelne Teilmodelle parallel zu verarbeiten. Durch diese Aufteilung können mehrere Berechnungen parallel auf verschiedenen Rechnern durchgeführt werden, wodurch sich die Rechenlast verteilt und die Berechnungsgeschwindigkeit erheblich gesteigert wird (Herzog, 2021).

3 Methodik

Zur Analyse der stochastischen Einflussfaktoren auf die Leistung logistischer Netzwerke wurde ein diskretes simulationsbasiertes Vorgehen gewählt, das auf einem bestehenden deterministischen Modell aufbaut und dieses gezielt mit zufallsbasierten Elementen erweitert. Ziel ist die Entwicklung einer Simulationsumgebung, die eine realitätsnähere Abbildung des Systems unter Berücksichtigung von Unsicherheiten ermöglicht. Hierzu wurden unter anderem schwankende Transport- und Bearbeitungszeiten integriert, um praxisnahe Bedingungen innerhalb des Netzwerkes zu schaffen.

Die Umsetzung des Modells erfolgt vollständig in der Programmiersprache Python, die eine Vielzahl an unterstützenden Bibliotheken zur Verfügung stellt. Sowohl die Datenverarbeitung, die Durchführung als auch die Auswertung der Simulation kann hierüber optimal abgebildet werden.

Die methodische Vorgehensweise gliedert sich in mehrere, aufeinander aufbauende Schritte. Zunächst erfolgt die Problemformulierung als Grundlage der Modellkonzeption, gefolgt von der Aufbereitung und Bereitstellung der Eingangsdaten. Darauf aufbauend wird die geeignete Python-Bibliothek zur Simulationsdurchführung ausgewählt und deren zentrale Bestandteile erläutert. Im Anschluss wird die generelle Struktur des erstellten Modells beschrieben, die in Form von mehreren Modulen umgesetzt wurde. Dieses Konzept sorgt für eine klare Trennung der Funktionsbereiche im Simulationsmodell.

3.1 Problembeschreibung

In Anlehnung an den Artikel „Service-Network Design for Same-Day Delivery with Hub Capacity Constraints“ von Haotian Wu et al. (2023) sowie auf Grundlage der Erkenntnisse aus der Zusammenarbeit mehrerer Professoren, welche im Artikel „The Scheduled Service Network Design Problem with Terminal Resource Acquisition“ von Belieres et al. (2025) festgehalten sind, wurde ein Problem identifiziert, das durch die Entwicklung eines geeigneten Modells gelöst werden soll.

Aufbauend auf dem Modell von Belieres et al. werden verschiedene Service Netzwerke (siehe Kapitel 2.3) analysiert, die sich in den Vereinigten Staaten befinden und unterschiedlich große Instanzen in den Bundestaaten Georgia und Alabama abbilden. Diese sind als logistisches Netzwerk gestaltet und umfassen je nach Instanz zwischen elf

und zwanzig Standorte, die im System als ansteuerbare Knotenpunkte fungieren. Innerhalb dieses Netzwerks erfolgt der Weitertransport über Lieferwagen, die an den jeweiligen Knotenpunkten Waren aufladen und abladen. Jeder der Punkte verfügt über mehrere Ladetore, an denen Trucks be- oder entladen werden können. Ein wesentliches Entscheidungskriterium besteht in der optimalen Nutzung dieser Tore. Es muss bestimmt werden, wie viele Tore für einen effizienten Logistikprozess geöffnet werden sollen, während parallel die entstehenden Betriebskosten berücksichtigt werden, die durch das Öffnen eines Tores entstehen.

Das derzeitige Modell, welches von Belieres et al. entwickelt wurde, basiert auf einer deterministischen Simulation, die in Kapitel 2.3.1 detailliert erläutert wird. In dieser sind die Menge der zu transportierenden Waren, deren frühestmögliche Verfügbarkeit sowie die spätesten zulässigen Lieferzeitpunkte vorgegeben. Die Waren müssen innerhalb des Netzwerkes von vordefinierten Startknoten zu festgelegten Endknoten geliefert werden. Dabei stehen unterschiedliche Transportwege zur Verfügung, die ebenfalls vorweg bestimmt wurden. Eine weitere Besonderheit besteht in der Möglichkeit zur Konsolidierung von Warenströmen, ein Begriff, der bereits in Kapitel 2.3 definiert wurde. Fahrzeuge können gezielt Zwischenknotenpunkte ansteuern, um dort entweder einen Teil ihrer Fracht oder die gesamte Ladung auf andere Lieferwagen zu übergeben. Diese Umladevorgänge ermöglichen eine effizientere Bündelung von Waren und tragen zur Optimierung des Weitertransports bei. Voraussetzung hierfür ist, dass auf dem empfangenden Fahrzeug ausreichend freie Kapazitäten vorhanden sind und die maximal zulässige Frachtmenge nicht überschritten wird. Diese Zusammenführung von Teilladungen ist bereits im bestehenden deterministischen Modell integriert und muss in der weiteren Modellierung entsprechend berücksichtigt werden.

Ein wesentlicher Nachteil des deterministischen Modells besteht darin, dass Fahrtzeitverzögerungen oder schwankende Wartezeiten an den Ladetoren der jeweiligen Knotenpunkte nicht berücksichtigt werden. Diese Verzögerungen können allerdings durch Belade- und Entladeprozesse auftreten und somit die gesamte Transportkette beeinflussen. Daher erscheint es sinnvoll, die bereits bestehenden Daten in ein stochastisches Modell zu integrieren, dessen Grundlagen in Kapitel 2.3.2 erläutert werden. Auf diese Weise können potenzielle Schwankungen innerhalb der Simulation berücksichtigt werden. Hierdurch besteht die Möglichkeit zu analysieren, ob die Waren trotz variierender Rahmenbedingungen weiterhin fristgerecht bis zum jeweils spätesten Lieferzeitpunkt zugestellt werden können. Darüber hinaus trägt dieser Ansatz zu einer

realitätsnäheren Abbildung der Transportprozesse bei und ermöglicht eine belastbare Bewertung des Systems unter variierenden Bedingungen.

3.2 Datenbeschaffung und -aufbereitung

Das bestehende deterministische Simulationsmodell, welches in Zusammenhang mit dem Artikel von Belieres et al. entwickelt wurde, stellt bereits eine umfangreiche Datenbasis sowie Simulationsergebnisse zur Verfügung. Diese Daten dienen als Grundlage für die Entwicklung der stochastischen Variante des Modells.

Ein zentrales Dokument enthält die verwendeten Eingangsdaten für das deterministische Modell. Es umfasst sämtliche Informationen, die für die Durchführung der Simulation erforderlich sind. Für die Erstellung der stochastischen Simulation sind bestimmte Daten von bedeutender Relevanz. Den Knotenpunkten des Netzwerkes wird jeweils ein numerischer Index zugewiesen. So kann eine eindeutige Identifikation der einzelnen Standorte innerhalb der Simulation sichergestellt werden. Darüber hinaus enthält das Dokument eine Übersicht über die möglichen Verbindungen zwischen den einzelnen Knotenpunkten, inklusive ihrer jeweiligen Transportkosten und Distanzen. Ein weiterer Bestandteil ist die detaillierte Darstellung der zu transportierenden Waren. Neben dem Start- und Zielknotenpunkt sind außerdem die Transportmenge, die frühestmöglichen Verfügbarkeiten und die spätesten Lieferzeitpunkte einzusehen. Des Weiteren wird angegeben, welche möglichen Transportwege der einzelnen Waren zur Verfügung stehen. Besonders hervorzuheben ist die Zeitangabe im vorliegenden Dokument, welche auf der Struktur eines Time-Expanded Networks (siehe Kapitel 2.3) basiert. Der gesamte Ablauf wurde dabei in einzelne, diskrete Perioden unterteilt, wobei eine Periode einer Zeiteinheit von 30 Minuten entspricht. Es bestehen jedoch zwei Ausnahmen in der Darstellung. Sowohl die Bearbeitungszeit an den Knotenpunkten als auch die maximale Wartezeit sind direkt in Minuten angegeben. Die Ladezeiten variieren je nach Modell und betragen in Abhängigkeit der jeweiligen Ausprägung 30, 60 oder 120 Minuten. Die maximale Wartezeit hingegen ist einheitlich angegeben und beträgt 60 Minuten.

Neben den Eingangsdaten existiert ein weiteres Dokument, das die Simulationsergebnisse des deterministischen Modells enthält. In diesem Dokument sind detaillierte Analysen verschiedener Parameter abgebildet, von denen einige auch für die stochastische Modellvariante von Bedeutung sind. Eine relevante Information der Ergebnisse ist die Anzahl der an den einzelnen Knotenpunkten benötigten Ladetore für das Beladen und

Entladen. Zudem liefert das Dokument umfassende Informationen über die benötigten Lieferwagen. Aus einer Übersicht ist die Anzahl der eingesetzten Fahrzeuge, einschließlich der zugewiesenen Routen sowie der Abfahrts- und Ankunftszeiten an den jeweiligen Zielorten zu entnehmen. Außerdem liefern die Daten Informationen über die spezifische Ladungsverteilung innerhalb der Lieferwagen. Jede Ware wurde mit einem eindeutigen Index versehen, der zur Fahrzeugzuordnung dient. Durch diese Kennzeichnung ist es nachvollziehbar, welche Waren sich auf den jeweiligen Transportfahrzeugen befinden.

Die durch das bestehende deterministische Modell verfügbaren Daten bilden die Grundlage für die Entwicklung und Implementierung der stochastischen Simulation. Zur Verarbeitung dieser Daten werden die einzelnen Dokumente mithilfe der Programmiersprache Python eingelesen, um die relevanten Informationen systematisch zu extrahieren und zu speichern. Dabei kommt die externe Bibliothek „pandas“ zum Einsatz, die zur effizienten Datenverarbeitung dient. Die gewonnenen Daten werden in strukturellen Tabellen zusammengefasst, wodurch eine übersichtliche Datenbasis zur simpleren Weiterbearbeitung entsteht. Anschließend werden diese Daten für die Simulation aufbereitet und in Form von CSV-Dateien abgespeichert. Ab diesem Punkt dienen die erstellten Dokumente als Grundlage für die stochastische Simulation und werden den jeweiligen Klassen zur weiteren Nutzung bereitgestellt.

Da die stochastische Simulation im Gegensatz zur deterministischen Variante darauf ausgelegt ist, Schwankungen und Unsicherheiten zu berücksichtigen, müssen für verschiedene Variablen realitätsnahe Annahmen getroffen werden. Insbesondere beziehen sich diese Annahmen auf die Variabilität in der Lieferzeit sowie in der Be- und Entladezeit. Um möglichst praxisnahe Bedingungen abzubilden, werden realistische Wahrscheinlichkeitsverteilungen verwendet, die flexibel über das Terminal definiert und anschließend in der Simulation integriert werden. Diese Umsetzung bietet die Möglichkeit, verschiedene Modellvarianten unter Berücksichtigung unterschiedlicher stochastischer Variablen durchzuführen und zu simulieren.

3.3 Einsatz von SimPy für die Simulationsdurchführung

Da die Datenaufbereitung bereits in Python durchgeführt wurde, liegt es nahe, auch die Simulationsdurchführung in derselben Umgebung umzusetzen. Dieses Vorgehen gewährleistet eine einheitliche Programmiersprache für die Umsetzung der Prozessschritte und ermöglicht eine reibungslose Integration der aufbereiteten Daten in die Simulationsumgebung. Für die Umsetzung wird die externe Bibliothek „SimPy“ integriert und für die Simulation genutzt.

SimPy ist speziell für diskrete Ereignissimulationen in Python entwickelt worden und ermöglicht die Modellierung von Abläufen und die Interaktion mit ihnen innerhalb der Simulationsumgebung. Die grundlegende Struktur basiert auf Prozessen, Ereignissen und der Umgebung, welche die Simulation steuert. Die Prozesse werden als Python-Generatorfunktion integriert und bilden aktive Komponenten ab, wie beispielsweise Kunden, Fahrzeuge oder Agenten. Diese können durch das Eintreten von Ereignissen unterbrochen oder aktiviert werden. Die Umgebung dient dabei als zentraler Zeitmanager, der die Reihenfolge der Ereignisse organisiert und kontrolliert (Team SimPy, 2024). Die flexible Definition und Steuerung der Prozesse gewährleisten aufgrund dieser Struktur eine möglichst realitätsnahe Abbildung komplexer Systeme.

3.3.1 Grundlagen von SimPy

Die nachfolgenden Bestandteile bilden die Grundlage für die ausgeführten Simulationen, um eine möglichst realitätsnahe Modellierung der vorliegenden Prozesse und Ereignisse abzubilden. Für die bessere Verständlichkeit der verwendeten Bibliothek SimPy werden alle eingesetzten Elemente im Folgenden einzeln aufgeführt und detailliert erklärt.

Das zentrale Element von SimPy ist die Simulationsumgebung, welche durch die Klasse simpy.Environment() definiert wird. Sie dient als Verwaltungseinheit und steuert die Simulationszeit sowie die Reihenfolge der Ereignisse. Aufgrund dieser Steuerung und durch den Einsatz der Methode env.run() wird die Simulation entweder so lange ausgeführt, bis keine Ergebnisse mehr anstehen oder bis zu einem vorgegebenen Zeitpunkt, an dem die Simulation automatisch endet. Dieser Zeitpunkt kann als Parameter innerhalb der Klammern der Methode angegeben werden. Ein weiteres Element ist die Methode env.process(), die eine kompakte Möglichkeit zur Erstellung von Simulationsprozessen darstellt. Ein Prozess wird in SimPy als Python-Generatorfunktion

definiert und wird durch das Aufrufen von env.process() in das Simulationsumfeld eingebunden. Die Methode verwendet dafür die Generatoren zur schrittweisen Ausführung von Ereignissen und ermöglicht dadurch realitätsnahe Szenarien, in denen Prozesse parallel oder asynchron ablaufen können. Eine weitere wichtige Komponente ist die Klasse simpy.Resource(env, capacity). Diese bildet gemeinsam genutzte Ressourcen mit einer festgelegten Kapazität ab. Der Parameter capacity, der innerhalb der Klammern definiert wird, bestimmt die maximale Anzahl von Prozessen, die gleichzeitig Zugriff auf die Ressourcen erhalten können. Besonders relevant ist diese Klasse für die Modellierung von Warteschlangensystemen, wie beispielsweise die Verwaltung von Kassen oder die Belegung von Maschinen. SimPy stellt verschiedene Typen solcher Ressourcen-Klassen zur Verfügung, die je nach spezifischem Anwendungsfall eingesetzt werden können. Eine dieser Erweiterungen ist die Klasse simpy.PriorityResource(env, capacity). Während die klassische Ressource Anfragen nach dem First-Come-First-Serve-Prinzip behandelt, erfolgt die Bearbeitung bei simpy.PriorityResource auf Grundlage von prioritätsbasierten Anfragen. Prozesse mit einem niedrigeren Prioritätswert erhalten bevorzugten Zugriff auf die Ressource, wodurch dringendere oder kritische Prozesse schneller bearbeitet werden können. Diese Klasse ist insbesondere in Szenarien mit unterschiedlich priorisierten Aufgaben von Bedeutung, wie beispielsweise bei der Simulation von Notaufnahmen. Zur Verwaltung dieser Ressourcen werden die Methoden resource.request() und resource.release() verwendet. Ein Prozess fordert mithilfe von resource.request() eine Ressource an, woraufhin er so lange in der Warteschlange verweilt, bis diese verfügbar ist. Um die Ressource nach der Nutzung wieder explizit freizugeben, wird resource.release() eingesetzt. Durch diese Methode wird die Ressource wieder für andere Prozesse zugänglich gemacht. Für die Simulation wurde zudem die SimPy-Variable env.now verwendet, mit der jederzeit die aktuelle Simulationszeit abgefragt werden kann. Diese Variable gibt den aktuellen Zeitpunkt der Simulation als numerischen Wert und ohne Einheit zurück. Die Zeitmessung beginnt standardmäßig bei null, kann jedoch während der Initiierung der Simulation individuell angepasst werden. Das letzte aus der Bibliothek genutzte Element ist die Anweisung yield env.timeout(t), die dazu dient, einen Prozess für t Zeiteinheiten zu unterbrechen. Sie wird verwendet, um realistische Verzögerungen oder Wartezeiten abzubilden, wie beispielsweise in einem Warteschlangensystem. Dadurch können Abläufe mit variierenden Prozessdauern realitätsnah simuliert werden.

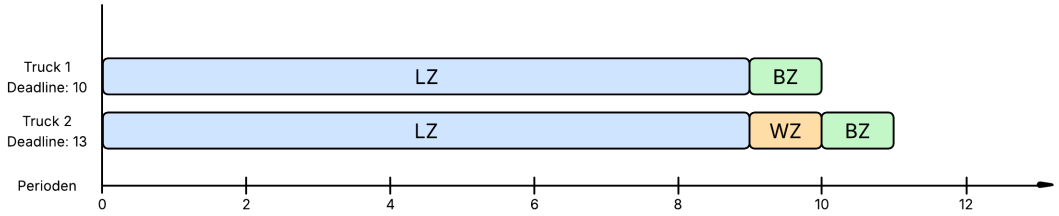
3.3.2 Struktur der Simulation

Um für eine klare und nachvollziehbare Struktur zu sorgen, wurde der Code in drei einzelne Module unterteilt, nämlich FileHandler, Simulation und Main. Durch diese Aufteilung wird eine systematische Trennung der Funktionen sichergestellt.

Das FileHandler-Modul ist für die Verarbeitung und Verwaltung der Daten verantwortlich. Darin werden die bestehenden Instanz- und Eingabedaten aus der deterministischen Simulation eingelesen und anschließend für die stochastische Simulation in aufbereiteter Form bereitgestellt. Relevante Daten, wie unter anderem Kapazitätswerte oder Zeitparameter, werden extrahiert, verarbeitet und in strukturierten Tabellen gespeichert. Zur einfachen Weiterverarbeitung werden diese Daten in Form von CSV-Dateien abgelegt und in logischer Struktur organisiert, sodass die Integration in das Simulationsmodul problemlos möglich ist.

Das Simulation-Modul stellt das zentrale Element der Simulationsumgebung dar. Innerhalb dieses Moduls befindet sich die Hauptklasse „TruckSimulation“, die für die Durchführung der Simulation verantwortlich ist. Das Ziel der Klasse besteht darin, die relevanten Abläufe der Simulation zu steuern und zu verwalten. Die Simulation wird in zwei separaten Durchläufen durchgeführt. Der erste Durchlauf bildet das bestehende deterministische Modell unter Berücksichtigung der stochastischen Einflussfaktoren ab. Im zweiten Durchlauf werden zusätzlich spezifische Entscheidungsregeln aktiviert, mit dem Ziel, die Simulationsergebnisse gezielt zu verbessern. Je nach Konfiguration der Simulation können bis zu drei unterschiedliche Optimierungsmechanismen aktiviert werden. Die erste Regel führt zu einer priorisierten Ressourcenzuweisung an den Knotenpunkten, bei der Lieferfahrzeuge mit einer früheren Deadline bevorzugt behandelt werden. Treffen mehrere Fahrzeuge gleichzeitig an einem Knotenpunkt ein, erfolgt die Entscheidung der Bearbeitungsreihenfolge anhand der verbleibenden Zeit bis zur Deadline. Fahrzeuge mit einer geringeren verbleibenden Frist werden priorisiert bearbeitet, während andere Fahrzeuge bis zur Bearbeitung in der Warteschlange verbleiben. Dieses Prinzip ist in Abbildung 1 veranschaulicht. Lieferwagen 1 weist eine Deadline von zehn Perioden auf und wird daher bevorzugt bearbeitet, da die Deadline von Lieferwagen 2 bei 13 Perioden liegt. Dieser wartet zunächst in einer Warteschlange und wird im Anschluss bearbeitet.

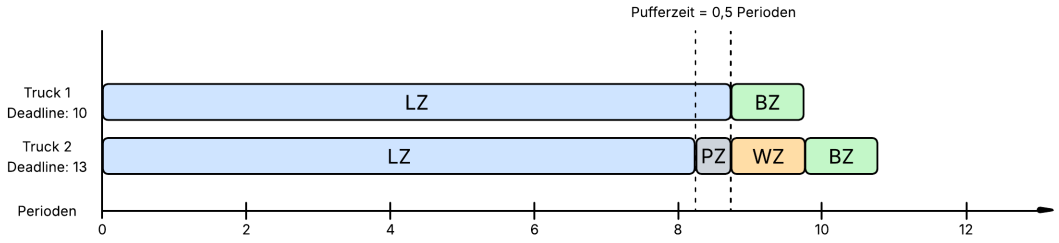
Abbildung 1: Priorisierung von Lieferfahrzeugen



Quelle: Eigene Darstellung

Da die erste Entscheidungsregel aufgrund stochastischer Schwankungen nicht in allen Fällen zur Anwendung kommt, weil Lieferfahrzeuge durch die Verzögerungen zeitversetzt an den Knotenpunkten eintreffen, wurde eine zweite Regel implementiert, die gezielt auf diese Problematik reagiert. Dabei handelt es sich um die Integration einer Pufferzeit, welche nach dem Eintreffen der Lieferfahrzeuge am jeweiligen Knotenpunkt aktiviert wird. Ziel dieser Maßnahme ist es, leicht verspätet eintreffende Lieferfahrzeuge zu berücksichtigen und gegebenenfalls zu bevorzugen, sofern sie eine engere Deadline aufweisen. Diese Entscheidungsregel ist in Abbildung 2 veranschaulicht. Fahrzeug 2 erreicht den Knotenpunkt zuerst, verzichtet jedoch auf die sofortige Bearbeitung und wartet 0,5 Perioden. In diesem Zeitraum trifft Fahrzeug 1 mit einer knapperen Deadline ein. Durch die Priorisierungsregel wird Lieferfahrzeug 1 nun bevorzugt behandelt und Fahrzeug 2 wartet in der Warteschlange auf die anschließende Bearbeitung.

Abbildung 2: Einsatz von Pufferzeiten zur Priorisierung von Lieferfahrzeugen



Quelle: Eigene Darstellung

Die dritte Entscheidungsregel ermöglicht das Öffnen zusätzlicher Ladetüren an den Knotenpunkten. Sobald sich an einem Knotenpunkt eine bestimmte Anzahl von Lieferfahrzeugen angestaut hat, wird automatisch ein weiteres Ladetor geöffnet, um die Bearbeitungskapazität zu erhöhen und die Wartezeiten zu reduzieren.

Die anschließende Auswertung der Ergebnisse erfolgt ebenfalls innerhalb des Moduls. Durch eine speziell implementierte Funktion werden die gewonnenen Ergebnisse extrahiert und in Form einer Tabelle ausgegeben, um eine systematische Analyse der Resultate zu gewährleisten.

Den Einstiegspunkt und damit die zentrale Steuereinheit der Simulation stellt das Main-Modul dar. Es koordiniert den Ablauf, indem zunächst das FileHandler-Modul aufgerufen wird, um die erforderlichen Eingabedaten zu laden. Im Anschluss wird das Simulation-Modul ausgeführt, in dem die Simulation initiiert wird. Die aufbereiteten Daten werden in diesem Schritt an die Klasse TruckSimulation übergeben. Des Weiteren ermöglicht das Main-Modul die Eingabe der stochastischen Parameter, die für die Simulation erforderlich sind. Diese können durch den Anwender individuell über das Terminal definiert werden.

3.3.3 Konzeption der Simulation

Das Grundprinzip der Simulation basiert auf der Modellierung eines Transportnetzwerkes für Lieferwagen unter Berücksichtigung stochastischer Transport- und Bearbeitungszeiten. Ziel ist es, durch variierende Parameter unterschiedliche Szenarien abzubilden, um ein realitätsnahes Modell zu erstellen.

Vor Beginn der Simulation wird der Anwender über das Terminal dazu aufgefordert, prozentuale Abweichungen für verschiedene Parameter festzulegen. Dabei kann angegeben werden, um wie viel Prozent die Fahrtzeiten sowie die Belade- und Entladezeiten sowohl nach oben als auch nach unten abweichen dürfen. Diese Eingaben werden anschließend in die Simulationsklasse übernommen, die zusätzlich auf die zuvor erstellten CSV-Tabellen zugreift, welche unter anderem die theoretischen Ladezeiten und die Kapazitäten der Ladetore enthalten.

Nach dem Einlesen und Verarbeiten der Daten erfolgt die Definition der Simulationsumgebung, in der alle benötigten Ressourcen erstellt werden. Dazu gehören vorwiegend die Ressourcen für die Knotenpunkte, an denen das Beladen und Entladen erfolgt. Darüber hinaus werden weitere Ressourcen definiert, um die Anzahl der vor den Ladetoren wartenden Fahrzeuge zu erfassen und die maximalen Warteschlangenlängen zu messen. So können im späteren Verlauf Engpässe und Verzögerungen im Transportprozess analysiert werden.

Nachdem die Simulation initialisiert wurde, kann deren Ausführung erfolgen, wobei alle zuvor eingelesenen Lieferwagen sequenziell abgearbeitet werden. Das Verhalten der Transporter ist dabei durch einen vordefinierten Prozessablauf vorgeschrieben. Für jede einzelne Fahrt werden vorab die individuellen Abweichungen der Transport- und Bearbeitungszeiten berechnet. Hierfür wird eine trianguläre Wahrscheinlichkeitsverteilung genutzt, bei der die untere festgelegte prozentuale Abweichung den Minimalwert, die obere prozentuale Abweichung den Maximalwert und die theoretische Zeit den Mittelwert darstellt. Durch diese Methode kann für jeden Lieferwagen eine möglichst realistische Variabilität der Fahrtzeit und der Be- und Entladezeit berechnet werden.

Sobald die stochastischen Parameter bestimmt wurden, kann der eigentliche Prozessablauf für die einzelnen Lieferwagen starten. Jedes Fahrzeug wartet zunächst bis die individuelle Startzeit in der Simulation erreicht ist, bevor es zur ersten Station fährt, an der das Beladen stattfindet. Um den Ladevorgang durchzuführen, fordert das Lieferfahrzeug die spezifische Ressource an, der eine individuelle Kapazität zugewiesen wurde. Falls die Ressource momentan nicht verfügbar ist, gelangt das Fahrzeug in eine Warteschlange. Während sich der Lieferwagen in dieser Warteschlange befindet, wird parallel die Zeit gemessen. Sobald er den Zugang zur Ladestation erhält, stoppt die Zeiterfassung und der eigentliche Ladevorgang beginnt. Die Dauer des Beladens entspricht nun der vorher berechneten Ladezeit inklusive stochastischer Abweichung.

Nach Abschluss des Beladevorgangs setzt das Fahrzeug die Fahrt zum Zielpunkt fort. Die Transportzeit läuft entsprechend der vorab berechneten stochastischen Fahrtzeit in der Simulation ab. Nach Ablauf der Fahrtzeit kommt der Transporter beim Zielknotenpunkt an und der Prozess des Entladens beginnt. Dieser läuft analog zum Beladeprozess ab. Die spezifische Ressource wird durch das Fahrzeug angefordert, das Fahrzeug wartet gegebenenfalls in der Warteschlange und durchläuft schlussendlich den Entladeprozess unter Berücksichtigung der stochastisch angepassten Bearbeitungszeit.

Nach Abschluss des Entladevorgangs wird die Simulationsdauer des Lieferwagens gestoppt und die Zeit registriert, zu der der Prozess beendet wurde. Dieser Ablauf wird für alle eingelesenen Transporter iterativ wiederholt, bis alle von ihnen den Transportprozess durchlaufen haben. Abschließend werden alle individuellen und gespeicherten Daten der Lieferwagen an eine Auswertungsfunktion übergeben. Diese Funktion analysiert die Simulationsergebnisse und bereitet sie strukturiert und

übersichtlich auf, um detaillierte Untersuchungen zu Transportzeiten und Warteschlangenlängen vornehmen zu können.

3.3.4 Ausgabe der Ergebnisse

Nach Durchführung beider Simulationsdurchläufe erfolgt die Ausgabe und Aufbereitung der jeweiligen Ergebnisse. Ziel ist es, die Auswirkungen der integrierten stochastischen Unsicherheiten sowie den Effekt der angewendeten Optimierungsmaßnahmen erkennbar darzustellen.

Im Anschluss an jede Simulationsvariante werden die Ergebnisse automatisiert in Tabellenform aufbereitet und im CSV-Format gespeichert. Diese Tabellen beinhalten verschiedene Leistungskennzahlen zur Beurteilung der jeweiligen Simulationsdurchführung, darunter unter anderem die maximale Warteschlangenlänge oder die Anzahl der wartenden Lieferwagen an den einzelnen Knotenpunkten. Auf Grundlage dieser Werte sollen potenzielle Engpässe identifiziert und daraus Optimierungsmöglichkeiten für die logistische Struktur abgeleitet werden.

Ein besonderer Fokus der Analyse liegt auf der termingerechten Zustellung der Waren. Jeder Ware wurde eine individuelle Deadline zugewiesen, deren planmäßige Einhaltung überprüft wird. In der Auswertung wird für jede Warenlieferung erfasst, ob sie pünktlich oder verspätet eintrifft, inklusive der Dauer einer möglichen Verspätung. Diese Kennzahl dient in der vorliegenden Arbeit als zentrales Bewertungskriterium der Modelle und wird im anschließenden Vergleich besonders hervorgehoben.

4 Implementierung

In diesem Kapitel wird die technische Umsetzung des entwickelten Simulationsmodells zur realitätsnahen Abbildung der logistischen Transportprozesse innerhalb des gegebenen Service-Netzwerks erläutert. Die Implementierung erfolgt in Form eines Python-Programms, welches in drei verschiedene Module aufgeteilt ist. Das erste Modul übernimmt die automatisierte Datenaufbereitung, das zweite Modul bildet das Kernstück der eigentlichen Simulation und das dritte Modul dient der übergeordneten Ausführung und Verknüpfung aller Bestandteile.

Die Simulation wird zur Analyse unterschiedlicher Strategien in zwei Varianten ausgeführt. Die erste Variante stellt ein Standardmodell dar, das ausschließlich stochastische Einflussfaktoren berücksichtigt. Die zweite und optimierte Variante implementiert zusätzlich erweiterte Entscheidungsregeln, mit dem Ziel, Ressourcenauslastungen effizienter zu gestalten und die Anzahl verspäteter Lieferungen zu reduzieren.

Die Auswertung der Simulationsergebnisse erfolgt schlussendlich auf Basis verschiedener Leistungskriterien. Darunter zählt die maximale Wartezeit der Lieferwagen, die maximale Warteschlangenlänge an den Knotenpunkten sowie die Einhaltung geplanter Lieferzeitpunkte. Im Mittelpunkt der Analyse steht jedoch die termingerechte Zustellung der transportierten Waren, da diese an vorgegebene Deadlines gebunden sind.

Die nachfolgenden Abschnitte beschreiben im Detail die technische Umsetzung, die Struktur sowie den schrittweisen Ablauf der einzelnen Module.

4.1 Modul zur Datenverarbeitung

Das entwickelte Python-Modul namens „FileHandler“ (siehe Anhang 1) dient der automatisierten Verarbeitung der bereits bestehenden Dokumente, die als Datengrundlage für die Simulation dienen. Ziel des Moduls ist es, die Rohdaten effizient in eine strukturierte und bereinigte Form umzuwandeln und zu speichern, sodass sie für die weiterführenden Schritte der Simulation unmittelbar zur Verfügung stehen. Dabei gewährleistet diese Vorgehensweise eine reproduzierbare und fehlerfreie Datenverarbeitung und ermöglicht somit die effiziente Bearbeitung variierender Datensätze.

Zu Beginn des Moduls werden die externen Python-Bibliotheken os, glob und pandas importiert. Durch die Bibliotheken os und glob wird Zugriff auf das Dateisystem sowie das gezielte Auffinden von Dateien mit bestimmten Endungen oder Bestandteilen ermöglicht. Die Bibliothek pandas hingegen dient der effizienten Verarbeitung und Manipulation von Daten in tabellarischer Form.

Die Gesamtstruktur des Moduls ist in neun separate Funktionen unterteilt, die jeweils einen spezifischen Teil des Prozesses zur Datenverarbeitung übernehmen. Diese Trennung dient der Modularität und der Wiederverwendbarkeit des Codes. Alle Funktionen folgen einem definierten Ablauf und bauen logisch aufeinander auf.

Im ersten Codeabschnitt erfolgen die automatische Identifikation und das Einlesen von relevanten Dateien mit einem bestimmten Muster im Namen. Durch die Funktion read_in_file() wird ein Dateipfad generiert, der auf Textdateien mit einem definierten Schlüsselwort innerhalb eines vorgegebenen Ordners abzielt. Es wird die Bibliothek glob genutzt, um die Suche nach den Dateien zu ermöglichen, die den bestimmten Mustern entsprechen. Diese Funktion wird innerhalb des Abschnittes zweimal aufgerufen, um die gewünschten Dateien mit dem richtigen Schlüsselwort zu extrahieren.

Die Funktion save_output_file() ist für die standardisierte Generierung von Ausgabepfaden für die im weiteren Verlauf erzeugten Dateien zuständig. Im zuvor definierten Ausgabeordner wird ein vollständiger Pfad unter Verwendung eines gegebenen Schlüsselwortes sowie der Dateiendung, die standardmäßig als CSV-Format definiert ist, erstellt. In Kombination mit read_in_file() ermöglicht diese Vorgehensweise eine einheitliche Verwaltung der Dateien. In den nachfolgenden Schritten kann direkt über den Aufruf der Funktionen auf die jeweiligen Eingabe- oder Ausgabedateien zugegriffen werden, ohne die Pfade in jeder Funktion neu definieren oder einlesen zu müssen.

Innerhalb der Funktion read_arcs() werden die Informationen zu den Transportrouten aus der Instanzdatei extrahiert und in tabellarischer Form aufbereitet. Zunächst wird die Datei zeilenweise eingelesen und der Inhalt in einer Liste zwischengespeichert. Anschließend wird der relevante Abschnitt des Dokuments identifiziert, der durch das Schlüsselwort „ARCS“ eingeleitet und durch das Auftreten von „COMMODITIES“ begrenzt wird. Innerhalb dieses Bereichs werden alle relevanten Zeilen extrahiert und bereinigt. Die gewonnenen Daten werden anschließend in einem sogenannten DataFrame abgelegt, einer durch die externe Bibliothek pandas bereitgestellten Datenstruktur in tabellarischer Form. Die Spaltenbezeichnungen der Tabelle werden entsprechend der Quelldatei

übernommen. Zu den extrahierten Werten zählen unter anderem die IDs der Start- und Zielknoten, die variablen und fixen Kosten oder die Anzahl der Perioden je Transportroute. Abschließend wird der DataFrame mithilfe der Funktion save_output_file() im vordefinierten Ausgabeordner gespeichert.

Im anschließenden Schritt werden mithilfe der Funktion read_trucks() die Fahrzeuginformationen aus der Ergebnisdatei der Netzplangestaltung extrahiert. Auch hier wird die Datei vollständig eingelesen und zeilenweise zwischengespeichert. Der relevante Datenbereich wird durch die Schlüsselwörter „EXPANDED_VEHICLE“ und „EXPANDED_FLOWS“ abgegrenzt, sodass ausschließlich die erforderlichen Zeilen weiterverarbeitet werden. Die gewonnenen Daten werden ebenfalls in einem pandas-DataFrame abgelegt. Da die Fahrzeuge keine Nummerierung in der Originaldatei enthalten, wird jedem Lieferwagen eine fortlaufende ID zugewiesen, die in einer zusätzlichen Spalte „Truck“ ausgegeben wird. Die übrigen Spalten enthalten beispielsweise Informationen über die zugeordnete Route oder die Start- und Endzeitpunkte des jeweiligen Fahrzeugs. Abschließend wird auch dieser DataFrame im CSV-Format im vorgegebenen Ausgabeordner gespeichert.

Zur Erfassung der Kapazitäten an den einzelnen Knotenpunkten wird die Funktion read_capacities() eingesetzt. Dabei erfolgt erneut ein Zugriff auf die Ergebnisdatei, aus der die relevanten Abschnitte für die Eingangskapazitäten sowie für die Ausgangskapazitäten identifiziert werden. Diese Informationen werden in zwei getrennten Dictionaries gespeichert. Im Rahmen der Verarbeitung wird zeilenweise geprüft, in welchem Abschnitt sich die Informationen befinden. Abhängig vom jeweils aktiven Bereich werden die Werte der entsprechenden Zeile extrahiert, in Ganzzahlen konvertiert und gespeichert. Nach dem Abschluss der Extraktion gibt die Funktion zwei separate Dictionaries zurück, die jeweils die Eingangskapazitäten und die Ausgangskapazitäten der einzelnen Knotenpunkte enthalten.

Mit der darauffolgenden Funktion read_times() werden drei zeitliche Parameter aus der Instanzdatei extrahiert, die für den Simulationsdurchlauf von zentraler Bedeutung sind: die Ladezeit, die Wartezeit sowie das Diskretisierungsintervall. Die Diskretisierung gibt an, in welchem Zeitraster die Simulation abläuft, da sämtliche Zeitangaben in der Instanzdatei in Minuten vorliegen, innerhalb der Simulation jedoch als Perioden behandelt werden. Zu diesem Zweck wird die Datei erneut zeilenweise eingelesen und nach den entsprechenden Schlüsselwörtern durchsucht. Die erkannten Werte für die Lade- und Wartezeit werden anschließend durch das angegebene Diskretisierungsintervall

dividiert, um die Zeitdauer in Perioden zu berechnen, die innerhalb der Simulation verwendet wird. Nach Verarbeitung der Daten gibt die Funktion die ermittelten Werte zurück.

Die Funktion read_nodes() dient der Extraktion der Knotenpunkte und ihrer jeweiligen Eigenschaften aus der Instanzdatei. Ziel ist es, die verschiedenen im System verwendeten Knotentypen zu identifizieren und bereitzustellen. Unterschieden wird dabei zwischen „EOL“ (End of line), „Breakbulk“ und „Transfer“. Knoten vom Typ „EOL“ sind reine Start- oder Zielpunkte innerhalb der Lieferkette, an denen ausschließlich Lade- und Entladevorgänge stattfinden. Sie bilden die Schnittstelle zwischen der „ersten Meile“ und der „letzten Meile“ und dienen als Anbindungs- sowie Übergabepunkte zum Netzwerk der mittleren Meile (Belieres et al., 2025). Eine Erläuterung des Begriffs „mittlere Meile“ findet sich in Kapitel 2.3. „Breakbulk“-Knoten dienen der Aufteilung von Waren, um Güterströme neu zu organisieren, während „Transfer“-Knotenpunkte reine Umschlagpunkte darstellen, an denen Güter ohne Zwischenlagerung von einem Fahrzeug auf ein anderes überführt werden können. Diese Eigenschaften bilden die Grundlage für den Einsatz der Pufferzeitregel (siehe Kapitel 3.3.2) in der späteren Simulationslogik. Für die Extraktion der Daten wird die Instanzdatei erneut zeilenweise eingelesen und der relevante Abschnitt anhand definierter Schlüsselwörter eingegrenzt. Aus den entsprechenden Zeilen werden die Gesamtzahl der Knoten bestimmt sowie die zugehörigen Knoteninformationen extrahiert. Anschließend wird die jeweilige Knoten-ID gemeinsam mit der entsprechenden Eigenschaft in einem Dictionary gespeichert, welches am Ende der Funktion zurückgegeben wird.

Im nächsten Schritt werden die zuvor erstellten CSV-Dateien mithilfe der Funktion combine_files() zusammengeführt, um eine einheitliche und übersichtliche Datengrundlage für die Simulation zu schaffen. Zunächst werden die Pfade zu den relevanten Datenquellen sowie der Zielpfad für die zusammengeführte Datei generiert. Um eine fehlerfreie Verknüpfung zu gewährleisten, wird vor der Zusammenführung sichergestellt, dass die entsprechenden Spalten, die miteinander verknüpft werden sollen, denselben Datentyp aufweisen. Anschließend erfolgt die eigentliche Zusammenführung der Tabellen über die Strecken-ID, indem die Spalte der Fahrzeugdaten mit dem Index der Streckeninformationen verknüpft wird. Durch dieses Vorgehen werden zusätzliche Informationen wie Start- und Zielknoten sowie die Anzahl der für eine Strecke benötigten Perioden ergänzt. Aus der resultierenden Tabelle wird im Anschluss ein Teilbereich extrahiert, der ausschließlich die Informationen enthält, die für die

Simulationsdurchführung relevant sind. Dazu zählen die Fahrzeugnummer, die ID der Strecke, die Start- und Zielknoten, die Start- und Endzeitpunkte sowie die Anzahl der benötigten Perioden. Die finale und zusammengeführte Tabelle wird abschließend im CSV-Format im Ausgabeordner gespeichert und zusätzlich von der Funktion für die weitere Verwendung in der Simulation zurückgegeben.

Die letzte Funktion des Moduls stellt mit assign_commodities() zugleich den umfangreichsten Schritt dar. Ziel der Funktion ist die Zuordnung der transportierten Güter (im Folgenden als Commodities bezeichnet) zu den jeweiligen Fahrzeugen und die Ermittlung aller relevanten und ergänzenden Informationen der Güter. Grundlage dieses Vorgehens sind die zuvor in combine_files() erstellte kombinierte Datei sowie die ursprünglichen Eingabedateien. Im ersten Schritt wird die Ergebnisdatei eingelesen, um den Abschnitt zu identifizieren, der Informationen über die Anzahl und die Struktur der Transportflüsse beinhaltet. Aus den entsprechenden Zeilen werden anschließend für jede Commodity die zugehörige Strecke (Arc), die Start- und Endzeit sowie die ID des Commodities extrahiert. Transportflüsse mit einer Dauer von lediglich einer Periode werden dabei ausgenommen, da diese in der Ergebnisdatei als Warteperioden interpretiert werden. Die extrahierten Informationen werden im Anschluss in einem Dictionary abgelegt, welches für jede Commodity die relevanten Transportinformationen enthält. Daraufhin wird die kombinierte Tabelle eingelesen, um die extrahierten Transportflüsse den entsprechenden Fahrzeugen zuzuordnen. Dabei werden alle Zeilen identifiziert, bei denen sich die ID der Strecke sowie die Start- und Endzeitpunkte exakt mit einem der extrahierten Warenflüsse überschneiden. Auf diese Weise entsteht eine eindeutige Zuweisung der Fahrzeuge zu den transportierten Commodities. Im Anschluss erfolgt ein erneuter Zugriff auf die Instanzdatei, um weitere Attribute der Güter zu ermitteln. Hierzu zählen insbesondere deren jeweilige Größe sowie die spätest zulässige Ankunftszeit. Diese Informationen werden mit den bereits bestehenden Daten zusammengeführt, sodass sich für jedes Fahrzeug eine vollständige Übersicht über die transportierten Commodities ergibt. Diese umfasst unter anderem deren jeweilige IDs, die Ankunftszeitpunkte, die Einzelgrößen sowie die aufsummierte Gesamtgröße aller Commodities auf dem Fahrzeug. Abschließend werden die aufbereiteten Daten in einem pandas-DataFrame gespeichert, absteigend nach Fahrzeugnummer sortiert und im CSV-Format im Ausgabeordner abgelegt.

Die im Modul generierten Datensätze bilden die zentrale Grundlage für die Durchführung der Simulation. Die gesamte Datenaufbereitung wurde so konzipiert, dass sie unabhängig

von konkreten Eingabedateien ausgeführt werden kann, vorausgesetzt, das strukturelle Format bleibt unverändert. Auf diese Weise wird eine flexible und automatisierte Durchführung der Simulation mit unterschiedlichen Service-Netzwerken garantiert, ohne dass bei wechselnden Datensätzen manuelle Veränderungen im Quellcode erforderlich sind.

4.2 Modul zur Simulationsdurchführung

Zur Durchführung der Simulation der Transport- und Bearbeitungsprozesse innerhalb des untersuchten Netzwerks wurde das Python-Modul „Simulation“ (siehe Anhang 2) entwickelt. Dieses Modul implementiert eine ereignisdiskrete Simulation in zwei Ausprägungen. Einerseits eine Standardvariante, welche ausschließlich stochastische Einflussgrößen berücksichtigt und andererseits eine erweiterte, optimierte Variante, die zusätzliche Entscheidungsregeln implementiert. Ziel des Moduls ist es, die vorab verarbeiteten Eingabedaten unter Einbeziehung zufallsbasierter Abweichungen in ein Simulationsmodell zu übertragen, welches unter identischen Ausgangsbedingungen in beiden Varianten ausgeführt wird. Die Struktur des Modells ermöglicht dabei die Durchführung mehrerer Iterationen mit denselben stochastisch generierten Einflussgrößen, wodurch eine vergleichbare Analyse unterschiedlicher Strategien ermöglicht wird.

Vor dem Aufruf der zentralen Simulationsklasse werden zunächst mehrere externe Bibliotheken importiert, die für die technische Umsetzung der Simulation erforderlich sind. Die Bibliothek random dient der Erzeugung von Zufallszahlen und wird im weiteren Verlauf zur Modellierung der stochastischen Abweichungen von Transport- und Bearbeitungszeiten verwendet. Die Bibliothek simpy bildet ein Framework in Python ab, welches auf ereignisdiskrete Simulationen spezialisiert ist. Sie erlaubt die Modellierung und Ausführung von Prozessen, Ressourcen und Ereignissen und stellt somit das Fundament für den Ablauf dar. Eine detaillierte Darstellung der Bibliothek sowie die Erklärung aller Funktionen, die im Rahmen der Simulation aufgerufen wurden, erfolgt in Kapitel 3.3.1. Zur Datenverarbeitung wird analog zum FileHandler-Modul die Bibliothek pandas genutzt, die den effizienten Umgang mit tabellarischen Daten ermöglicht. Ergänzend wird die Bibliothek subprocess verwendet, um externe Prozesse auszuführen. Diese Bibliothek hat keinen direkten Einfluss auf die Simulation, erlaubt jedoch die

automatisierte Öffnung der erzeugten Ergebnisdateien nach Abschluss der Simulation für eine benutzerfreundliche Auswertung der Resultate.
Unmittelbar im Anschluss erfolgt die Definition der Funktion calculate_overlap(), die der Bestimmung der maximalen Anzahl gleichzeitig wartender Lieferwagen innerhalb eines gegebenen Zeitintervalls dient. Diese Information stellt die Grundlage zur Ermittlung der maximalen Warteschlangenlänge an den einzelnen Knotenpunkten des Netzwerks dar. Zur Auswertung weist die Funktion aus einer übergebenen Liste von Zeitintervallen jedem Startzeitpunkt einen positiven und jedem Endzeitpunkt einen negativen Zählerwert zu. Diese Ereignisliste wird anschließend aufsteigend nach den Zählerwerten sortiert, wobei bei identischen Zählpunkten endende Ereignisse vor beginnende Ereignisse gereiht werden. Dies verhindert, dass Intervalle mit einer Wartezeit von null fälschlicherweise als Überlappung in der Warteschlange interpretiert werden. Im nachfolgenden Schritt wird die Anzahl der gleichzeitig aktiven Intervalle durch Aufaddieren der Zählerwerte ermittelt. Der dabei erreichte Maximalwert entspricht der höchsten Zahl simultan wartender Einheiten und gilt als Maß für die maximale Warteschlangenlänge an einem Knoten. Dieser Wert wird am Ende der Funktion zurückgegeben.

4.2.1 Ablauf und Steuerung der Simulationsprozesse

Nach dem Import der erforderlichen Bibliotheken und der Definition der vorbereitenden Funktion wird mit der Klasse „TruckSimulation“ das zentrale Steuerungselement der Simulation implementiert. Sie bildet das technische und logische Gerüst zur Darstellung der Transport- und Bearbeitungsprozesse innerhalb des betrachteten Netzwerks ab und beinhaltet sämtliche dafür relevanten Parameter, Daten und Methoden. Die Initialisierung der Simulationsklasse erfolgt über den Konstruktor __init__(), der eine Vielzahl an Eingabewerten übergibt, die zur Konfiguration und Durchführung der Simulation benötigt werden. Zu Beginn werden die Pfade übergeben, die bereits im FileHandler-Modul aufbereitet wurden. Einerseits wird die kombinierte Datei mit den Daten zu den Lieferwagen und den dazugehörigen Streckeninformationen eingelesen und andererseits erfolgt das Einlesen der Datei mit den zugeordneten Commodities. Anschließend werden die Parameter der Kapazitäten eingebunden, welche die verfügbare Anzahl an Toren je Knotenpunkt bestimmen. Zur Modellierung der stochastischen Unsicherheiten werden für die Fahrtzeiten sowie für die Be- und Entladezeiten an den Ladetüren die jeweils oberen und unteren Abweichungsgrenzen in Prozent übergeben. Anschließend werden

zwei Variablen integriert, die ausschließlich im optimierten Simulationsmodus zur Anwendung kommen. Dabei handelt es sich zum einen um einen definierten Zeitpuffer, der die Simulation zur Bündelung von Lieferfahrzeugen temporär pausiert und zum anderen um die individuell vorgegebene maximale Anzahl zusätzlicher Ladetore. Zusätzlich wird eine binäre Variable eingebunden, die die Auswahl zwischen der Standard- und der optimierten Simulationsvariante ermöglicht. Darüber hinaus werden über einen weiteren Parameter zusätzliche Eigenschaften der Knotenpunkte, wie beispielsweise deren Typen, bestimmt. Abschließend stellt ein letzter Parameter sicher, dass in beiden Simulationsdurchläufen identische stochastische Bedingungen vorliegen, indem die generierte Menge an Zufallswerten gespeichert wird. Wird dieser Parameter nicht definiert, generiert das Modell bei jeder Initialisierung neue Zufallsabweichungen, wodurch ein direkter Vergleich der Ergebnisse nicht möglich ist.

Nach der Übergabe der Konfigurationsparameter erfolgt deren Zuweisung zu internen Objektvariablen innerhalb der Initialisierung. Dadurch stehen sämtliche relevante Parameter während der gesamten Laufzeit der Simulation konsistent in der Instanz zur Verfügung und können innerhalb weiterer Methoden flexibel verarbeitet werden. Außerdem werden die zentralen Eingangsdaten aus den zuvor im FileHandler-Modul generierten CSV-Dateien eingelesen. Die kombinierte Datei enthält alle relevanten Transportinformationen, wie beispielsweise die Start- und Zielknoten oder die individuellen Transportzeiten. Nach dem Einlesen werden diese Informationen im Attribut self.truck_data abgelegt und bilden die Grundlage der Lieferprozesse. Zusätzlich wird die Datei mit den Commodity-Informationen eingelesen, in der zum Beispiel die für jedes Fahrzeug zugewiesenen Commodity-IDs dokumentiert sind. Diese Informationen werden ebenfalls gespeichert und dienen im weiteren Verlauf insbesondere zur Überprüfung der Einhaltung von Lieferfristen.

Im Anschluss erfolgt die Initialisierung der stochastischen Abweichungen für die Transport- und Bearbeitungszeiten. Ziel dieses Schrittes ist es, für jede Einheit individuelle Zufallsgrößen zu erzeugen, um realitätsnahe und variierende Prozesslaufzeiten im Simulationsmodell abzubilden. Die Generierung der Abweichungen erfolgt für drei verschiedene Zeitgrößen pro Fahrzeug: die Fahrtzeit, die Bearbeitungszeit am Ausgangsknoten (Beladung) und die Bearbeitungszeit am Zielknoten (Entladung). Für die Berechnung wird die Funktion random.triangular() verwendet, welche eine Zufallszahl auf Basis einer Dreiecksverteilung erzeugt. Diese Form der Verteilung ist durch ein Minimum, ein Maximum und einen Modus charakterisiert. In der vorliegenden

Implementierung werden die unteren und oberen Schranken durch benutzerdefinierte Abweichungsgrenzen in Prozent definiert, während der Modus auf null Prozent festgelegt ist. Dadurch wird eine asymmetrische Verteilung sichergestellt, in der geringe Abweichungen um den Sollwert wahrscheinlicher sind als extreme Ausreißer in Richtung der Grenzen. Die resultierenden Abweichungswerte werden für jedes Fahrzeug in einem Dictionary gespeichert, welches für alle nachfolgenden Schritte der Simulation zur Verfügung steht. Sollte bereits beim Aufruf der Simulationsklasse ein solches Dictionary übergeben worden sein, wird dieses übernommen. Auf diese Weise kann sichergestellt werden, dass die Simulation mit einer konstanten Menge an Zufallswerten durchgeführt wird. Dies ermöglicht insbesondere bei mehreren Iterationen die Erzeugung von vergleichbaren Simulationsergebnissen mit identischen stochastischen Rahmenbedingungen.

Im nächsten Schritt wird die Simulationsumgebung durch das Anlegen eines simpy.Environment()-Objekts initialisiert. Dieses Objekt stellt den zentralen Rahmen dar, innerhalb dessen sämtliche Prozesse und Ressourcen während der gesamten Simulationsdauer hinweg verwaltet und koordiniert werden.

Anschließend werden die erforderlichen internen Datenstrukturen vorbereitet, indem verschiedene Dictionaries und Listen für die Erfassung und Verwaltung der Informationen angelegt werden. Beispielsweise werden Dictionaries definiert, um die Kapazitäten an den einzelnen Knotenpunkten oder die Anzahl der wartenden Lieferfahrzeuge zu speichern. Ergänzend wird die Liste self.truck_records vorbereitet, in der im weiteren Verlauf detaillierte Informationen über den tatsächlichen Ablauf der einzelnen Transportprozesse gespeichert werden. Für den Fall, dass die Simulation im optimierten Modus ausgeführt wird, werden zusätzliche Variablen zur Umsetzung der Entscheidungsregeln initialisiert. Dazu gehören Zähler zur Erfassung der zusätzlich geöffneten Ladetore, Dictionaries zur Ermittlung der aktuellen Warteschlangenlänge an den einzelnen Knotenpunkten sowie zwei weitere Dictionaries zur Zählung der Lieferfahrzeuge, die im Rahmen der Entscheidungsregel zusätzlich geöffnete Ladetüren genutzt haben.

Im Anschluss erfolgt die Ermittlung aller im Datensatz vorkommenden Knotenpunkte, indem alle Start- und Zielknoten aus der Tabelle extrahiert und mithilfe einer Funktion zu einer gemeinsamen Menge zusammengeführt werden. Diese Menge bildet die Grundlage für die Ressourcen, die mit dem Aufruf der Methode initialize_resources() initialisiert werden.

Im Rahmen dieser Methode wird für jeden Knotenpunkt jeweils eine Ressource angelegt und mit der entsprechenden Kapazität verknüpft. Die Kapazitätswerte werden zunächst aus den zuvor eingelesenen Datensätzen extrahiert. Sollte für einen Knoten keine Angabe vorliegen, wird standardmäßig eine Kapazität von eins zugewiesen, um einen unterbrechungsfreien Simulationsablauf sicherzustellen. Je nach gewähltem Simulationsmodus unterscheidet sich die Art der Ressourcenzuordnung. In der Standardvariante werden die Ressourcen als einfache simpy.Resource angelegt, bei denen die Zuweisung der Zugriffe streng nach dem First-Come-First-Served-Prinzip erfolgt. Im optimierten Modus hingegen kommen simpy.PriorityResource-Objekte zum Einsatz, die eine prioritätsbasierte Steuerung der Ressourcenzugriffe gewährleisten. Dadurch können insbesondere Lieferwagen mit zeitkritischeren Anforderungen bevorzugt behandelt werden. Darüber hinaus werden für jeden Knotenpunkt interne Zähl- und Speicherstrukturen initialisiert. Dazu gehören Listen zur Erfassung der individuellen Warteintervalle oder Zähler für die Anzahl wartender Fahrzeuge. Im optimierten Modus werden zusätzlich die aktuellen Warteschlangenlängen an den Knoten erfasst, um im weiteren Verlauf dynamische Entscheidungen, wie etwa die Erweiterung von Ressourcenkapazitäten treffen zu können. Des Weiteren wird in diesem Modus dokumentiert, welche Fahrzeuge die zusätzlichen Ressourcen tatsächlich in Anspruch nehmen, um den Nutzen der Kapazitätserweiterung bewerten zu können.

Die nachfolgenden Methoden run_iteration1() und run_iteration2() sind für die Ausführung der beiden Simulationsvarianten verantwortlich. Sie steuern die Initialisierung, die Durchführung und die anschließende Auswertung eines vollständigen Simulationsdurchlaufs. Die Methode run_iteration1() führt die Simulation in der Standardkonfiguration aus. Hierbei wird für jeden Transporter ein separater SimPy-Prozess erzeugt, indem die Methode truck_process() aufgerufen wird. Alle generierten Prozesse werden anschließend in die SimPy-Umgebung integriert. Diese wird gestartet und solange ausgeführt, bis sämtliche definierten Ereignisse abgearbeitet wurden. Im Anschluss erfolgt die Ergebnisauswertung für die Standard-Simulationsvariante durch den Aufruf der Methode print_statistics(), welche für die erste Iteration mit der Kennzeichnung „1“ erfolgt.

Die Methode run_iteration2() dient der Durchführung der optimierten Simulationsvariante. Zu Beginn wird eine neue SimPy-Umgebung aufgesetzt, um sämtliche vorherige Ereignisse zurückzusetzen. Zusätzlich werden alle Ressourcen- und Speicherstrukturen neu initialisiert, um einen konsistenten Ausgangszustand

wiederherzustellen. Wie in der ersten Iteration wird anschließend für jede Transporteinheit ein neuer Prozess angelegt und in die Umgebung eingebunden. Nach dem Start der Simulation und der Abarbeitung aller Prozesse erfolgt erneut die Ergebnisauswertung über die Methode print_statistics(), diesmal mit der Kennzeichnung „2“ für die optimierte Variante.

Das Kernstück für die Abbildung der Lebenszyklen der einzelnen Transportvorgänge innerhalb der Simulation bildet die Methode truck_process(). Zu Beginn werden für jede Transporteinheit die wesentlichen Eingangsparameter aus der zuvor erstellten Transportdatentabelle extrahiert. Dazu zählen die eindeutige Identifikationsnummer des Fahrzeugs, der Start- und Zielknoten, die geplanten Start- und Endzeitpunkte sowie die ursprünglich geplante Reisedauer in Perioden. Ergänzend wird die deterministische Wartezeit (det_truck_waiting) berechnet, welche sich aus der Differenz zwischen der geplanten Gesamtzeit und der Summe der regulären Reise- und Bearbeitungszeiten ergibt. Diese deterministische Wartezeit wird ausschließlich in der Standard-Simulation berücksichtigt und dient zur Nachbildung der geplanten Wartephase innerhalb des Transportablaufs.

Im nächsten Schritt erfolgt die Ermittlung der verbleibenden Zeit bis zur frühesten Deadline der transportierten Waren. Hierzu wird aus der Commodity-Tabelle die Zeile selektiert, die dem aktuellen Fahrzeug zugeordnet ist. Aus diesen Informationen werden die zugehörige geplante Ankunftszeit und die früheste Deadline der transportierten Waren ermittelt. Die verbleibende Zeit, die im weiteren Verlauf der Simulation im Parameter remaining_time gespeichert wird, ergibt sich als Differenz zwischen der frühesten Deadline und der geplanten Endzeit des Fahrzeugs. Diese Größe wird im weiteren Verlauf insbesondere im optimierten Modus verwendet, um den Einsatz der Entscheidungslogiken steuern zu können.

Im Anschluss werden die zuvor ermittelten stochastischen Abweichungen auf die einzelnen Zeitgrößen der Transporteinheiten angewendet. Hierzu wird für jedes Fahrzeug das entsprechende Dictionary abgerufen, welches die prozentualen Abweichungen für die Fahrtzeit sowie für die Bearbeitungszeiten am Start- und Zielknoten enthält. Sollten für ein Fahrzeug keine Daten hinterlegt sein, werden sämtliche Abweichungen standardmäßig auf null Prozent gesetzt. Auf Basis dieser Abweichungswerte erfolgt anschließend die Anpassung der planmäßigen Zeitgrößen. Die Fahrtzeit sowie die Bearbeitungszeiten an beiden Knotenpunkten werden proportional mit dem jeweiligen stochastischen Faktor multipliziert. Um sicherzustellen, dass keine unrealistisch kurzen

Fahrtzeiten oder negativen Bearbeitungszeiten entstehen, wird eine Mindestfahrtzeit von einer Zeitperiode definiert und negative Bearbeitungszeiten werden auf null gesetzt. Diese Regulierungen gewährleisten eine konsistente und realitätsnahe Abbildung der Prozesse innerhalb der Simulation.

Nach der Anpassung der Zeitgrößen erfolgt die Integration der Lieferprozesse in die Simulationsumgebung. Hierbei wird zunächst geprüft, ob der geplante Startzeitpunkt des jeweiligen Fahrzeugs mit der aktuellen Simulationszeit übereinstimmt. Befindet sich die Simulation vor dem vorgesehenen Start, wird der Lieferwagen in einen passiven Wartezustand versetzt und der Simulationsschritt wird über einen Timeout simuliert. Sobald der geplante Startzeitpunkt erreicht ist, wird der tatsächliche Startzeitpunkt des Lieferwagens erfasst und dokumentiert. Dieser Wert dient im weiteren Verlauf als Referenzgröße für die spätere Auswertung und ermöglicht die Ermittlung der Pünktlichkeit. Zur Dokumentation während der Simulationsdurchführung wird zusätzlich eine Ausgabemeldung im Terminal erzeugt. Diese enthält die ID des Fahrzeugs, den Startzeitpunkt, den Start- und Zielknoten sowie die auf Basis der stochastischen Abweichung angepasste Fahrtzeit. Diese Meldung dient ausschließlich der laufenden Überprüfung im Terminal und erscheint nicht in den im Anschluss erstellten Auswertungsdateien.

Im weiteren Verlauf werden die Kernelemente der Transportprozesse modelliert, indem die Abläufe an den Start- und Zielknotenpunkten detailliert abgebildet werden. Im Fokus stehen hierbei die Steuerung des Ressourcenzugriffs sowie die Implementierung definierter Entscheidungsregeln zur Optimierung der Ressourcennutzung.

Zu Beginn erfolgt die Bearbeitung am Startknoten. Hierzu wird im ersten Schritt die zuvor initialisierte Ressource für den jeweiligen Startknoten aus dem entsprechenden Dictionary abgerufen. Im Anschluss daran wird, ausschließlich im optimierten Modus, eine dynamische Entscheidungslogik zur Erweiterung der verfügbaren Ladekapazitäten implementiert. Für dieses Vorgehen wird überprüft, ob die aktuelle Länge der Warteschlange am Startknoten einen definierten Schwellenwert von zwei überschreitet. Ist dies der Fall und wurde die vom Benutzer definierte maximale Anzahl an zusätzlich zulässigen Ladetüren (door_number) noch nicht erreicht, so wird die Kapazität der betreffenden Ressource um eine weitere Ladetür erhöht. Auf diese Weise kann eine erhöhte Nachfrage am Knotenpunkt flexibel bedient werden. Zusätzlich wird das Öffnen eines weiteren Ladetores durch eine Ausgabemeldung im Terminal dokumentiert. Unabhängig von der Simulationsvariante wird anschließend der Zeitpunkt zu Beginn der

Wartephase an der Ressource gespeichert. Diese Information bildet unter anderem die Grundlage zur späteren Ermittlung der tatsächlichen Wartezeiten.

Im weiteren Ablauf wird eine weitere Besonderheit des optimierten Modells umgesetzt. Für dieses wird unter bestimmten Voraussetzungen vor dem Zugriff auf die Ressource eine künstliche Verzögerung eingebaut, um eine gezielte Bündelung von Fahrzeugen zu ermöglichen. Diese zusätzliche Wartezeit (batching_delay) kommt nur zum Einsatz, wenn folgende Bedingungen gleichzeitig erfüllt sind. Erstens darf sich zum aktuellen Zeitpunkt kein anderes Fahrzeug in der Warteschlange am betreffenden Knoten befinden. Zweitens muss der Knotenpunkt die im Vorfeld extrahierte Eigenschaft als „Breakbulk“ oder „Transfer“ aufweisen. Drittens muss noch ausreichend Zeit bis zur frühesten Deadline der verladenen Waren verbleiben. Hierzu wird der zuvor berechnete Parameter remaining_time herangezogen. Dieser Wert muss positiv sein, sodass eine zusätzliche Verzögerung die termingerechte Lieferung nicht gefährdet. Sind diese Voraussetzungen erfüllt, wird eine künstliche Verzögerung in Form eines Timeouts eingefügt. Ziel dieser Maßnahme ist es, Lieferwagen, die aufgrund von stochastischen Verzögerungen zeitlich leicht versetzt am Knotenpunkt eintreffen würden, gezielt zu bündeln. Durch das kontrollierte Anstauen mehrerer Fahrzeuge bleibt die Prioritätsvergabe innerhalb der Ressource, die auf Basis der geplanten Ankunftszeit erfolgt, weiterhin gewährleistet. Insbesondere Fahrzeuge mit zeitkritischeren Deadlines werden dadurch bevorzugt abgefertigt, obwohl sie aufgrund der zufälligen Transportzeitabweichungen und der leicht verzögerten Ankunft am Knotenpunkt andernfalls benachteiligt worden wären.

Im Anschluss erfolgt die Anforderung der Ressource am Startknoten durch das Fahrzeug. Im Standardmodus wird eine normale Anforderung an die einfache Ressource gestellt, während im optimierten Simulationsmodus eine priorisierte Anforderung an die Prioritätsressource erfolgt. Als Prioritätskriterium dient dabei die geplante Ankunftszeit des Lieferwagens.

Nach Erstellung der Anforderung wartet das Fahrzeug, bis eine freie Kapazität an den Ladetoren für das Beladen verfügbar ist. Mit dem erfolgreichen Zugriff auf die Ressource erfolgt die Dokumentation des Endzeitpunkts der Wartephase. Darüber hinaus wird im optimierten Modus überprüft, ob ein Lieferfahrzeug ein zusätzlich geöffnetes Ladetor genutzt hat. Hierzu wird während der Ressourcennutzung kontrolliert, ob die aktuell belegte Kapazität des Ressourcenobjekts die ursprünglich definierte Standardkapazität überschreitet. Trifft dies zu, wird das Fahrzeug dem entsprechenden Dictionary

hinzugefügt, um in der späteren Auswertung die tatsächliche Nutzung der zusätzlichen Kapazitäten zu ermitteln.
Im nächsten Schritt wird die tatsächliche Wartezeit des Fahrzeugs ermittelt, indem die Differenz zwischen dem Ende und dem Beginn der Wartephase berechnet wird. Sobald der Wert von null überschritten wurde und der Lieferwagen somit warten musste, wird das zugehörige Zeitintervall in der entsprechenden Liste für den Startknoten gespeichert und für die spätere Analyse der maximalen Warteschlangenlänge genutzt. Zusätzlich wird der Zähler für die Anzahl der am Startknoten wartenden Fahrzeuge um eine Einheit erhöht.
Nach Erfassung der Wartezeit beginnt die Bearbeitungsphase am Startknoten, die den Beladevorgang der Lieferwagen abbildet. Die Bearbeitungsdauer basiert auf der zuvor individuell angepassten, stochastischen Ladezeit am Startknoten und wird durch einen Timeout simuliert. Nach Abschluss des Beladens wird die Ressource freigegeben, sodass nachfolgende Fahrzeuge auf die Kapazität zugreifen können. Im direkten Anschluss wird die Reisezeit der Lieferwagen zwischen Start- und Zielknoten simuliert. Dieses Vorgehen erfolgt über einen weiteren Timeout, basierend auf der stochastisch angepassten Fahrtzeit. Nach Abschluss der Fahrt erreicht das Fahrzeug den jeweiligen Zielknoten.
Bevor jedoch der eigentliche Zugriff der Ressource am Zielknoten erfolgt, ist im Standardmodell eine zusätzliche deterministische Wartezeit zu berücksichtigen. Diese ergibt sich aus dem im Vorfeld berechneten Wert det_truck_waiting, der die ursprünglich vorgesehene Wartezeit im planmäßigen deterministischen Ablauf ohne stochastische Schwankungen widerspiegelt. Um diese Wartephase im Modell konsistent darzustellen, wird geprüft, ob sich die Simulation im Standardmodus befindet und ob der berechnete Wert positiv ist. Sobald diese Bedingungen erfüllt sind, wird das Lieferfahrzeug für die Dauer der deterministisch berechneten Wartezeit mithilfe eines Timeouts in einen passiven Wartezustand versetzt. Beginn und Ende dieser Wartephase werden dokumentiert und das entsprechende Warteintervall im Datensatz abgelegt. Zudem wird der Zähler für die Anzahl der am Zielknoten wartenden Fahrzeuge im Standardmodell um eine Einheit erhöht.
Im Anschluss erfolgt der Zugriff auf die Ressource am Zielknoten. Hierzu wird zunächst die entsprechende Ressource aus dem zuvor definierten Dictionary geladen. Im optimierten Simulationsmodus wird anschließend erneut die dynamische Logik zur Erweiterung der Ladetore umgesetzt, analog zum Vorgehen am Startknoten. Sobald die aktuelle Warteschlange am Zielknoten einen definierten Schwellenwert überschreitet und

die maximal zulässige Anzahl zusätzlicher Türen noch nicht erreicht wurde, wird die Ressourcenkapazität um eine weitere Einheit erhöht. Unmittelbar danach wird der Beginn einer potenziellen Wartezeit an der Ressource dokumentiert. Wie bereits am Startknoten kann im optimierten Modus unter bestimmten Voraussetzungen eine zusätzliche Verzögerung eingefügt werden, um mehrere Lieferwagen gezielt zu bündeln und priorisiert abzufertigen. Diese Verzögerung tritt nur dann in Kraft, wenn aktuell keine anderen Fahrzeuge auf die Ressource zugreifen, der Knoten die Eigenschaft „Breakbulk“ oder „Transfer“ aufweist und planmäßig noch ausreichend Zeit bis zur frühesten Deadline verbleibt.

Anschließend erfolgt die Anforderung der Ressourcen abhängig vom gewählten Simulationsmodell. Im Standardmodell wird eine reguläre Ressourcenzuteilung vorgenommen, im optimierten Simulationsmodell hingegen eine priorisierte Anforderung, bei der die geplante Endzeit des Lieferfahrzeugs als Prioritätskriterium dient. Nach erfolgter Zuweisung der entsprechenden Ressource wird erneut geprüft, ob das Fahrzeug eine der zusätzlich geöffneten Ladetüren genutzt hat. In diesem Fall wird die Fahrzeugkennung entsprechend hinterlegt. Im Anschluss wird der Endzeitpunkt der Wartephase dokumentiert und die aktuelle Warteschlangenlänge um eine Einheit reduziert.

Nach dem Zugriff auf die Ressourcen wird, wie bereits am Startknoten, die tatsächliche Wartezeit berechnet. Ist die Differenz zwischen Beginn und Ende der Wartephase größer als null und besteht somit eine tatsächliche Wartezeit, wird das zugehörige Intervall im Dictionary abgelegt. Zusätzlich wird der Zähler der wartenden Lieferwagen am Zielknoten um eine Einheit erhöht.

Im nächsten Schritt wird der Entladevorgang am Zielknoten simuliert. Grundlage hierfür ist die zuvor individuell angepasste Bearbeitungsdauer, die in der SimPy-Umgebung mithilfe eines Timeouts abgebildet wird. Nach Abschluss des Entladeprozesses wird die Ressource wieder freigegeben, um die Kapazität für nachfolgende Lieferwagen bereitzustellen. Abschließend wird der tatsächliche Endzeitpunkt des jeweiligen Transports erfasst.

Daraufhin wird überprüft, ob die Lieferfahrzeuge die planmäßige Ankunftszeit einhalten konnten. Dazu wird der gemessene tatsächliche Endzeitpunkt mit der ursprünglich vorgesehenen Endzeit verglichen. Ist der tatsächliche Zeitpunkt kleiner oder gleich der geplanten Ankunftszeit, wird das Fahrzeug als „pünktlich“ klassifiziert, andernfalls als „verspätet“. Diese Bewertung wird als Text gespeichert und später in der Auswertung

genutzt. Dabei wird ausschließlich die Pünktlichkeit der Lieferwagen in Bezug auf deren geplante Ankunftszeit betrachtet. Ein Abgleich mit der frühesten Deadline der transportierten Waren findet an dieser Stelle nicht statt.
Im weiteren Verlauf wird die gesamte effektive Wartezeit der Fahrzeuge berechnet. Diese setzt sich aus der Summe der tatsächlich aufgetretenen Wartezeiten an Start- und Zielknoten zusammen. Im Standardmodell wird zusätzlich die deterministisch berechnete Wartezeit addiert, die nur in dieser Modellvariante zur Anwendung kommt. Unabhängig vom aktiven Modus wird die effektive Wartezeit um jene Zeitspanne reduziert, die durch stochastisch bedingte Verlängerungen der Bearbeitungsprozesse entstanden sind. Diese werden zwar innerhalb der Wartephasen miterfasst, stellen jedoch keine effektive Wartezeit für die Lieferfahrzeuge dar und werden daher bei der Berechnung entsprechend korrigiert.
Abschließend werden sämtliche relevanten Informationen zum Transport für jedes einzelne Fahrzeug in einem strukturierten Datensatz zusammengeführt. Dieser wird in einer zentralen Ergebnisliste gespeichert und dient als Grundlage für die anschließende statistische Auswertung der Simulationsergebnisse.

4.2.2 Ergebnisaufbereitung und Datenexport

Nach Abschluss der Simulationsdurchführung folgt die Auswertung der aufgezeichneten Prozessdaten. Ziel dieser Auswertung ist es, Leistungskennzahlen wie Wartezeiten, Ressourcenauslastungen und Termintreue übersichtlich darzustellen und vergleichend gegenüberzustellen. Die folgenden Abschnitte beschreiben das methodische Vorgehen zur Ergebnisanalyse der im Simulationsverlauf gesammelten Informationen.
Die Methode print_statistics() stellt die zentrale Auswertungsfunktion des Simulationsmoduls dar. Sie wird nach dem Abschluss einer vollständigen Iteration aufgerufen und kommt sowohl im Standard- als auch im optimierten Modus zum Einsatz. Über das Argument „iteration“ wird definiert, mit welcher Kennzeichnung die ausgewerteten Ergebnisdateien gespeichert werden. Dabei wird dem Standardmodell die Kennzeichnung „1“ zugewiesen, während die optimierte Variante mit der Kennziffer „2“ versehen wird.
Im ersten Schritt wird eine alphabetisch sortierte Liste aller im Netzwerk vorkommenden Knotenpunkte generiert. Für jeden dieser Knoten wird anschließend mithilfe der Methode calculate_overlap() die maximale Warteschlangenlänge an den Ausgangs- und an den

Eingangsknoten ermittelt. Neben der Warteschlangenlänge wird zusätzlich die Anzahl der Fahrzeuge erfasst, die vor den einzelnen Knotenpunkten eine Wartezeit aufweisen. Die dabei ermittelten Informationen werden in vier separaten Datenstrukturen zwischengespeichert: die maximale Warteschlangenlänge an den Startknoten, die Anzahl wartender Fahrzeuge an den Startknoten, die maximale Warteschlangenlänge an den Zielknoten sowie die Anzahl der wartenden Lieferwagen an den Zielknoten. Diese Kennzahlen werden darauf in einem strukturierten Datensatz zusammengeführt. Hierzu wird ein DataFrame mit dem Namen „df_queue“ erstellt, der für jeden Knotenpunkt die vier genannten Kennzahlen enthält. Ergänzend wird eine aggregierte Zeile mit der Bezeichnung „Summary“ an den Datensatz angehängt. Diese fasst die Gesamtergebnisse über alle Knotenpunkte hinweg zusammen. Sie enthält zum einen die Summe aller wartenden Fahrzeuge an Start- und Zielknoten und zum anderen den höchsten über alle Knoten hinweg auftretenden Wert der maximalen Warteschlangenlänge, getrennt nach Start- und Zielknoten. Die zusammenfassende Zeile wird an den DataFrame angefügt und vervollständigt somit den Datensatz für das erste auswertende Ergebnisdokument.

Im Anschluss an die Analyse der ressourcenbezogenen Kennzahlen erfolgt die Aufbereitung der Daten der transportbezogenen Prozessdaten. Als Grundlage hierfür dient die zuvor während der Simulation befüllte Liste self.truck_records, die für jede Transporteinheit die relevanten Ablaufdaten enthält. Diese Informationen werden in einem strukturierten DataFrame mit dem Namen „df_trucks“ zusammengeführt. Der Datensatz enthält die zentralen Parameter zum Ablauf jedes Lieferfahrzeugs. Darunter zählen die Fahrzeug-ID, die Start- und Zielknoten, die geplanten Start- und Endzeitpunkte, die ermittelte Pünktlichkeit des Lieferwagens sowie die gemessene effektive Wartezeit. Zur besseren Übersicht wird der Datensatz nach absteigender Fahrzeug-ID sortiert.

Im nächsten Schritt werden diese Transportdaten mit der Commodity-Datei verknüpft, um die Fahrzeugdaten mit Informationen zu den transportierten Waren zu ergänzen. Zu diesem Zweck wird die CSV-Datei der Waren erneut eingelesen und auf Grundlage der Fahrzeug-IDs mit dem DataFrame df_trucks zusammengeführt. Das Ergebnis ist ein neuer Datensatz mit der Bezeichnung „df_merged“, der die Informationen beider Dateien in zusammengeführter Form enthält.

Da sich auf einem Fahrzeug mehrere verschiedene Waren befinden können, ist für die Bewertung der Pünktlichkeit eine Betrachtung auf Warenebene erforderlich. Hierzu wird die Funktion compare_end() definiert, welche für jede Transporteinheit prüft, ob alle

beladenen Waren die jeweils vorgegebene Deadline einhalten können. Die Funktion enthält dafür drei Eingabeparameter: den tatsächlichen Endzeitpunkt des Lieferwagens, eine Liste der zugeordneten Commodity-IDs und die jeweils spätesten zulässigen Ankunftszeitpunkte. Zunächst wird überprüft, ob für das betrachtete Fahrzeug überhaupt Warendaten vorliegen oder ob diesem keine zugeordnet wurden. Im Anschluss erfolgt eine Validierung, ob für jeden Warenstrom entsprechende Deadlinedaten vorliegen und ob deren Format für die weitere Bearbeitung geeignet ist. Sind alle Voraussetzungen erfüllt, durchläuft die Funktion sämtliche Commodities und vergleicht deren Deadline mit der tatsächlichen Ankunftszeit des zugehörigen Lieferfahrzeugs. Für jede transportierte Ware wird auf diese Weise überprüft, ob die Zustellung innerhalb der vorgegebenen Frist erfolgt oder ob eine zeitliche Verspätung vorliegt. Wird die Deadline eingehalten, wird dies mit der Kennzeichnung „on time“ dokumentiert. Bei Überschreitung wird die Höhe der Verspätung berechnet und ebenfalls als Textausgabe formatiert. Das Ergebnis der Funktion ist eine Zeichenkette, die für jedes Fahrzeug eine Statusmeldung aller transportierten Waren enthält. Die resultierenden Auswertungen werden abschließend in einer neuen Spalte zum bestehenden DataFrame df_merged hinzugefügt.

Anschließend erfolgt eine standardisierte Formatierung ausgewählter numerischer Größen, mit dem Ziel, die Übersichtlichkeit der Auswertung zu verbessern. Konkret betrifft dies die Spalten für die tatsächlich gemessenen Start- und Endzeitpunkte sowie die tatsächliche Wartezeit der Lieferfahrzeuge, deren Werte jeweils auf zwei Dezimalstellen gerundet werden.

Zur abschließenden Bewertung der Modellergebnisse werden zwei zusätzliche Zeilen in den bestehenden Datensatz eingefügt. Die erste Ergänzung erfasst die Gesamtzahl aller verspätet zugestellten Waren. Als Grundlage dieser Auswertung dient dabei die Spalte, in der zuvor die Termintreue der einzelnen Commodities ausgewertet wurde. Darin werden alle Einträge identifiziert, die mit einem Hinweis auf eine Verspätung versehen sind, um daraus die Anzahl aller verspäteten Lieferungen zu bestimmen. Das Ergebnis wird in Form einer neuen Zeile im DataFrame festgehalten. Hierzu wird ein Dictionary erzeugt, in dem die ermittelten Kennzahlen in den entsprechenden Spalten abgelegt werden.

Darüber hinaus wird eine zweite Kennzahl ergänzt, welche die durchschnittliche Wartezeit aller Transporteinheiten darstellt. Für die Berechnung wird auf die tatsächliche Wartezeit der Fahrzeuge zurückgegriffen und der arithmetische Mittelwert über alle gültigen Einträge gebildet. Der resultierende Durchschnittswert wird auf zwei

Dezimalstellen gerundet und analog zur vorherigen Kennzahl als zusätzliche Zeile am Ende des Ergebnisdatensatzes hinzugefügt.

Im nächsten Schritt wird analysiert, ob und in welchem Umfang die zusätzlich geöffneten Ladetüren im Rahmen des optimierten Simulationsmodus tatsächlich von Lieferfahrzeugen genutzt wurden. Hierzu wird zunächst überprüft, ob das Öffnen zusätzlicher Türen im Modell erlaubt war. Ist dies der Fall, werden die entsprechenden Datenstrukturen ausgewertet, in denen die betroffenen Fahrzeuge gespeichert wurden. Sowohl für den Start- als auch für den Zielknoten wird die Anzahl der Lieferfahrzeuge berechnet, die auf die zusätzliche Ressource zugegriffen haben. Für beide Bereiche werden die Einträge in den zugehörigen Dictionaries summiert. Die ermittelten Werte geben Aufschluss über die tatsächliche Nutzung der optionalen Ladetüren und werden abschließend im Terminal ausgegeben.

Nachfolgend erfolgt eine ergänzende Auswertung der verspätet zugestellten Waren. Als Grundlage dienen die bereits ermittelten Verspätungen der einzelnen Warenströme aus den Simulationsergebnissen. Für jede Zeile wird geprüft, ob eine Verspätung vorliegt. Sofern dies zutrifft, wird der entsprechende Wert in einer Liste gespeichert. Die gewonnenen Werte werden anschließend sowohl als Gesamtsumme berechnet als auch zur Ermittlung der Durchschnittswerte über alle Einträge hinweg genutzt. Liegen keine Informationen vor, werden beide Kennzahlen mit null gespeichert. Die berechneten Kennzahlen geben Aufschluss über die Gesamtverzögerung sowie die durchschnittliche Dauer der Verspätungen innerhalb des Netzwerks. Die Darstellung der Ergebnisse erfolgt im Anschluss im Terminal.

Abschließend erfolgt der Export der beiden erstellten Ergebnistabellen. Dafür werden sowohl die Auswertung der Ressourcenauslastung an den Knotenpunkten als auch die Analyse der Transportprozesse unter definierten Dateipfaden zur Ausgabe gespeichert. Die Dateinamen der Pfade beinhalten eine Kennzeichnung der jeweiligen Iteration, wodurch die Ergebnisse beider Simulationsmodelle getrennt gespeichert und unterschieden werden können.

Zur Unterstützung einer benutzerfreundlichen Auswertung werden alle erzeugten Dateien unmittelbar nach dem Export automatisiert für eine direkte Einsicht geöffnet. Dies geschieht über einen systemseitigen Aufruf mithilfe der zuvor importierten Bibliothek subprocess, die die entsprechenden CSV-Dateien im Standardanzeigeprogramm des verwendeten Betriebssystems öffnet.

4.3 Modul zur Ausführung

Das Main-Modul (siehe Anhang 3) stellt das zentrale Steuerungselement der Simulation dar. Der Ablauf innerhalb dieses Moduls folgt einer fest vorgegebenen Struktur. Zunächst werden sämtliche erforderlichen Eingabedaten eingelesen, verarbeitet und miteinander verknüpft. Anschließend wird der Nutzer dazu aufgefordert, verschiedene Eingabeparameter zu definieren, darunter den Schwankungsbereich der simulierten Transport- und Bearbeitungszeiten, die Dauer einer optional eingesetzten Pufferzeit zur gezielten Bündelung von Lieferfahrzeugen sowie die Anzahl zusätzlich verfügbarer Ladetore. Basierend auf diesen Angaben werden sowohl der Standarddurchlauf als auch der optimierte Simulationslauf, wie in den vorhergehenden Kapiteln beschrieben, initialisiert. Die konkrete Umsetzung der im Main-Modul ablaufenden Prozessschritte wird im Folgenden detailliert erläutert.

Im ersten Abschnitt des Moduls werden alle für die Durchführung der Simulation erforderlichen Eingabedaten geladen und aufbereitet. Zu diesem Zweck werden zunächst die zuvor beschriebenen Module „FileHandler“ und „Simulation“ importiert. Im Anschluss daran erfolgen die Aufrufe der im Modul „FileHandler“ definierten Funktionen, durch die sämtliche Netzwerk-, Transport- und Zeitdaten im benötigten Format zur Verfügung gestellt werden. Diese Schritte bilden die Grundlage für die folgende Initialisierung und Ausführung der Simulation.

Im nächsten Schritt erfolgt die Definition der stochastischen und strukturellen Einflussgrößen durch den Nutzer, die das Verhalten des Simulationsmodells maßgeblich prägen. Zunächst werden die Abweichungsbereiche für Transport- und Bearbeitungszeiten festgelegt. Diese Parameter definieren die Grenzen, innerhalb derer im weiteren Verlauf die zufallsbedingten Schwankungen für jede Transporteinheit individuell generiert werden. Im Anschluss wird abgefragt, ob an ausgewählten Knotenpunkten eine gezielte Bündelung von Lieferfahrzeugen durch eine definierte Pufferzeit erfolgen soll und welcher Zeit in Perioden diese Wartezeit entsprechen soll. Abschließend wird durch eine weitere Angabe festgelegt, ob zur Entlastung der Ressourcen zusätzliche Ladetore dynamisch geöffnet werden dürfen. In diesem Fall ist anschließend die maximale Anzahl an zusätzlichen Ladetüren anzugeben. Wird diese Option abgelehnt, verbleibt das Modell bei den standardmäßig definierten Ressourcenkapazitäten. Mit Abschluss dieser Eingabe sind alle erforderlichen Parameter gegeben, sodass die Simulation initialisiert und durchgeführt werden kann.

Abschließend erfolgt die Durchführung der Simulation in zwei voneinander getrennten Durchläufen durch Aufruf der TruckSimulation-Klasse. Im ersten Schritt wird das Standardmodell ausgeführt, bei dem alle Fahrzeugprozesse unter den definierten Eingabeparametern simuliert werden, ohne dass zusätzliche Entscheidungsmaßnahmen zur Optimierung zum Einsatz kommen. Der Konstruktor der Klasse erhält sämtliche erforderlichen Informationen, wobei der Parameter für den optimierten Modus explizit auf „False“ gesetzt wird, um die deterministische Basisvariante zu wählen. Anschließend erfolgt die Ausführung der Simulation durch den Aufruf der ersten Iteration.

Im zweiten Simulationsdurchlauf wird das optimierte Modell initialisiert, welches die erweiterten Entscheidungsregeln implementiert. Der Parameter des optimierten Modus wird in diesem Fall auf „True“ gesetzt, wodurch die erweiterten Steuerungslogiken aktiviert werden. Um eine Vergleichbarkeit mit dem Standardmodell zu gewährleisten, werden in dieser Variante dieselben zufälligen Abweichungen verwendet wie im ersten Durchlauf. Die Ausführung des Simulationsmodells erfolgt im Rahmen der zweiten Iteration. Durch dieses Vorgehen erzeugen beide Ausführungen jeweils eigenständige Ergebnisdateien, die als Grundlage zur anschließenden Bewertung dienen.

5 Ergebnisse

Nach Durchführung der einzelnen Simulationsläufe erfolgt im nachfolgenden Kapitel die Darstellung und Analyse der erzielten Ergebnisse. Sämtliche Auswertungen basieren auf dem stochastischen Simulationsmodell, in dem für alle Durchführungen ein einheitlicher Schwankungsbereich zwischen -10 Prozent und +20 Prozent sowohl für die Transport- als auch für die Bearbeitungszeiten hinterlegt wurde. Auf diese Weise sollen einerseits eine realitätsnahe Abbildung der Unsicherheiten und andererseits eine Vergleichbarkeit der einzelnen Modellvarianten gewährleistet werden.

Zur Erhöhung der Übersichtlichkeit wird zunächst in Abschnitt 5.1 ein Basisszenario betrachtet, welches auf dem vereinfachten Modell mit elf Knotenpunkten und einer Be- und Entladezeit von 30 Minuten basiert. In diesem Rahmen erfolgt die Gegenüberstellung der Ergebnisse des Standardmodells ohne Optimierungsmaßnahmen mit der optimierten Modellvariante, in der einzelne Entscheidungsregeln gezielt aktiviert werden.

Ergänzend enthält Abschnitt 5.2 eine Zusammenfassung sämtlicher weiterer Konfigurationen. Hierbei wurden unterschiedliche Netzausprägungen unter Anwendung der verschiedenen Regelkombinationen ausgewertet, um die Robustheit sowie die Übertragbarkeit der Ergebnisse auf alternative Anwendungsszenarien bewerten zu können.

5.1 Darstellung der Simulationsergebnisse

Um eine Vergleichsbasis zur Wirkung der eingesetzten Entscheidungsregeln zu schaffen, wurde im ersten Schritt ein Simulationsdurchlauf nahezu ohne Maßnahmen zur möglichen Optimierung durchgeführt. In dieser Konfiguration wurden sämtliche Entscheidungslogiken deaktiviert, mit Ausnahme der Priorisierung von Lieferwagen mit besonders knappem Zeitfenster. Abgesehen von dieser Regel ist der weitere Ablauf der Simulation identisch. Ein zweiter wesentlicher Unterschied zwischen den beiden Modellvarianten besteht in der Einbeziehung der festgelegten deterministischen Wartezeiten, welche ausschließlich in der Standardausführung der Simulation berücksichtigt werden. Diese werden dort entsprechend der ursprünglichen Planung unter deterministischen Werten implementiert, während sie im optimierten Modell entfallen. Als Folge der stochastischen Einflussgrößen resultieren in der optimierten Variante

jedoch eigenständige Wartezeiten. Ein direkter Vergleich der beiden Varianten ist in Tabelle 1 dargestellt.

Tabelle 1: Vergleich der Leistungskennzahlen ohne Entscheidungsregeln

Kennzahl	Standard-modell	Optimiertes Modell	Veränderung in %
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	1,40	1,40	0,00
Durchschnittliche Anzahl wartender LKW (Startknoten)	22,20	22,20	0,00
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	2,20	2,00	-9,09
Durchschnittliche Anzahl wartender LKW (Zielknoten)	49,70	21,20	-57,34
Durchschnittliche Wartezeit in Perioden	0,47	0,13	-72,53
Durchschnittliche Anzahl verspäteter Waren	28,70	23,20	-19,16

Quelle: Eigene Darstellung

Wie ersichtlich, stimmen die Auswertungen an den Startknoten exakt überein. Dies ist darauf zurückzuführen, dass in beiden Modellen identische Startzeitpunkte der Lieferfahrzeuge verwendet werden und an dieser Stelle noch keine Entscheidungsregeln greifen. Am Zielknoten zeigen sich hingegen deutliche Unterschiede. Die durchschnittliche Anzahl wartender Lieferwagen konnte im optimierten Modell um rund 57 Prozent reduziert werden. Ebenso ist die durchschnittliche Wartezeit um etwa 72,5 Prozent beträchtlich geringer. Besonders hervorzuheben ist die durchschnittliche Anzahl verspäteter Warenlieferungen, die in der optimierten Variante, trotz deaktivierter Entscheidungsregeln, um 19,16 Prozent reduziert werden konnte. Diese Kennzahl stellt die zentrale Zielgröße der Auswertung dar. Die durchschnittliche Verspätungsdauer pro Warenstrom blieb dabei in beiden Modellvarianten identisch und lag bei 0,64 Perioden. Sie wurde zusätzlich ausgewertet, um mögliche Unterschiede im Ausmaß der Verspätungen zu ermitteln.

Im nächsten Schritt wurde die Entscheidungsregel aktiviert, welche gezielt eine Pufferzeit einführt, um an bestimmten Knotenpunkten eine Bündelung der ankommenden Lieferfahrzeuge zu ermöglichen. Die Simulation wurde dazu in mehreren Durchläufen

mit unterschiedlich langen Pufferzeiten getestet. Die besten Ergebnisse hinsichtlich der Anzahl verspäteter Waren konnten bei einer Verzögerung von 0,2 Perioden erzielt werden. Während geringere Verzögerungen ebenfalls zu leichten Verbesserungen führten, verschlechterten sich die Ergebnisse ab einer Pufferzeit von 0,5 Perioden im Vergleich zum Standardmodell. Ein Überblick für diese Konfiguration ist in Tabelle 2 dargestellt.

Tabelle 2: Vergleich mit eingesetzter Pufferzeit von 0,2 Perioden

Kennzahl	Standard-modell	Optimiertes Modell	Veränderung in %
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	1,30	1,30	0,00
Durchschnittliche Anzahl wartender LKW (Startknoten)	20,40	45,90	125,00
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	2,30	2,10	-8,70
Durchschnittliche Anzahl wartender LKW (Zielknoten)	49,70	51,80	4,23
Durchschnittliche Wartezeit in Perioden	0,48	0,24	-49,89
Durchschnittliche Anzahl verspäteter Waren	30,20	23,40	-22,52

Quelle: Eigene Darstellung

Besonders auffällig ist der deutliche Anstieg der durchschnittlichen Anzahl wartender Lieferfahrzeuge, insbesondere am Startknotenpunkt, wo eine Zunahme von 125 Prozent zu verzeichnen ist. Diese Erhöhung ist jedoch eine direkte Folge der implementierten Entscheidungsregel, bei der Fahrzeuge bewusst für eine definierte Zeitspanne angehalten werden. Dieser Effekt wirkt sich allerdings nicht in Form einer längeren Gesamtwartezeit aus, da diese aufgrund der nur kurzfristigen Verzögerungen im Durchschnitt sogar um rund 50 Prozent zurückgegangen ist. Die durchschnittliche Anzahl verspätet eintreffender Warenlieferungen sank aufgrund der im optimierten Modell integrierten Pufferzeit im Vergleich zum Standardmodell um 22,52 Prozent. Die durchschnittliche Dauer der Verspätung blieb hingegen unverändert. Alle übrigen Kennzahlen weisen nur moderate Veränderungen auf.

Im nächsten Szenario wurde die zuvor eingesetzte Pufferzeitregel deaktiviert und stattdessen eine alternative Entscheidungsregel aktiviert, durch die zusätzliche Ladetüren

an den Start- und Zielknoten geöffnet werden können. Konkret erlaubt diese Regel, dass maximal ein weiteres Ladetor geöffnet werden darf, sobald sich mehr als zwei Fahrzeuge in der Warteschlange befinden. Die resultierenden Ergebnisse dieser Regel sind in Tabelle 3 abgebildet.

Tabelle 3: Vergleich mit einer zusätzlichen Ladetür

Kennzahl	Standard-modell	Optimiertes Modell	Veränderung in %
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	1,30	1,00	-23,08
Durchschnittliche Anzahl wartender LKW (Startknoten)	21,00	19,40	-7,62
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	2,60	1,60	-38,46
Durchschnittliche Anzahl wartender LKW (Zielknoten)	50,00	18,20	-63,60
Durchschnittliche Wartezeit in Perioden	0,48	0,10	-79,87
Durchschnittliche Anzahl verspäteter Waren	31,20	23,80	-23,72

Quelle: Eigene Darstellung

Insgesamt nutzten durchschnittlich 7,8 Lieferfahrzeuge die zusätzlich geöffnete Ladetür. Durch diese Konfiguration konnten sämtliche untersuchten Kennzahlen durch das optimierte Modell erheblich verbessert werden. Die durchschnittliche maximale Warteschlangenlänge verringerte sich sowohl an den Start- als auch an den Zielknoten beträchtlich. Am Zielknoten konnte diese sogar um mehr als 38 Prozent reduziert werden. Besonders hervorzuheben ist zudem die Anzahl wartender Lieferwagen an den Zielknoten, die im optimierten Modell im Vergleich zur Standardausführung um durchschnittlich 63,6 Prozent gesenkt werden konnte. Auch die gesamte Wartezeit wurde durch diese Maßnahme erheblich beeinflusst und verzeichnete einen Rückgang von nahezu 80 Prozent. Darüber hinaus konnte die durchschnittliche Anzahl verspätet zugestellter Waren um 23,72 Prozent reduziert werden. Ergänzend verringerte sich die durchschnittliche Verspätung je Lieferung um knapp 7 Prozent.

Im nächsten Schritt wurde die zuvor eingesetzte Entscheidungsregel der dynamischen Kapazitätsanpassung erweitert. Die maximale Anzahl zusätzlicher Ladetüren wurde in

dieser Konfiguration von einer auf zwei erhöht, um verbleibende Engpässe weiter zu reduzieren. Die resultierenden Ergebnisse befinden sich in Tabelle 4.

Tabelle 4: Vergleich mit zwei zusätzlichen Ladetüren

Kennzahl	Standard-modell	Optimiertes Modell	Veränderung in %
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	1,50	1,00	-33,33
Durchschnittliche Anzahl wartender LKW (Startknoten)	21,30	18,90	-11,27
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	2,60	1,10	-57,69
Durchschnittliche Anzahl wartender LKW (Zielknoten)	51,00	18,10	-64,51
Durchschnittliche Wartezeit in Perioden	0,47	0,09	-81,82
Durchschnittliche Anzahl verspäteter Waren	29,70	22,00	-25,93

Quelle: Eigene Darstellung

Die zusätzlichen Ladetore wurden im Durchschnitt von 8,2 Transportern genutzt. Es zeigen sich im Vergleich zur Variante mit nur einer zusätzlichen Ladetür in jedem Bereich weitere, wenn auch teils nur geringe Verbesserungen. Die durchschnittliche maximale Warteschlangenlänge stellt in dieser Auswertung den auffälligsten Kennwert dar. Sowohl am Start- als auch am Zielknoten liegt diese bei 1,0 beziehungsweise 1,1, was einer Reduktion von etwa 33 Prozent beziehungsweise rund 58 Prozent entspricht. Auch die durchschnittliche Anzahl verspäteter Waren konnte mit dieser Konfiguration verringert werden und liegt nun bei 25,93 Prozent. Die durchschnittliche Verzögerung pro Ware wurde ebenfalls um 13,74 Prozent verringert.

Da beide zuvor eingesetzten Entscheidungsregeln einzeln zu signifikanten Verbesserungen geführt haben, wurde im nächsten Schritt der kombinierte Einsatz dieser Logiken getestet. Die Anzahl zusätzlicher Ladetüren wurde für diese Konfiguration auf eine beschränkt, da zuvor ersichtlich war, dass das Öffnen von mehr als einem zusätzlichen Ladetor lediglich zu geringfügigen Verbesserungen führt und nur wenige weitere Lieferwagen die zusätzliche Kapazität tatsächlich nutzen. Gleichzeitig führt jede zusätzliche Ressource zu potenziellen Mehrkosten.

Die Pufferzeit zum gezielten Bündeln von Fahrzeugen wurde erneut auf 0,2 Perioden festgelegt, da diese Konfiguration erneut die besten Ergebnisse lieferte. Die Resultate der kombinierten Variante sind in Tabelle 5 einzusehen.

Tabelle 5: Vergleich bei kombiniertem Einsatz der Entscheidungsregeln

Kennzahl	Standard-modell	Optimiertes Modell	Veränderung in %
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	1,40	1,00	-28,57
Durchschnittliche Anzahl wartender LKW (Startknoten)	20,00	44,30	121,50
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	2,60	2,30	-11,54
Durchschnittliche Anzahl wartender LKW (Zielknoten)	51,00	47,30	-7,25
Durchschnittliche Wartezeit in Perioden	0,49	0,20	-59,88
Durchschnittliche Anzahl verspäteter Waren	29,80	22,50	-24,50

Quelle: Eigene Darstellung

In dieser Konfiguration wurde die zusätzlich geöffnete Ladetür von durchschnittlich 5,6 Lieferwagen genutzt. Besonders auffällig ist die starke Erhöhung der durchschnittlichen Anzahl wartender Lieferwagen am Startknoten um 121,5 Prozent. Allerdings ist dieser Anstieg erneut auf die gezielte Anstauung der Transporter zurückzuführen. Trotz der erhöhten Anzahl wartender Fahrzeuge konnte die gesamte Wartezeit um durchschnittlich rund 60 Prozent reduziert werden, was klar für die Effizienz der Konfiguration spricht. Auch die übrigen Kennzahlen weisen Verbesserungen gegenüber dem Standardmodell auf. So verringerte sich unter anderem die durchschnittliche Warteschlangenlänge und die Anzahl verspätet eintreffender Waren konnte auf diese Weise um 24,5 Prozent gesenkt werden. Die durchschnittliche Verspätung pro Ware ging hingegen nur um knapp 2 Prozent zurück.

5.2 Auswertung der Simulationsergebnisse

Im folgenden Abschnitt steht der direkte Vergleich der verschiedenen Modellkonfigurationen im Mittelpunkt. Zu diesem Zweck wurden mehrere Varianten des Simulationsmodells unter variierenden Parametern ausgeführt und deren Ergebnisse gegenübergestellt. Untersucht wurden unter anderem zwei unterschiedlich große Service-Netzwerke: das Basismodell mit elf Knotenpunkten sowie eine erweiterte Variante mit zwanzig Knotenpunkten. Beide Netzwerke wurden jeweils mit einer Bearbeitungszeit von 30 Minuten sowie mit einer erhöhten Bearbeitungszeit von 120 Minuten simuliert, um die Auswirkungen unterschiedlicher Belastungen auf die Modellsysteme zu analysieren.

Tabelle 6 zeigt noch einmal die Gegenüberstellung der in Kapitel 5.1 dargestellten Ergebnisse des Basismodells, um einen direkten Vergleich der verschiedenen Entscheidungslogiken zu ermöglichen.

Tabelle 6: Vergleich im Basismodell unter 30 Minuten Bearbeitungszeit

Kennzahlen in %	PZ = 0 ZT = 0	PZ = 0,2 ZT = 0	PZ = 0 ZT = 1	PZ = 0 ZT = 2	PZ = 0,2 ZT = 1
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	0,00	0,00	-23,08	-33,33	-28,57
Durchschnittliche Anzahl wartender LKW (Startknoten)	0,00	125,00	-7,62	-11,27	121,50
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	-9,09	-8,70	-38,46	-57,69	-11,54
Durchschnittliche Anzahl wartender LKW (Zielknoten)	-57,34	4,23	-63,60	-64,51	-7,25
Durchschnittliche Wartezeit in Perioden	-72,53	-49,89	-79,87	-81,82	-59,88
Durchschnittliche Anzahl verspäteter Waren	-19,16	-22,52	-23,72	-25,93	-24,50

Quelle: Eigene Darstellung

Im direkten Vergleich fällt auf, dass insbesondere die Modellkonfigurationen, in denen zusätzliche Ladetüren geöffnet werden können, in allen betrachteten Kennzahlen die besten Resultate erzielen. Sowohl die durchschnittliche maximale Warteschlangenlänge an den Start- und Zielknoten als auch die Anzahl wartender Lieferwagen gehen in diesen

Varianten erheblich zurück. Auch die durchschnittliche Wartezeit reduziert sich beträchtlich. Besonders hervorzuheben ist jedoch die Entwicklung der durchschnittlichen Anzahl verspäteter Waren, welche die zentrale Zielgröße in der Auswertung darstellt. Diese konnte in den Varianten mit optionaler Türerweiterung am stärksten reduziert werden. In den Simulationsdurchläufen, in denen ausschließlich eine Pufferzeit zum gezielten Anstauen eingesetzt wurde, zeigt sich erwartungsgemäß ein Anstieg der wartenden Fahrzeuge sowie der Wartezeiten. Da diese Methode explizit vorsieht, Fahrzeuge kurzzeitig zu verzögern, verschlechtern sich jene Kennzahlen, in denen die Wartephase berücksichtigt wird, im Vergleich zu Konfigurationen ohne Verzögerung. Dennoch konnte auch hier eine Verbesserung hinsichtlich der Pünktlichkeit der Warenlieferungen erzielt werden. Besonders interessant ist die kombinierte Variante, in der alle Entscheidungsregeln gleichzeitig angewendet werden. Durch diese Konfiguration konnte die durchschnittliche Anzahl verspäteter Waren noch weiter gesenkt werden als in der Variante, in der lediglich eine zusätzliche Tür geöffnet werden konnte. Dies lässt darauf schließen, dass durch die kombinierte Anwendung aller Maßnahmen, trotz kurzfristiger Verzögerungen und einer Verschlechterung einzelner Wartezeitkennzahlen, insgesamt die besten Ergebnisse im Hinblick auf die Zielgröße Pünktlichkeit erzielt werden können.

Tabelle 7 zeigt die prozentuale Veränderung der Kennzahlen im Basismodell mit elf Knotenpunkten bei einer erhöhten Lade- und Entladezeit von 120 Minuten. Dargestellt ist, in welchem Ausmaß sich die Ergebnisse der jeweiligen optimierten Modellvariante im Vergleich zur Standardvariante verändert haben. In der Simulationskonfiguration, in der ausschließlich eine Pufferzeit eingesetzt und keine zusätzliche Ladetür geöffnet wurde, stellte sich erneut ein Pufferzeitwert von 0,2 Perioden als effektivste Variante heraus. Wird ergänzend zum Einsatz der Pufferzeit das Öffnen einer weiteren Tür erlaubt, konnten die besten Ergebnisse mit einer eingesetzten Wartezeit von 0,3 Perioden erzielt werden.

Tabelle 7: Vergleich im Basismodell unter 120 Minuten Bearbeitungszeit

Kennzahlen in %	PZ = 0 ZT = 0	PZ = 0,2 ZT = 0	PZ = 0 ZT = 1	PZ = 0 ZT = 2	PZ = 0,3 ZT = 1
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	0,00	11,11	-10,53	-16,67	5,26
Durchschnittliche Anzahl wartender LKW (Startknoten)	0,00	55,32	-11,30	-43,75	32,82
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	-14,29	-5,00	-31,82	-47,62	-10,00
Durchschnittliche Anzahl wartender LKW (Zielknoten)	-45,80	-28,66	-49,59	-52,01	-32,72
Durchschnittliche Wartezeit in Perioden	-38,04	-35,26	-45,38	-51,90	-36,53
Durchschnittliche Anzahl verspäteter Waren	-9,24	-10,74	-13,64	-16,94	-14,29

Quelle: Eigene Darstellung

Wie bereits in Tabelle 6 ersichtlich, zeigen auch hier besonders die Modellkonfigurationen mit erweiterten Ressourcenkapazitäten deutliche Verbesserungen über alle Kennzahlen hinweg. Durch den Einsatz der zusätzlichen Bearbeitungsmöglichkeiten konnten sowohl die maximale Warteschlangenlänge als auch die Anzahl wartender Lieferwagen erheblich gesenkt werden. Auch die durchschnittliche Wartezeit wurde in dieser Variante beträchtlich reduziert. Demgegenüber ist erneut erkennbar, dass der Einsatz einer Pufferzeit zwar zu einem Anstieg der Wartezeitkennzahlen führen kann, gleichzeitig jedoch zu einer Verbesserung der Liefertreue beiträgt. So konnte die durchschnittliche Anzahl verspätet zugestellter Waren durch den Einsatz der Pufferzeit in sämtlichen Varianten reduziert werden. Besonders hervorzuheben ist die Konfiguration, in der alle Entscheidungsregeln gleichzeitig zum Einsatz kommen. Auch unter erhöhten Bearbeitungszeiten zeigt sich, dass die Kombination der Maßnahmen zu insgesamt besseren Ergebnissen in Bezug auf die Pünktlichkeit führt als durch die separate Nutzung der Optimierungsregeln.

Im nächsten Schritt wurde das erweiterte Modell mit insgesamt 20 Knotenpunkten analysiert. Die Simulation erfolgte zunächst mit einer Bearbeitungszeit von 30 Minuten. Für diese Konfiguration stellte sich sowohl im Szenario ohne zusätzliche Ladetüren als auch in der kombinierten Variante eine Pufferzeit von 0,1 Perioden als am effizientesten

heraus, um die besten Ergebnisse im Hinblick auf die Pünktlichkeit der Warenlieferungen zu erzielen. Tabelle 8 zeigt die prozentualen Veränderungen dieses Modells im Vergleich zur Standardvariante.

Tabelle 8: Vergleich im erweiterten Modell unter 30 Minuten Bearbeitungszeit

Kennzahlen in %	PZ = 0 ZT = 0	PZ = 0,1 ZT = 0	PZ = 0 ZT = 1	PZ = 0 ZT = 2	PZ = 0,1 ZT = 1
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	0,00	0,00	0,00	0,00	0,00
Durchschnittliche Anzahl wartender LKW (Startknoten)	0,00	98,25	0,00	0,00	98,72
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	-20,00	-12,50	-17,39	-56,00	-16,00
Durchschnittliche Anzahl wartender LKW (Zielknoten)	-47,19	50,39	-56,10	-55,65	42,42
Durchschnittliche Wartezeit in Perioden	-68,59	-57,23	-75,38	-79,05	-64,88
Durchschnittliche Anzahl verspäteter Waren	-12,62	-14,89	-15,49	-17,71	-16,13

Quelle: Eigene Darstellung

Auffällig ist, dass durch keine der getesteten Modellvarianten eine Verbesserung der durchschnittlichen maximalen Warteschlangenlänge am Startknoten erreicht werden konnte. Alle Konfigurationen weisen hier einen Veränderungswert von null Prozent auf. Ein ähnliches Bild zeigt sich bei der durchschnittlichen Anzahl wartender Lieferwagen am Startknoten. Verbesserungen durch den Einsatz zusätzlicher Ladetüren sind nicht festzustellen. Es kommt lediglich durch die gezielte Verzögerung der Transporter erwartungsgemäß zu einer Erhöhung dieses Werts. Hervorzuheben ist zudem, dass die Modellvariante, bei der die Öffnung von bis zu zwei weiteren Ladetüren ermöglicht wird, einen besonders positiven Effekt auf die maximale Warteschlangenlänge am Zielknoten zeigt. Die übrigen Ergebnisse zeigen ein weitgehend vergleichbares Muster zu den vorherigen Auswertungen. Im Hinblick auf die Zielgröße der durchschnittlichen Anzahl verspäteter Waren zeigen sich ebenfalls konsistente und vergleichbare Verbesserungen wie zuvor. Alle Varianten führen zu einer Reduktion der verspäteten Lieferungen, wobei

erneut die Modellkonfiguration mit zwei zusätzlichen Türen sowie die kombinierte Variante besonders hervorzuheben sind.

Im letzten Schritt wurde das erweiterte Modell mit einer Bearbeitungszeit von 120 Minuten unter Anwendung der verschiedenen Modellkonfigurationen simuliert. Diese Kombination bildet das umfangreichste und gleichzeitig zeitintensivste Szenario der gesamten Untersuchung ab. Für jede Variante, in der eine Pufferzeit zum Einsatz kam, erwies sich ein Wert von 0,1 Perioden erneut als am effektivsten. Der Vergleich der Ergebnisse mit dem jeweiligen Standardmodell ist in Tabelle 9 dargestellt.

Tabelle 9: Vergleich im erweiterten Modell unter 120 Minuten Bearbeitungszeit

Kennzahlen in %	PZ = 0 ZT = 0	PZ = 0,1 ZT = 0	PZ = 0 ZT = 1	PZ = 0 ZT = 2	PZ = 0,1 ZT = 1
Durchschnittliche maximale Warteschlangenlänge (Startknoten)	0,00	100,00	0,00	0,00	100,00
Durchschnittliche Anzahl wartender LKW (Startknoten)	0,00	79,02	-4,12	-16,89	64,57
Durchschnittliche maximale Warteschlangenlänge (Zielknoten)	0,00	0,00	-41,18	-44,44	0,00
Durchschnittliche Anzahl wartender LKW (Zielknoten)	-37,27	16,62	-41,25	-41,07	12,70
Durchschnittliche Wartezeit in Perioden	-32,14	-31,64	-35,99	-36,79	-35,79
Durchschnittliche Anzahl verspäteter Waren	-11,08	-12,46	-11,45	-11,75	-13,33

Quelle: Eigene Darstellung

Bemerkenswert sind die Entwicklungen in Bezug auf die durchschnittliche maximale Warteschlangenlänge. Am Startknoten konnte durch keine der getesteten Modellkonfigurationen eine Verbesserung erzielt werden, in Varianten mit eingesetzter Pufferzeit wurde die Warteschlangenlänge sogar verdoppelt. Im Gegensatz dazu haben sich die Werte am Zielknoten in keinem Szenario verschlechtert, sondern zeigen deutliche Verbesserungen durch den Einsatz zusätzlicher Ladetüren von bis zu 44,44 Prozent. Die übrigen Ergebnisse zeigen erneut ein ähnliches Muster wie in den vorangegangenen Auswertungen. Auffällig ist jedoch, dass in diesem Fall nicht die Konfiguration mit zusätzlichen Ladetüren die besten Ergebnisse im Hinblick auf die Zielgröße erzielt,

sondern die Simulationsvariante mit eingesetzter Pufferzeit. Eine Pufferzeit von 0,1 Perioden führte in diesem Modell zu einer stärkeren Reduktion verspätet eintreffender Waren als die Varianten mit zusätzlichen Ladetüren. Die Kombination beider Maßnahmen erzielte jedoch die besten Ergebnisse und bewirkte eine durchschnittliche Senkung um 13,33 Prozent. Damit übertraf sie die Konfiguration mit zwei zusätzlichen Ladetüren, die in allen vorherigen Fällen das beste Ergebnis lieferte und in diesem Szenario vergleichsweise eine Reduktion um 11,75 Prozent erreichte.

6 Analyse

Im folgenden Kapitel werden die Ergebnisse der durchgeführten Simulation ausgewertet und interpretiert. Ziel ist es, die Auswirkungen stochastischer Einflussfaktoren sowie der eingesetzten Entscheidungsregeln auf das Verhalten der logistischen Netzwerke zu beurteilen und daraus Optimierungspotenziale abzuleiten. Zunächst erfolgt ein vergleichender Überblick über die verschiedenen Modellvarianten und deren Auswirkung auf die zentralen Bewertungskennzahlen. Im Anschluss werden bestehende Engpässe und mögliche Erweiterungen des Simulationsmodells kritisch betrachtet. Die daraus gewonnenen Erkenntnisse dienen abschließend als Grundlage für die Bewertung des Modells und liefern Ansatzpunkte für zukünftige Weiterentwicklungen.

6.1 Interpretation der Ergebnisse

Im Gesamtvergleich aller durchgeführten Simulationsläufe zeigt sich, dass jede untersuchte Modellkonfiguration zu einer Verbesserung gegenüber dem Standardmodell geführt hat. Bereits im Vergleich zwischen dem Standardmodell, welches auf deterministischen Daten basiert, und der optimierten Variante ohne zusätzliche Entscheidungsregeln wurden bessere Ergebnisse erzielt. Diese Verbesserung resultiert aus dem Wegfall der deterministischen Wartezeiten, die vorab im Standardmodell aufgrund des starren Prozessablaufs ohne stochastische Einflussfaktoren festgelegt wurden und integriert werden mussten. Im optimierten Modell entfallen diese festen Wartezeiten, wodurch die Lieferprozesse flexibler aufeinander reagieren können. Dies führt aufgrund der zufallsbasierten Verzögerungen bereits ohne weitere Entscheidungsregeln zu neuen und geringeren auftretenden Wartezeiten sowie zu einer reduzierten Anzahl verspäteter

Warenlieferungen. In dieser Variante konnten am Startknoten allerdings keine Verbesserungen festgestellt werden, unabhängig von der Modellgröße. Dies ist darauf zurückzuführen, dass in beiden Modellvarianten identische Startzeitpunkte verwendet werden und der Ablauf am Startknoten ohne zusätzliche Entscheidungsregeln unverändert bleibt.

Die Einführung der Entscheidungsregel zur Öffnung zusätzlicher Ladetüren hatte signifikante Verbesserungen auf alle Kennzahlbereiche zur Folge. Durch die dynamische Erweiterung der Ressourcenkapazitäten konnten sowohl die maximale Warteschlangenlänge als auch die durchschnittliche Anzahl wartender Fahrzeuge deutlich reduziert werden. Auch die durchschnittliche Wartezeit und insbesondere die Anzahl verspäteter Waren verbesserten sich erheblich, im Vergleich zur Standardvariante. Die Erweiterung der maximal zulässigen zusätzlichen Ladetore auf mehr als zwei führte zu weiteren Verbesserungen, jedoch nur in einem geringfügigen Ausmaß. Da der Ausbau der Ressourcenkapazität mit potenziell steigenden Kosten verbunden ist, sind die Konfigurationen mit zusätzlichen Ladetüren als kostenintensiv zu bewerten, auch wenn sie durchweg bessere Ergebnisse liefern. Zudem zeigte sich, dass die zusätzlich bereitgestellten Kapazitäten lediglich von einer begrenzten Anzahl weiterer Lieferfahrzeuge genutzt wurden. Daher ist ein zurückhaltender und bedarfsgerechter Einsatz dieser Maßnahme geboten.

Auch der Einsatz einer Pufferzeit zum gezielten Anstauen und Priorisieren der Lieferwagen ermöglichte Verbesserungen hinsichtlich der Zielgröße Pünktlichkeit im Vergleich zum Standardmodell. Trotz einer erheblichen Erhöhung der maximalen Warteschlangenlänge und der Anzahl wartender Lieferfahrzeuge infolge der bewusst integrierten Verzögerungen führte diese Strategie zu einer deutlichen Reduktion der durchschnittlichen Gesamtwartezeit und insbesondere der verspätet eintreffenden Waren. Die negative Entwicklung der Kennzahlen, die Wartezeiten berücksichtigen, war allerdings zu erwarten, da die Transporter bewusst für eine kurze Zeit gestoppt werden und diese temporäre Dauer in die Auswertung einbezogen wird. Die Effektivität der Pufferzeitregel hängt jedoch stark von einer sorgfältigen Kalibrierung der Verzögerungsdauer ab, da nicht alle getesteten Pufferzeiten gleich gute Ergebnisse lieferten. In einzelnen Fällen führten zu hohe Zeitspannen der Verzögerungen sogar zu Verschlechterungen gegenüber dem Standardmodell. Darüber hinaus zeigte sich, dass die optimale Pufferzeit stark von der Modellgröße sowie der Dauer der Be- und Entladezeit abhängig ist und somit nicht universell einsetzbar ist. Sobald diese Konfiguration

allerdings erfolgreich implementiert ist, lassen sich deutliche Reduktionen in Bezug auf die verspätet eintreffenden Warenlieferungen erzielen. Insbesondere in größeren Modellen mit längeren Bearbeitungszeiten zeigt sich diese Optimierungsregel als besonders wirkungsvoll.

Da beide Entscheidungsregeln einzeln gesehen deutliche Verbesserungen bewirkten, wurde im weiteren Verlauf eine kombinierte Variante getestet, in der sowohl eine Pufferzeit eingesetzt wurde als auch das Öffnen zusätzlicher Ladetüren erlaubt war. Die Anzahl zusätzlicher Ladetore wurde dabei bewusst auf eins beschränkt, um potenzielle Mehrkosten zu begrenzen. Die Ergebnisse dieser Konfiguration zeigten erneut die für die Pufferzeit typische Erhöhung wartender Fahrzeuge an den Knotenpunkten. Die negativen Effekte wurden jedoch durch die zusätzliche Ressourcenkapazität bemerkbar geschwächt. Gleichzeitig konnten die durchschnittliche Wartezeit sowie die Anzahl verspäteter Waren erheblich durch den kombinierten Einsatz der Entscheidungsregeln reduziert werden. In einzelnen Szenarien zeigte diese Modellvariante sogar die besten Ergebnisse gegenüber allen anderen Varianten. Daraus lässt sich ableiten, dass die Synergie aller Regeln zu einer besonders hohen Liefertreue führt. Zudem stellt diese Variante im Sinne des Kosten-Nutzen-Verhältnisses eine besonders vorteilhafte Strategie dar, da mit minimalem Ressourcenausbau ein deutlicher Gewinn der Gesamtperformance erzielt werden konnte.

Ergänzend ist festzuhalten, dass sowohl die Größe des Modells als auch die Dauer der Ladezeiten Einfluss auf das Ausmaß der Verbesserungen haben. Besonders bei hohen Bearbeitungszeiten ist der Spielraum für Verzögerungen geringer, was zur Folge hat, dass die Wirkung einzelner Entscheidungsregeln abgeschwächt wird. Dennoch haben diese Rahmenbedingungen keinen Einfluss auf die grundsätzliche Wirksamkeit und Funktion der implementierten Optimierungsregeln. In allen Modellkonfigurationen konnten bemerkenswerte Verbesserungen der Liefertreue erzielt werden, wodurch die wirksame Funktionsweise der eingesetzten Optimierungsmechanismen bestätigt werden kann.

6.2 Identifikation von Engpässen und Optimierungspotenzialen

Ein zentraler Engpass innerhalb der simulierten Modelle zeigt sich an den Startknoten. Anhand der Simulationsergebnisse wird verdeutlicht, dass die dort auftretenden Warteschlangenlängen und Wartezeiten kaum durch die eingesetzten Optimierungsmaßnahmen beeinflusst werden können. Auffällig ist, dass durch den

Einsatz von Pufferzeiten an dieser Stelle sogar Verschlechterungen auftreten. Dies ist darauf zurückzuführen, dass das Modell in allen Varianten die gleichen Startzeitpunkte nutzt und die Lieferfahrzeuge insbesondere bei einer Pufferung gezielt zum Warten gestoppt werden. In erweiterten Modellen mit umfassenderen Netzwerken war dieser Effekt besonders stark ausgeprägt. Selbst durch den Einsatz zusätzlicher Ladetüren konnte an den Startknoten keine beträchtliche Verbesserung erzielt werden. Es ist jedoch anzumerken, dass der Ausgangswert der maximalen Warteschlangenlänge an dieser Stelle vergleichsweise niedrig ist, wodurch sich im Gegensatz zum Zielknoten ein geringeres Potenzial für Verbesserungen ergibt.

Daraus lässt sich ableiten, dass die Zielknoten als zentrale Stellgröße für die Optimierung im Modell dienen. In nahezu allen Konfigurationen weisen sie eine signifikant höhere Warteschlangenlänge und Anzahl wartender Lieferwagen auf als die Startknoten. Gleichzeitig zeigen die Ergebnisse, dass insbesondere der Einsatz zusätzlicher Ladetüren am Zielknoten zu beträchtlichen Verbesserungen in allen untersuchten Kennzahlen führte. Dies führt zu dem Schluss, dass an diesem Punkt das größte Optimierungspotenzial besteht.

Ein weiteres Optimierungspotenzial ergibt sich durch den gezielten Einsatz von Pufferzeiten. Die Entscheidungsregel verbessert nachweislich die Pünktlichkeit der Warenlieferungen, führt jedoch zeitgleich zu einer kurzfristigen Verschlechterung der Kennzahlen bezüglich der Wartephasen. Die Wirksamkeit der Regel ist zudem stark abhängig von der Modellgröße sowie der zugrunde liegenden Be- und Entladezeiten. Eine universelle Festlegung ist daher nicht empfehlenswert. Für jeden Anwendungsfall sollte die optimale Pufferzeit individuell kalibriert werden, da überhöhte Werte zu einer Überlastung der Knotenpunkte führen können und infolgedessen eine Verschlechterung der Ergebnisse in Bezug auf die Zielgröße verursachen.

Auch die aktuell implementierte Logik zur Erweiterung der Ladetore bietet Verbesserungspotenzial. Derzeit bleiben geöffnete Ladetüren während des gesamten Simulationsdurchlaufs aktiv, unabhängig von ihrer tatsächlichen Auslastung. In einer Weiterentwicklung könnte eine dynamische und bedarfsorientierte Logik implementiert werden, bei der zusätzliche Tore nur so lange geöffnet bleiben, wie sie tatsächlich benötigt werden. Durch eine derartige Steuerung ließe sich nicht nur die Anzahl gleichzeitig geöffneter Ressourcen minimieren, sondern auch ein gezieltes und dynamisches Öffnen und Schließen von Türen realisieren. Dadurch können zusätzliche Kosten für ungenutzte

Kapazitäten vermieden und Engpässe an anderen Knotenpunkten flexibler bewältigt werden.

Ein weiterer Optimierungsansatz betrifft den derzeit statischen Schwankungsbereich der Abweichungen. Dieser ist aktuell für die Transport- und Bearbeitungszeiten auf eine feste Spanne zwischen -10 Prozent und +20 Prozent begrenzt. Für den Einsatz in der Praxis wäre es sinnvoll, diesen Bereich dynamisch und datenbasiert zu gestalten. Eine Möglichkeit besteht in der Integration von Echtzeitdaten, um die stochastischen Parameter an den jeweils aktuellen Systemzustand anzupassen. So könnten beispielsweise Live-Verkehrs-, Wetter- und Fahrzeugdaten genutzt werden, um kurzfristige Abweichungen tages- oder sogar stundenaktuell in die Simulation zu integrieren. Darüber hinaus eröffnet der ergänzende Einsatz von Machine-Learning-Methoden weiteres Potenzial. Auf Grundlage historischer Daten und vergangener Simulationsdurchläufe könnten adaptive Algorithmen entwickelt werden, die sich kontinuierlich an verändernde Rahmenbedingungen anpassen. Auf diese Weise könnten Entscheidungsregeln implementiert werden, die situationsabhängig automatisch die jeweils effektivste Steuerungsstrategie identifizieren und anwenden.

6.3 Kritische Reflexion des Modells

Das entwickelte Simulationsmodell bildet die realen Abläufe deutlich realistischer ab als das zugrunde liegende deterministische Modell, auf dessen Grundlage die entstandene Simulation basiert. Dennoch stellt es, trotz der Integration stochastischer Einflussgrößen, lediglich ein abstrahiertes Abbild der Realität dar. Verschiedene idealisierende Annahmen, die für die Durchführung der Simulation erforderlich waren, schränken die Übertragbarkeit auf reale Netzwerke ein. Dazu zählen beispielsweise die Annahmen, dass sämtliche Fahrzeuge exakt zu den vorgesehenen Startzeiten verfügbar sind, dass Transportrouten dauerhaft befahrbar bleiben und keine unvorhersehbaren Straßensperrungen zu Routenänderungen führen oder dass keine Ausfälle des Personals auftreten.

Auch die im Modell implementierten Entscheidungsregeln bieten erhebliche Verbesserungsmöglichkeiten hinsichtlich der Zielgröße der verspäteten Warenlieferungen. Da sie unter idealisierten Bedingungen ausgeführt werden, sind in der praktischen Umgebung zahlreiche Rahmenbedingungen zu berücksichtigen, die eine direkte Übertragbarkeit der Regeln erschweren. So erfordert das dynamische Öffnen

zusätzlicher Ladetüren in der Realität ein kontinuierliches Monitoring der Auslastung, eine durchgängige Kommunikation mit den Fahrern sowie die Integration von Echtzeitdaten. Nur auf dieser Basis kann flexibel auf Veränderungen reagiert und die Ressourcenkapazität angepasst werden. Gleiches gilt für den Einsatz der Pufferzeitregel, die im Modell auf einem festen vorgegebenen Wert beruht. In realen Netzwerken muss dieser Parameter situationsabhängig konfiguriert werden, um eine Verschlechterung der Zielgröße zu vermeiden und verbesserte Ergebnisse zu erzielen. Die Anwendung beider Logiken erfordert somit nicht nur den Ausbau der technischen Infrastruktur, sondern auch einen gesteigerten organisatorischen Aufwand, was einen erhöhten Kosteneinsatz zur Folge hat.

Ebenso ist die aktuelle Verwendung der stochastischen Elemente innerhalb der Simulation kritisch zu betrachten. Zwar ermöglicht der Einsatz von Zufallsgeneratoren in Kombination mit einem definierten Schwankungsbereich eine realitätsnähere Abbildung des Modells, jedoch besteht die Gefahr, dass dadurch verzerrte Ergebnisse in der Praxis entstehen, insbesondere wenn aktuelle Schwankungsursachen, wie beispielsweise die derzeitige Verkehrslage oder Live-Wetterdaten nicht integriert werden. Um die Qualität der Simulationsergebnisse in Bezug auf die stochastischen Schwankungen weiter zu verbessern, wäre eine datenbasierte Anpassung dieses Bereichs empfehlenswert. Die Umsetzung könnte beispielsweise durch die Integration historischer Erfahrungswerte und Live-Daten erfolgen.

Insgesamt lässt sich festhalten, dass der Einsatz der integrierten Entscheidungslogiken nur eingeschränkt auf reale Logistiknetzwerke übertragbar ist. Das Modell liefert zwar wertvolle Erkenntnisse hinsichtlich der Wirkung der Optimierungsmaßnahmen, für einen praktischen Einsatz ist jedoch eine weiterführende Implementierung erforderlich, die dynamische und kontextbezogene Reaktionen ermöglicht. Die aktuelle Modellversion dient somit in erster Linie als Unterstützung bei strategischen Entscheidungen. Aufgrund der vereinfachten Annahmen ist sie allerdings nicht unmittelbar auf die oftmals komplexen und dynamischen Strukturen realer Netzwerke anwendbar und daher nicht als finales Steuerungselement geeignet.

6.4 Bewertung des Modells

Die entwickelte Simulation stellt durch den Einsatz stochastischer Abweichungen sowie gezielter Entscheidungsregeln mit dem Ziel der Ergebnisoptimierung einen deutlich höheren Realitätsbezug im Vergleich zum deterministischen Ausgangsmodell her. Besonders die Einführung zufälliger Schwankungen in die Liefer- und Bearbeitungszeiten bildet praxisnahe Unsicherheiten ab, wie sie auch im realen Alltag in der Logistik regelmäßig eintreten, wie beispielsweise durch Verkehrsstörungen oder Verzögerungen bei der Be- und Entladung.

Ein grundlegendes Merkmal des Modells ist die parallele Durchführung der Standardvariante und der optimierten Variante unter identischen Rahmenbedingungen. Da die stochastischen Abweichungen in beiden Durchläufen gleich sind, wird eine direkte Vergleichbarkeit der Simulationsergebnisse gewährleistet. Dabei wird deutlich, dass die Entscheidungsregeln des optimierten Modells zu signifikanten Verbesserungen führen, da sie gezielt auf die stochastisch auftretenden Schwankungen reagieren. Diese positiven Effekte konnten in sämtlichen Netzwerkvarianten und unterschiedlichen Modellkonfigurationen nachgewiesen werden. Die wiederholt auftretende Wirksamkeit der Maßnahmen über verschiedene Ausprägungen hinweg unterstreicht die Robustheit und Belastbarkeit der Ergebnisse sowie der zugrunde liegenden Modelle.

Ein weiterer Vorteil liegt in der hohen Flexibilität des Modells. Die technische Struktur erlaubt eine unkomplizierte Anpassung an verschiedene Modellgrößen, Kapazitäten und Parameter. Unabhängig von der Anzahl an Knotenpunkten oder von den eingesetzten Ressourcenkonfigurationen ist die Simulation auf umfangreiche Netzwerke übertragbar und somit auch für komplexe Anwendungsszenarien geeignet. Damit stellt der Modellaufbau eine belastbare Grundlage für die Bewertung logistischer Netzwerke unter Einbeziehung stochastischer Abweichungen zur Verfügung.

Die in Kapitel 6.2 angeführten Optimierungspotenziale bilden die Basis für zukünftige Erweiterungen. Ergänzend dazu könnte das Modell um weitere Entscheidungsregeln ausgebaut werden. Denkbar wären beispielsweise dynamische Tourenentscheidungen oder Umleitungsstrategien, die bei überlasteten Knotenpunkten eine automatische Routenanpassung ermöglichen. Auch eine Weiterentwicklung der Pufferzeitregel wäre denkbar, indem adaptive und situationsabhängig kalibrierte Pufferzeiten statt festgelegter Wartephasen verwendet werden. Dadurch ließen sich Zeiträume vermeiden, in denen Fahrzeuge pausieren, ohne dass ein weiteres Fahrzeug zur Priorisierung eintrifft.

7 Fazit

Im Rahmen dieser Masterarbeit wurde ein Simulationsmodell zur Analyse und Optimierung eines logistischen Netzwerks unter Berücksichtigung stochastischer Einflussfaktoren entwickelt und umfassend untersucht. Ausgehend war die Feststellung, dass reale logistische Prozesse erheblichen Unsicherheiten unterliegen, wie etwa durch Verkehrsstörungen, Umwelteinflüsse oder begrenzte Ressourcenkapazitäten. Zur realitätsnäheren Abbildung dieser Unsicherheiten wurde ein stochastisches Modell entwickelt, welches auf einem bestehenden deterministischen Netzwerkmodell basiert und um zufallsbasierte Abweichungen erweitert wurde.

Aufbauend auf diesem Modell wurde eine erweiterte Simulationsvariante entwickelt, in der verschiedene Entscheidungsregeln integriert wurden, um das Systemverhalten gezielt zu optimieren. Das wesentliche Ziel bestand darin, die Anzahl verspätet eintreffender Warenlieferungen zu minimieren, ohne dabei negative Auswirkungen auf andere Komponenten des Systems zu verursachen.

Im Zentrum der Untersuchung stand die Frage, ob sich die Ergebnisse des bestehenden deterministischen Modells unter Berücksichtigung stochastischer Einflussfaktoren und durch den Einsatz zusätzlicher Optimierungsmaßnahmen im Hinblick auf die termingerechte Ankunft von Warenlieferungen verbessern lassen. Diese Frage kann auf Grundlage der durchgeführten Simulationsläufe und der detaillierten Ergebnisanalyse eindeutig positiv beantwortet werden.

Die Resultate verdeutlichen, dass die gezielte Integration von Entscheidungsregeln erhebliche Effekte auf das Gesamtsystem bewirkte. Besonders hervorzuheben ist die dynamische Erweiterung der Bearbeitungskapazität durch das Öffnen zusätzlicher Ladetore. Diese Maßnahme sorgte für beträchtliche Verbesserungen in Bezug auf die durchschnittliche und maximale Wartezeit, die Warteschlangenlänge sowie die Verspätung der Warenlieferungen. Gleichzeitig zeigte sich jedoch, dass die Erweiterung um mehr als eine zusätzliche Ladetür lediglich einen geringfügigen zusätzlichen Effekt bewirkte. Darüber hinaus wurde die zusätzliche Ressource nur von einem vergleichsweise geringen Anteil der Lieferfahrzeuge in Anspruch genommen, während die damit verbundenen potenziellen Kosten unverhältnismäßig anstiegen. Dies weist auf ein abnehmendes Kosten-Nutzen-Verhältnis hin, das im Hinblick auf die Zweckmäßigkeit der Maßnahme kritisch zu beurteilen ist.

Eine weitere getestete Entscheidungsregel war die Integration einer definierten Pufferzeit. Dieses Vorgehen ermöglichte es Fahrzeuge gezielt anzustauen, um leicht verspätet eintreffende Fahrzeuge gegebenenfalls priorisiert zu behandeln. Obwohl durch diese Regel temporär längere Warteschlangen entstanden, konnten sowohl die durchschnittliche Wartezeit als auch die Anzahl verspätet eintreffender Waren reduziert werden.

Besonders wirkungsvoll erwies sich jedoch die kombinierte Anwendung aller Entscheidungsregeln. In dieser Variante konnten negative Effekte gegenseitig kompensiert werden. So wurde beispielsweise dem durch die Pufferzeit auftretenden Anstieg der Warteschlangen durch die zusätzliche Ladetür erfolgreich entgegengewirkt. Insgesamt erzielte die kombinierte Variante im Hinblick auf die definierte Zielgröße und den damit verbundenen Ressourceneinsatz die besten Ergebnisse. Besonders hervorzuheben ist, dass bereits das Öffnen nur eines zusätzlichen Ladetores in Kombination mit den weiteren Maßnahmen ausreichte, um ein ausgewogenes Verhältnis zwischen Kosten und Nutzen zu erreichen. Vor diesem Hintergrund ist diese Konfiguration insgesamt als die effektivste und zugleich empfehlenswerteste Lösung zu bewerten.

Der Vergleich zwischen dem stochastischen Standardmodell und den erweiterten Modellvarianten verdeutlicht, dass die Berücksichtigung zufallsbedingter Schwankungen nicht nur zu einer realitätsnäheren Darstellung logistischer Netzwerke beiträgt, sondern zugleich den gezielten Einsatz intelligenter und unterstützender Steuerungsmechanismen erfordert. Die ergänzenden Entscheidungsregeln ermöglichen es, flexibel auf sich verändernde Systemzustände zu reagieren, potenzielle Engpässe frühzeitig zu vermeiden und die Effizienz des Netzwerks nachhaltig zu verbessern. Die robuste und effiziente Funktionsweise des entwickelten Ansatzes konnte durch zahlreiche Simulationsdurchläufe über verschiedene Modellgrößen, Netzwerkkonfigurationen und Parameter hinweg nachgewiesen werden, was die Übertragbarkeit des Modells unterstreicht.

Darüber hinaus lassen sich die Ergebnisse dieser Arbeit schlüssig in den bestehenden wissenschaftlichen Forschungsstand einordnen. Während klassische Planungsansätze im logistischen Kontext oftmals auf deterministischen Modellen basieren und damit von konstanten Parametern ausgehen, zeigt das entwickelte Simulationsmodell die Relevanz stochastischer Einflussfaktoren für eine realitätsnahe Entscheidungsbasis explizit auf. Die Kombination aus zufallsbasierten Abweichungen und zusätzlichen

Optimierungsmaßnahmen stellt eine praxisnahe Erweiterung dieser Ansätze dar und verstärkt die Relevanz dynamischer Steuerungsmechanismen in realen Umgebungen.

Aufbauend auf den gewonnenen Erkenntnissen ergeben sich verschiedene Ansatzpunkte für zukünftige Forschungsarbeiten. Eine vielversprechende Weiterentwicklung ist die Integration von Echtzeitdaten, um die Schwankungsbereiche der stochastischen Eingangsparameter dynamisch an den aktuellen Systemzustand anzupassen. Durch die Anbindung an Verkehrsdaten, Wetterprognosen und Live-Fahrzeugdaten ließen sich beispielsweise die Schwankungsbereiche der Transport- und Bearbeitungszeiten künftig dynamischer und realitätsnäher abbilden. Zudem könnten kurzfristige Abweichungen, die infolge von Staus oder Unwetterereignissen im Tagesverlauf eintreten, deutlich präziser in die Simulation integriert werden. Dies würde die Reaktionsfähigkeit des Systems im operativen Einsatz erheblich verbessern.

Darüber hinaus eröffnet der ergänzende Einsatz dynamischer Entscheidungsregeln auf Grundlage von Methoden des maschinellen Lernens ein erhebliches Entwicklungspotenzial. Durch die Anwendung adaptiver Algorithmen könnten Steuerungsansätze entwickelt werden, die sich fortlaufend an verändernde Daten anpassen. Diese basieren auf den Erkenntnissen vergangener Simulationsdurchläufe sowie auf praktischen Anwendungsdaten und sind in der Lage, eigenständig die jeweils optimale Steuerungsstrategie auszuwählen.

Abschließend lässt sich festhalten, dass die im Rahmen dieser Arbeit entwickelten Simulationsmodelle sowie die daraus entstandenen Analyseansätze einen wichtigen Beitrag zur Weiterentwicklung anpassungsfähiger und realitätsnaher logistischer Netzwerke leisten. Sie ermöglichen eine grundlegende Bewertung des Systemverhaltens unter Unsicherheit und verdeutlichen das Potenzial der gewählten Entscheidungsmechanismen zur Effizienzsteigerung in komplexen Netzwerken. Die Verknüpfung stochastischer Abweichungen mit integrierten Steuerungsregeln eröffnet neue Perspektiven zur gezielten Optimierung logistischer Prozesse. Das entwickelte Simulationsmodell ist jedoch nicht als abschließendes Entscheidungstool zu verstehen, sondern vielmehr als unterstützende Grundlage für Entscheidungsprozesse in realen Anwendungsbereichen. Es ermöglicht die praxisorientierte Bewertung logistischer Systeme und bietet zugleich eine fundierte Basis für weiterführende Forschungen und zukünftige Entwicklungen im Bereich dynamischer Logistiknetzwerke. Insbesondere im Hinblick auf die zunehmende Komplexität moderner Lieferketten liefert die vorliegende Arbeit wichtige Impulse für die Integration adaptiver und flexibler Planungsansätze.

Literaturverzeichnis

Altiok, T., & Melamed, B. (2010). *Simulation Modeling and Analysis with ARENA*. Elsevier Science.

Bandyopadhyay, S., & Bhattacharya, R. (2014). *Discrete and continuous simulation: Theory and practice*. CRC Press.

Bankhofer, U. (2022). Simulation. In U. Bankhofer, *Quantitative Unternehmensplanung* (S. 299–311). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-8348-2466-0_16

Belieres, S., Scherr, Y. O., & Hewitt, M. (2025). *The Scheduled Service Network Design Problem with Terminal Resource Acquisition*. *1*. https://hal.science/hal-05046720v1

Benidis, K., Paschos, G., Gross, M., & Iosifidis, G. (2023). *Middle-mile optimization for next-day delivery* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2310.18388

Bernardo, M., & Hillston, J. E. (Hrsg.). (2007). *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-72522-0

Bi, S., Beer, M., Cogan, S., & Mottershead, J. (2023). Stochastic Model Updating with Uncertainty Quantification: An Overview and Tutorial. *Mechanical Systems and Signal Processing*, *204*, 110784. https://doi.org/10.1016/j.ymssp.2023.110784

Boland, N., Hewitt, M., Marshall, L., & Savelsbergh, M. (2019). The price of discretizing time: A study in service network design. *EURO Journal on Transportation and Logistics*, *8*(2), 195–216. https://doi.org/10.1007/s13676-018-0119-x

Crainic, T. G. (2000). Service network design in freight transportation. *European Journal of Operational Research*, *122*(2), 272–288. https://doi.org/10.1016/S0377-2217(99)00233-7

Crainic, T. G. (with Gendreau, M., & Gendron, B.). (2021). *Network Design with Applications to Transportation and Logistics*. Springer International Publishing AG.

Crainic, T. G., Ricciardi, N., & Storchi, G. (2009). Models for Evaluating and Planning City Logistics Systems. *Transportation Science*, *43*(4), 432–454. https://doi.org/10.1287/trsc.1090.0279

Dayarian, I., & Savelsbergh, M. (2020). Crowdshipping and Same-day Delivery: Employing In-store Customers to Deliver Online Orders. *Production and Operations Management*, *29*(9), 2153–2174. https://doi.org/10.1111/poms.13219

Dinkelbach, W., & Lorscheider, U. (1994). *Entscheidungsmodelle und lineare Programmierung: Übungsbuch zur Betriebswirtschaftslehre*. Oldenbourg Wissenschaftsverlag. https://doi.org/10.1515/9783486785722

Fabricius, D. (2016). Simulation. *KZfSS Kölner Zeitschrift für Soziologie und Sozialpsychologie*, *68*(3), 565–571. https://doi.org/10.1007/s11577-016-0378-1

Frota Neto, J. Q., Bloemhof-Ruwaard, J. M., Van Nunen, J. A. E. E., & Van Heck, E. (2008). Designing and evaluating sustainable logistics networks. *International Journal of Production Economics*, *111*(2), 195–208. https://doi.org/10.1016/j.ijpe.2006.10.014

Goh, M., Lim, J. Y. S., & Meng, F. (2007). A stochastic model for risk management in global supply chain networks. *European Journal of Operational Research*, *182*(1), 164–173. https://doi.org/10.1016/j.ejor.2006.08.028

Haas, M. (with Zorn, W.). (2020). *Methodische Leistungsanalyse Von Rechensystemen* (1st ed). Walter de Gruyter GmbH.

Hartig, F., Calabrese, J. M., Reineking, B., Wiegand, T., & Huth, A. (2011). Statistical inference for stochastic simulation models - theory and application: Inference for stochastic simulation models. *Ecology Letters*, *14*(8), 816–827. https://doi.org/10.1111/j.1461-0248.2011.01640.x

Hendrickson, C., & Kocur, G. (1981). Schedule Delay and Departure Time Decisions in a Deterministic Model. *Transportation Science*, *15*(1), 62–77. https://doi.org/10.1287/trsc.15.1.62

Herzog, A. (2021). *Simulation Mit Dem Warteschlangensimulator: Mathematische Modellierung und Simulation Von Produktions- und Logistikprozessen*. Springer Fachmedien Wiesbaden GmbH.

Hillier, F. S., Lieberman, G. J., Bauer, G., & Hillier, F. S. (2002). *Operations Research: Einführung* (5. Aufl., (unveränd. Nachdr. der 4. Aufl.)). Oldenbourg.

Kalayci, C. B., Ertenlice, O., & Akbay, M. A. (2019). A comprehensive review of deterministic models and applications for mean-variance portfolio optimization. *Expert Systems with Applications*, *125*, 345–368. https://doi.org/10.1016/j.eswa.2019.02.011

Klapp, M. A., Erera, A. L., & Toriello, A. (2018). The Dynamic Dispatch Waves Problem for same-day delivery. *European Journal of Operational Research*, *271*(2), 519–534. https://doi.org/10.1016/j.ejor.2018.05.032

Lium, A.-G., Crainic, T. G., & Wallace, S. W. (2009). A Study of Demand Stochasticity in Service Network Design. *Transportation Science*, *43*(2), 144–157. https://doi.org/10.1287/trsc.1090.0265

Mesbah, A. (2016). Stochastic Model Predictive Control: An Overview and Perspectives for Future Research. *IEEE Control Systems*, *36*(6), 30–44. https://doi.org/10.1109/MCS.2016.2602087

Peng, P., Snyder, L. V., Lim, A., & Liu, Z. (2011). Reliable logistics networks design with facility disruptions. *Transportation Research Part B: Methodological*, *45*(8), 1190–1211. https://doi.org/10.1016/j.trb.2011.05.022

Piontek, J. (1997). *Global Sourcing*. De Gruyter Oldenbourg. https://doi.org/10.1515/9783486791280

Roth, M. W. (2018). *Modeling and Simulation of Everyday Things* (1. Aufl.). CRC Press. https://doi.org/10.1201/9781351067751

Schweer, J., Hillerbrand, R., & Elstner, M. (2024). Simulation. In M. Gutmann, K. Wiegerling, & B. Rathgeber (Hrsg.), *Handbuch Technikphilosophie* (S. 335–344). J.B. Metzler. https://doi.org/10.1007/978-3-476-05991-8_33

Team SimPy. (2024). *SimPy—Documentation for SimPy*. https://simpy.readthedocs.io/en/latest/contents.html#

Tripp, C. (2021). Logistische Netzwerke. In C. Tripp, *Distributions- und Handelslogistik* (S. 53–167). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-34532-7_2

Ulmer, M. W. (2020). Dynamic Pricing and Routing for Same-Day Delivery. *Transportation Science*, *54*(4), 1016–1033. https://doi.org/10.1287/trsc.2019.0958

Ulmer, M. W., & Streng, S. (2019). Same-Day delivery with pickup stations and autonomous vehicles. *Computers & Operations Research*, *108*, 1–19. https://doi.org/10.1016/j.cor.2019.03.017

Ulmer, M. W., & Thomas, B. W. (2018). Same-day delivery with heterogeneous fleets of drones and vehicles. *Networks*, *72*(4), 475–505. https://doi.org/10.1002/net.21855

Uusitalo, L., Lehikoinen, A., Helle, I., & Myrberg, K. (2015). An overview of methods to evaluate uncertainty of deterministic models in decision support. *Environmental Modelling & Software*, *63*, 24–31. https://doi.org/10.1016/j.envsoft.2014.09.017

Voccia, S. A., Campbell, A. M., & Thomas, B. W. (2019). The Same-Day Delivery Problem for Online Purchases. *Transportation Science*, *53*(1), 167–184. https://doi.org/10.1287/trsc.2016.0732

Wieberneit, N. (2007). Service network design for freight transportation: A review. *OR Spectrum*, *30*(1), 77–112. https://doi.org/10.1007/s00291-007-0079-2

Wu, H., Herszterg, I., Savelsbergh, M., & Huang, Y. (2023). Service Network Design for Same-Day Delivery with Hub Capacity Constraints. *Transportation Science*, *57*(1), 273–287. https://doi.org/10.1287/trsc.2022.1155

Anhang

Anhang 1: Python-Modul „FileHandler“

```
import os
import glob
import pandas as pd

# ================================================
# Defining input and output folders
# ================================================

# Define input folder
def read_in_file(folder, keyword, extension="*.txt"):
    pattern = os.path.join(folder, f"*{keyword}*{extension}")
    files = glob.glob(pattern)
    if files:
        return files
    else:
        raise FileNotFoundError(f"No data with keyword {keyword} found in {folder}.")

input_folder = "/Users/johannesschumann/Documents/Uni_Wien/4.Semester/Masterarbeit/Data-File/input"
instance_path = read_in_file(input_folder, "instance", ".txt")[0]
results_path = read_in_file(input_folder, "networkdesign", ".txt")[0]

# Define output folder
output_folder = "/Users/johannesschumann/Documents/Uni_Wien/4.Semester/Masterarbeit/Data-File/output"

def save_output_file(keyword, extension=".csv"):
    return os.path.join(output_folder, f"{keyword}{extension}")

# ================================================
# Reading information from instance file
# ================================================
def read_arcs():
    # Data path
    instance_file = instance_path
    # Read data file
    with open(instance_file, 'r', encoding='utf-8') as file:
        file_content = file.readlines()

    # Extract relevant data
    start_extracting = False
    data_lines = []

    for line in file_content:
        if "ARCS" in line:
            start_extracting = True
            continue
        elif "COMMODITIES" in line:
            break
        if start_extracting:
            data_lines.append(line.strip())

```

```
        # Define column names
        column_names = ["Index", "Start node", "End node", "variable
costs", "fixed costs", "capacity", "# periods",
                        "distance", "# periods redundant"]

        # Create data frame
        df = pd.DataFrame([line.split(',') for line in data_lines],
columns=column_names)

        # Export as csv-file
        output_path_arcs = save_output_file("arc_data")
        df.to_csv(output_path_arcs, index=False)

        print(f"The arc-csv-file is saved under
'{output_path_arcs}'.")

    # ================================================
    # Reading information from networkdesign file
    # ================================================
    def read_trucks():
        # Data path
        results_file = results_path

        # Read data file
        with open(results_file, 'r', encoding='utf-8') as file:
            file_content = file.readlines()

        # Extract relevant data
        start_extracting = False
        data_lines = []

        for line in file_content:
            if "EXPANDED_VEHICLE" in line:
                start_extracting = True
                continue
            elif "EXPANDED_FLOWS" in line:
                break
            if start_extracting:
                data_lines.append(line.strip())

        # Define column names
        column_names = ["Truck", "Arc", "Start time", "End time",
"selected"]

        # Assign truck numbers
        df = pd.DataFrame([line.split(',') for line in data_lines],
columns=column_names[1:])
        df.insert(0, "Truck", range(1, len(df) + 1))

        # Export as csv-file
        output_path_trucks = save_output_file("truck_data")
        df.to_csv(output_path_trucks, index=False)

        print(f"The truck-csv-file is saved under
'{output_path_trucks}'.")

    # ================================================
    # Reading in the capacities
    # ================================================
    def read_capacities():
        # Data path
```

```
106      results_file = results_path
107
108      # Read data file
109      with open(results_file, 'r', encoding='utf-8') as file:
110          file_content = file.readlines()
111
112      # Extract capacities
113      inbound_capacities = {}
114      outbound_capacities = {}
115      is_inbound = False
116      is_outbound = False
117
118      for line in file_content:
119          line = line.strip()
120          if line.startswith("INBOUND_DOOR_CAPACITY"):
121              is_inbound = True
122              is_outbound = False
123              continue
124          elif line.startswith("OUTBOUND_DOOR_CAPACITY"):
125              is_inbound = False
126              is_outbound = True
127              continue
128          elif line.startswith("PATHS"):
129              break
130          elif is_inbound and line:
131              node_ID_str, capacity_str = line.split(',')
132              node_ID = int(float(node_ID_str))
133              capacity = int(float(capacity_str))
134              inbound_capacities[node_ID] = capacity
135          elif is_outbound and line:
136              node_ID_str, capacity_str = line.split(',')
137              node_ID = int(float(node_ID_str))
138              capacity = int(float(capacity_str))
139              outbound_capacities[node_ID] = capacity
140
141      return inbound_capacities, outbound_capacities
142
143  # ================================================
144  # Reading in the loading and waiting times
145  # ================================================
146  def read_times():
147      # Data Path:
148      instance_file = instance_path
149
150      # Define variables
151      loading_periods = 0
152      waiting_periods = 0
153      discretization_minutes = 30  # Default value
154
155      # Read data file
156      with open(instance_file, 'r', encoding='utf-8') as file:
157          for line in file:
158              line = line.strip()
159              if line.startswith("discretization"):
160                  disc_str = line.split('=')[1]
161                  discretization_minutes = float(disc_str)
162              elif line.startswith("loading"):
163                  loading_str = line.split('=')[1]
164                  loading_periods = float(loading_str) /
discretization_minutes
165              elif line.startswith("waiting"):
```

```
                waiting_str = line.split('=')[1]
                waiting_periods = float(waiting_str) /
discretization_minutes
    return loading_periods, waiting_periods,
discretization_minutes

# ================================================
# Reading in the node properties
# ================================================
def read_nodes():
    # Data path:
    instance_file = instance_path

    # Read data file
    with open(instance_file, 'r', encoding='utf-8') as file:
        file_content = file.readlines()

    start_extracting = False
    num_nodes = None
    node_properties = {}

    # Extract relevant data
    for x, line in enumerate(file_content):
        if line.startswith("NODES,"):
            parts = line.strip().split(',')
            try:
                num_nodes = int(parts[1])
            except ValueError:
                print("Error when reading in number of nodes")
                return {}
            start_extracting = True
            continue

        if line.startswith("ARCS"):
            break

        if start_extracting:
            node_parts = line.strip().split(',')
            if len(node_parts) >= 3:
                try:
                    node_ID = int(node_parts[0])
                except ValueError:
                    continue
                node_property = node_parts[2]
                node_properties[node_ID] = node_property

            if num_nodes is not None and len(node_properties) >=
num_nodes:
                break

    return node_properties

# ================================================
# Combine created files
# ================================================
def combine_files():
    # Paths to the CSV files
    output_path_arcs = save_output_file("arc_data")
    output_path_trucks = save_output_file("truck_data")
    output_path_Combined = save_output_file("combined_data")

```

```
224       # Read CSV files into data frames
225       df_arcs = pd.read_csv(output_path_arcs)
226       df_trucks = pd.read_csv(output_path_trucks)
227
228       # Ensure columns are the same type
229       df_trucks['Arc'] = df_trucks['Arc'].astype(int)
230       df_arcs['Index'] = df_arcs['Index'].astype(int)
231
232       # Merge data frames (auf Basis der Arc-Nummer)
233       df_combined = pd.merge(df_trucks, df_arcs, left_on='Arc',
right_on='Index', how='left')
234
235       # Select required columns
236       df_output = df_combined[['Truck', 'Arc', 'Start node', 'End
node', 'Start time', 'End time', '# periods']]
237
238       # Export combined data frame to CSV file
239       df_output.to_csv(output_path_Combined, index=False)
240
241       print(f"The combined-csv-file is saved under
{output_path_Combined}.")
242
243       # Combined file for Main class
244       return output_path_Combined
245
246   # ================================================
247   # Assign commodities to trucks
248   # ================================================
249   def assign_commodities():
250       # Data paths:
251       instance_file = instance_path
252       results_file = results_path
253       combined_path = save_output_file("combined_data")
254       output_path_Commodity = save_output_file("commoditiy_data")
255
256       # Read data file
257       with open(results_file, 'r', encoding='utf-8') as file:
258           result_lines = file.readlines()
259
260       # Determine number of flow lines
261       flows_index = None
262       number_flows = None
263       for i, line in enumerate(result_lines):
264           if line.startswith("EXPANDED_FLOWS"):
265               parts = line.strip().split(',')
266               if len(parts) >= 2:
267                   try:
268                       number_flows = int(parts[1])
269                   except:
270                       number_flows = None
271               flows_index = i
272               break
273
274       # Create Dictionary for Commodities:
275       commodity_to_flows = {}
276       flow_lines = result_lines[flows_index + 1: flows_index + 1 +
number_flows]
277       for line in flow_lines:
278           parts = line.strip().split(',')
279           # Check the right format
280           if len(parts) < 5:
```

```
                continue
            try:
                arc_ID = int(float(parts[0]))
                commodity_start = float(parts[1])
                commodity_end = float(parts[2])
                commodity_ID = int(float(parts[3]))
            except:
                continue
            # Ignore Waiting-flows (lines with a waiting period 1)
            if (commodity_end - commodity_start) <= 1:
                continue
            if commodity_ID not in commodity_to_flows:
                commodity_to_flows[commodity_ID] = []
            commodity_to_flows[commodity_ID].append((arc_ID,
commodity_start, commodity_end))

        # Read combined file
        df_combined = pd.read_csv(combined_path)
        # Convert times to numerical values
        df_combined['Start time'] = pd.to_numeric(df_combined['Start
time'], errors='coerce')
        df_combined['End time'] = pd.to_numeric(df_combined['End
time'], errors='coerce')
        df_combined['Arc'] = df_combined['Arc'].astype(int)

        # Create Assignment (truck and commodity)
        commodity_to_trucks = {}
        for commodity, flows in commodity_to_flows.items():
            for (arc_ID, commodity_start, commodity_end) in flows:
                matching_rows = df_combined[
                    (df_combined['Arc'] == arc_ID) &
                    (df_combined['Start time'] == commodity_start) &
                    (df_combined['End time'] == commodity_end)
                ]
                if not matching_rows.empty:
                    for truck in matching_rows['Truck']:
                        if commodity not in commodity_to_trucks:
                            commodity_to_trucks[commodity] = set()
                        commodity_to_trucks[commodity].add(truck)

        # Get all commodities for each truck
        truck_to_commodity = {}
        for commodity, trucks in commodity_to_trucks.items():
            for truck in trucks:
                if truck not in truck_to_commodity:
                    truck_to_commodity[truck] = set()
                truck_to_commodity[truck].add(commodity)

        # Assign latest arrival time for commodities
        with open(instance_file, 'r', encoding='utf-8') as file2:
            instance_lines = file2.readlines()

        commodity_index = None
        number_commodities = None
        for i, line in enumerate(instance_lines):
            if line.startswith("COMMODITIES"):
                parts = line.strip().split(',')
                if len(parts) >= 2:
                    try:
                        number_commodities = int(parts[1])
                    except:
```

```
                        number_commodities = None
                commodity_index = i
                break
        commodity_size = {}
        commodity_latest_arrival = {}
        if commodity_index is not None and number_commodities is not
None:
            commodity_lines = instance_lines[commodity_index + 1:
commodity_index + 1 + number_commodities]
            for line in commodity_lines:
                parts = line.strip().split(',')
                if len(parts) >= 6:
                    try:
                        com_ID = int(parts[0])
                        size = float(parts[3])
                        latest_arrival = float(parts[5])
                        commodity_size[com_ID] = size
                        commodity_latest_arrival[com_ID] =
latest_arrival
                    except:
                        continue

        # Create list
        output_rows = []
        for truck in truck_to_commodity:
            sorted_commodities = sorted(truck_to_commodity[truck])
            commodity_str = ', '.join(map(str, sorted_commodities))
            latest_arrivals = []
            single_commodity_size = []
            total_commodity_size = 0
            for comm in sorted_commodities:
                if comm in commodity_latest_arrival:

latest_arrivals.append(str(commodity_latest_arrival[comm]))
                else:
                    latest_arrivals.append("N/A")
            for comm in sorted_commodities:
                if comm in commodity_size:

single_commodity_size.append(f"{commodity_size[comm]:.2f}")
                    total_commodity_size += (commodity_size[comm])

            latest_arrivals_str = ', '.join(latest_arrivals)
            output_rows.append({'Truck': truck, 'Commodity':
commodity_str, 'Latest arrival': latest_arrivals_str, 'Single size':
", ".join(single_commodity_size), 'Total size':
f"{total_commodity_size:.2f}"})

        # Sort list
        output_rows_sorted = sorted(output_rows, key=lambda x:
x['Truck'])

        # Create data frame
        df_commodity = pd.DataFrame(output_rows_sorted)

        # Export as csv-file
        df_commodity.to_csv(output_path_Commodity, index=False)
        print(f"The commodity-csv-file is saved under
{output_path_Commodity}.")
        return output_path_Commodity
```

Anhang 2: Python-Modul „Simulation“

```
1    import random
2    import simpy
3    import pandas as pd
4    import subprocess
5
6    # Calculate max number of simultaneously overlapping intervals
for max queue length
7    def calculate_overlap(intervals):
8        events = []
9        for start, end in intervals:
10           events.append((start, 1))
11           events.append((end, -1))
12
13       # To avoid waiting trucks with waiting time of 0
14       events.sort(key=lambda x: (x[0], x[1]))
15       current = 0
16       max_current = 0
17       for time, delta in events:
18           current += delta
19           if current > max_current:
20               max_current = current
21       return max_current
22
23   # ================================================
24   # Create simulation class
25   # ================================================
26   class TruckSimulation:
27       def __init__(
28               self,
29               combined_data_path,
30               inbound_capacities,
31               outbound_capacities,
32               travel_time_down,
33               travel_time_up,
34               loading_periods,
35               processing_time_down,
36               processing_time_up,
37               commodity_data_path,
38               batching_delay,
39               door_number,
40               optimized_mode=False,
41               node_properties=None,
42               truck_deviations=None
43       ):
44
45           self.combined_data_path = combined_data_path
46           self.outbound_capacities = outbound_capacities
47           self.inbound_capacities = inbound_capacities
48           self.travel_time_down = travel_time_down
49           self.travel_time_up = travel_time_up
50           self.loading_periods = loading_periods
51           self.processing_time_down = processing_time_down
52           self.processing_time_up = processing_time_up
53           self.commodity_data_path = commodity_data_path
54           self.batching_delay = batching_delay
55           self.door_number = door_number
56           self.optimized_mode = optimized_mode
57           self.node_properties = node_properties
58
```

```
59            # Read data files
60            self.truck_data = pd.read_csv(self.combined_data_path)
61            self.df_commodities =
pd.read_csv(self.commodity_data_path)
62
63            # Calculate random deviations per truck
64            if truck_deviations is None:
65                self.truck_deviations = {}
66                for x, row in self.truck_data.iterrows():
67                    truck_ID = int(row['Truck'])
68                    travel_time_deviation = random.triangular(-
self.travel_time_down, self.travel_time_up, 0)
69                    processing_time_deviation_outbound =
random.triangular(-self.processing_time_down, self.processing_time_up,
0)
70                    processing_time_deviation_inbound =
random.triangular(-self.processing_time_down, self.processing_time_up,
0)
71                    self.truck_deviations[truck_ID] = {
72                        "travel_time_deviation":
travel_time_deviation,
73                        "processing_time_deviation_outbound":
processing_time_deviation_outbound,
74                        "processing_time_deviation_inbound":
processing_time_deviation_inbound
75                    }
76            else:
77                self.truck_deviations = truck_deviations
78
79            # Initialize SimPy environment
80            self.env = simpy.Environment()
81
82            # Create data structures
83            self.outbound_resources = {}
84            self.inbound_resources = {}
85            self.outbound_waiting_intervals = {}
86            self.inbound_waiting_intervals = {}
87            self.trucks_waited_outbound = {}
88            self.trucks_waited_inbound = {}
89            self.truck_records = []
90            if self.optimized_mode:
91                self.extra_outbound_doors = 0
92                self.extra_inbound_doors = 0
93                self.current_outbound_queue = {}
94                self.current_inbound_queue = {}
95                self.trucks_used_extra_outbound = {}
96                self.trucks_used_extra_inbound = {}
97
98            # Identify all nodes from file
99            start_nodes = self.truck_data['Start
node'].astype(int).unique()
100           end_nodes = self.truck_data['End
node'].astype(int).unique()
101           self.all_nodes = set(start_nodes).union(set(end_nodes))
102           self.initialize_resources()
103
104       # Assign value to resources
105       def initialize_resources(self):
106           for node_ID in self.all_nodes:
107               outbound_capacity =
self.outbound_capacities.get(node_ID, 1)
```

```
108                 inbound_capacity =
self.inbound_capacities.get(node_ID, 1)
109                 if self.optimized_mode:
110                     self.outbound_resources[node_ID] =
simpy.PriorityResource(self.env, capacity=outbound_capacity)
111                     self.inbound_resources[node_ID] =
simpy.PriorityResource(self.env, capacity=inbound_capacity)
112                     # Waiting queue
113                     self.current_outbound_queue[node_ID] = 0
114                     self.current_inbound_queue[node_ID] = 0
115                     # Additional doors
116                     self.trucks_used_extra_outbound[node_ID] = set()
117                     self.trucks_used_extra_inbound[node_ID] = set()
118                 else:
119                     self.outbound_resources[node_ID] =
simpy.Resource(self.env, capacity=outbound_capacity)
120                     self.inbound_resources[node_ID] =
simpy.Resource(self.env, capacity=inbound_capacity)
121
122                 self.outbound_waiting_intervals[node_ID] = []
123                 self.inbound_waiting_intervals[node_ID] = []
124                 self.trucks_waited_outbound[node_ID] = 0
125                 self.trucks_waited_inbound[node_ID] = 0
126
127         # Define run method (Start Iteration 1)
128         def run_iteration1(self):
129             # Simulation for every single truck
130             for x, row in self.truck_data.iterrows():
131                 self.env.process(self.truck_process(row))
132
133             # Run Simulation
134             self.env.run()
135
136             # Print statistics
137             self.print_statistics(iteration="1")
138
139         # Define run method (Start Iteration 2)
140         def run_iteration2(self):
141             # New environment and resources
142             self.env = simpy.Environment()
143             self.outbound_resources = {}
144             self.inbound_resources = {}
145             self.outbound_waiting_intervals = {}
146             self.inbound_waiting_intervals = {}
147             self.trucks_waited_outbound = {}
148             self.trucks_waited_inbound = {}
149             self.truck_records = []
150             self.initialize_resources()
151
152             # Simulation for every single truck
153             for x, row in self.truck_data.iterrows():
154                 self.env.process(self.truck_process(row))
155
156             # Run Simulation
157             self.env.run()
158
159             # Print statistics
160             self.print_statistics(iteration="2")
161
162         # ================================================
163         # Define truck lifecycle
```

```
164         # ================================================
165         def truck_process(self, truck_info):
166             truck_ID = int(truck_info['Truck'])
167             start_node = int(truck_info['Start node'])
168             end_node = int(truck_info['End node'])
169             start_time = float(truck_info['Start time'])
170             end_time = float(truck_info['End time'])
171             travel_time = int(truck_info['# periods'])
172             det_truck_waiting = int((end_time - start_time) -
travel_time - self.loading_periods * 2)
173 
174             # Calculate remaining time to commodity deadline if
simulation go as planned
175             remaining_time = None
176             earliest_deadline = None
177 
178             truck_data =
self.df_commodities[self.df_commodities['Truck'] == truck_ID]
179             if not truck_data.empty:
180                 deadlines_str = truck_data['Latest
arrival'].tolist()[0]
181                 try:
182                     deadlines = [float(deadline.strip()) for deadline
in deadlines_str.split(',') if deadline.strip()]
183                     if deadlines:
184                         earliest_deadline = min(deadlines)
185                         remaining_time = earliest_deadline - end_time
186                     else:
187                         remaining_time = None
188                 except Exception as e:
189                     print("Error in processing of deadlines.")
190                     remaining_time = None
191             else:
192                 remaining_time = None
193 
194             # Use calculated truck deviations
195             deviations = self.truck_deviations.get(truck_ID, {})
196             travel_time_deviation =
deviations.get('travel_time_deviation', 0)
197             processing_time_deviation_outbound =
deviations.get('processing_time_deviation_outbound', 0)
198             processing_time_deviation_inbound =
deviations.get('processing_time_deviation_inbound', 0)
199 
200             # Travel time
201             adjusted_travel_time = travel_time * (1 +
travel_time_deviation / 100)
202             if adjusted_travel_time < 1:
203                 adjusted_travel_time = 1
204 
205             # Processing time outbound
206             adjusted_processing_time_outbound = self.loading_periods
* (1 + processing_time_deviation_outbound / 100)
207             if adjusted_processing_time_outbound < 0:
208                 adjusted_processing_time_outbound = 0
209 
210             # Processing time inbound
211             adjusted_processing_time_inbound = self.loading_periods *
(1 + processing_time_deviation_inbound / 100)
212             if adjusted_processing_time_inbound < 0:
213                 adjusted_processing_time_inbound = 0
```

```
214
215             # Truck waits until start time
216             if self.env.now < start_time:
217                 yield self.env.timeout(start_time - self.env.now)
218
219             actual_start_time = self.env.now
220
221             # Output: Truck Start
222             print(f"Truck {truck_ID} starting at time {self.env.now}
from node {start_node} to node {end_node} with
{adjusted_travel_time:.2f} travel time.")
223
224             # ================================================
225             # Outbound (Start node)
226             # ================================================
227             outbound_resource = self.outbound_resources[start_node]
228
229             # Additional door logic
230             if self.optimized_mode:
231                 self.current_outbound_queue[start_node] += 1
232                 if self.current_outbound_queue[start_node] >= 2 and
self.extra_outbound_doors < self.door_number:
233                     outbound_resource._capacity += 1
234                     self.extra_outbound_doors += 1
235                     print(f"Added extra outbound door at node
{start_node}.")
236
237             outbound_start_waiting = self.env.now
238
239             # Create request and note the time
240             if self.optimized_mode and outbound_resource.count == 0:
241                 node_prop = self.node_properties.get(start_node)
242                 if node_prop in ["Breakbulk", "Transfer"] and
remaining_time > 0:
243                     yield self.env.timeout(self.batching_delay)
244
245             if self.optimized_mode:
246                 request_outbound =
outbound_resource.request(priority=end_time)
247             else:
248                 request_outbound = outbound_resource.request()
249
250             # Wait until resource is available
251             yield request_outbound
252
253             # Use of additional door
254             if self.optimized_mode:
255                 original = self.outbound_capacities.get(start_node,
1)
256                 if outbound_resource.count > original:
257
self.trucks_used_extra_outbound[start_node].add(truck_ID)
258
259             if self.optimized_mode:
260                 self.current_outbound_queue[start_node] -= 1
261
262             outbound_end_waiting = self.env.now
263
264             # Determine waiting time
265             wait_outbound = outbound_end_waiting -
outbound_start_waiting
```

```
266             if wait_outbound > 0:
267 
self.outbound_waiting_intervals[start_node].append((outbound_start_wai
ting, outbound_end_waiting))
268                 self.trucks_waited_outbound[start_node] += 1
269 
270             # Simulate processing time
271             yield self.env.timeout(adjusted_processing_time_outbound)
272             outbound_resource.release(request_outbound)
273 
274             # Simulate travel time
275             yield self.env.timeout(adjusted_travel_time)
276 
277             # ================================================
278             # Inbound (End node)
279             # ================================================
280 
281             # Consider waiting time of deterministic model
282             if not self.optimized_mode and det_truck_waiting > 0:
283                 det_start_waiting = self.env.now
284                 yield self.env.timeout(det_truck_waiting)
285                 det_end_waiting = self.env.now
286 
self.inbound_waiting_intervals[end_node].append((det_start_waiting,
det_end_waiting))
287                 self.trucks_waited_inbound[end_node] += 1
288 
289             inbound_resource = self.inbound_resources[end_node]
290 
291             # Additional door logic
292             if self.optimized_mode:
293                 self.current_inbound_queue[end_node] += 1
294                 if self.current_inbound_queue[end_node] >= 2 and
self.extra_inbound_doors < self.door_number:
295                     inbound_resource._capacity += 1
296                     self.extra_inbound_doors += 1
297                     print(f"Added extra inbound door at node
{end_node}.")
298 
299             inbound_start_waiting = self.env.now
300 
301             # Create request and note the time
302             if self.optimized_mode and inbound_resource.count == 0:
303                 node_prop = self.node_properties.get(end_node)
304                 if node_prop in ["Breakbulk", "Transfer"] and
remaining_time > 0:
305                     yield self.env.timeout(self.batching_delay)
306 
307             if self.optimized_mode:
308                 request_inbound =
inbound_resource.request(priority=end_time)
309             else:
310                 request_inbound = inbound_resource.request()
311 
312             # Wait until resource is available
313             yield request_inbound
314 
315             # Use of additional door
316             if self.optimized_mode:
317                 original = self.inbound_capacities.get(end_node, 1)
318                 if inbound_resource.count > original:
```

```
319
self.trucks_used_extra_inbound[end_node].add(truck_ID)
320
321             if self.optimized_mode:
322                 self.current_inbound_queue[end_node] -= 1
323
324             inbound_end_waiting = self.env.now
325
326             # Determine waiting time
327             wait_inbound = inbound_end_waiting -
inbound_start_waiting
328             if wait_inbound > 0:
329
self.inbound_waiting_intervals[end_node].append((inbound_start_waiting
, inbound_end_waiting))
330                 self.trucks_waited_inbound[end_node] += 1
331
332             # Simulate processing time
333             yield self.env.timeout(adjusted_processing_time_inbound)
334             inbound_resource.release(request_inbound)
335
336             actual_end_time = self.env.now
337
338             on_time = "Yes" if actual_end_time <= end_time else "No"
339
340             # Calculate waiting time
341             st_waiting_time = wait_outbound + wait_inbound
342             if self.optimized_mode:
343                 waiting_time = st_waiting_time -
(adjusted_processing_time_outbound + adjusted_processing_time_inbound
- (self.loading_periods * 2))
344             else:
345                 waiting_time = det_truck_waiting + st_waiting_time -
(adjusted_processing_time_outbound + adjusted_processing_time_inbound
- (self.loading_periods * 2))
346
347             if waiting_time < 0:
348                 waiting_time = 0
349
350             self.truck_records.append({
351                 "Truck": truck_ID,
352                 "Start node": start_node,
353                 "End node": end_node,
354                 "Planned Start": start_time,
355                 "Planned End": end_time,
356                 "Actual Start": actual_start_time,
357                 "Actual End": actual_end_time,
358                 "On Time": on_time,
359                 "Waiting Time": waiting_time,
360             })
361
362         # ================================================
363         # Output: Statistics
364         # ================================================
365         def print_statistics(self, iteration="1"):
366             data = []
367
368             # Queue-statistics
369             nodes = sorted(self.outbound_resources.keys())
370             for node in nodes:
```

```
371                 max_queue_outbound =
calculate_overlap(self.outbound_waiting_intervals.get(node, []))
372                 max_queue_inbound =
calculate_overlap(self.inbound_waiting_intervals.get(node, []))
373                 data.append({
374                     "Node": node,
375                     "Max Queue Length (Outbound)":
max_queue_outbound,
376                     "Trucks Waited (Outbound)":
self.trucks_waited_outbound.get(node, 0),
377                     "Max Queue Length (Inbound)": max_queue_inbound,
378                     "Trucks Waited (Inbound)":
self.trucks_waited_inbound.get(node, 0),
379                 })
380
381             df_queue = pd.DataFrame(data, columns=[
382                 "Node",
383                 "Max Queue Length (Outbound)",
384                 "Trucks Waited (Outbound)",
385                 "Max Queue Length (Inbound)",
386                 "Trucks Waited (Inbound)",
387             ])
388
389             # Summary line
390             summary_queue = {
391                 "Node": "Summary",
392                 "Max Queue Length (Outbound)": df_queue["Max Queue
Length (Outbound)"].max(),
393                 "Trucks Waited (Outbound)": df_queue["Trucks Waited
(Outbound)"].sum(),
394                 "Max Queue Length (Inbound)": df_queue["Max Queue
Length (Inbound)"].max(),
395                 "Trucks Waited (Inbound)": df_queue["Trucks Waited
(Inbound)"].sum(),
396             }
397             df_queue = pd.concat([df_queue,
pd.DataFrame([summary_queue])], ignore_index=True)
398
399             # Trucks-statistics
400             df_trucks = pd.DataFrame(self.truck_records, columns=[
401                 "Truck",
402                 "Start node",
403                 "End node",
404                 "Planned Start",
405                 "Planned End",
406                 "Actual Start",
407                 "Actual End",
408                 "On Time",
409                 "Waiting Time",
410             ])
411
412             # Table changes
413             df_trucks.sort_values(by="Truck", inplace=True)
414
415             # Merge Commodities
416             df_commodities = pd.read_csv(self.commodity_data_path)
417             # Check if commodity columns are integer
418             df_commodities['Truck'] =
df_commodities['Truck'].astype(int)
419             df_trucks['Truck'] = df_trucks['Truck'].astype(int)
420             # Merge
```

```
421          df_merged = pd.merge(df_trucks, df_commodities,
on="Truck", how="left")
422
423          # Compare Actual End with Latest Commodity End
424          def compare_end(actual_end, commodities_str,
latest_arrival_str):
425              if pd.isna(commodities_str) or
pd.isna(latest_arrival_str):
426                  return "No commodity"
427              try:
428                  commodities = [x.strip() for x in
commodities_str.split(',')]
429                  deadlines = [float(x.strip()) for x in
latest_arrival_str.split(',')]
430                  if len(commodities) != len(deadlines):
431                      return "Mismatch in commodity counts"
432              except Exception:
433                  return "Invalid format"
434
435              results = []
436              for com, deadline in zip(commodities, deadlines):
437                  if actual_end <= deadline:
438                      results.append(f"Commodity {com}: On time")
439                  else:
440                      lateness = actual_end - deadline
441                      results.append(f"Commodity {com}: Late by
{lateness:.2f}")
442              return "; ".join(results)
443
444          df_merged['Commodity Status'] = df_merged.apply(lambda
row: compare_end(row['Actual End'], row['Commodity'], row['Latest
arrival']), axis=1)
445
446          # Rounding
447          df_merged['Actual Start'] = df_merged['Actual
Start'].round(2)
448          df_merged['Actual End'] = df_merged['Actual
End'].round(2)
449          df_merged['Waiting Time'] = df_merged['Waiting
Time'].round(2)
450
451          # Calculate total number of delayed commodities
452          late_com_count = df_merged['Commodity
Status'].str.count("Late by").sum()
453
454          # Add delayed commodities to csv
455          summary_row = {col: "" for col in df_merged.columns}
456          summary_row["Truck"] = "Total delayed commodities"
457          summary_row["Planned Start"] = late_com_count
458          df_merged = pd.concat([df_merged,
pd.DataFrame([summary_row])], ignore_index=True)
459
460          # Add average waiting time to csv
461          df_merged['Waiting Time'] =
pd.to_numeric(df_merged['Waiting Time'], errors='coerce')
462          avg_waiting = df_merged['Waiting Time'].mean()
463          summary_row2 = {col: "" for col in df_merged.columns}
464          summary_row2["Truck"] = "Average waiting time"
465          summary_row2["Waiting Time"] = round(avg_waiting, 2)
466          df_merged = pd.concat([df_merged,
pd.DataFrame([summary_row2])], ignore_index=True)
```

```
467
468             # Extra door summary
469             if self.door_number > 0:
470                 if self.optimized_mode:
471                     total_trucks_outbound = sum(len(x) for x in
self.trucks_used_extra_outbound.values())
472                     total_trucks_inbound = sum(len(x) for x in
self.trucks_used_extra_inbound.values())
473                     print(f"Total trucks used extra outbound door:
{total_trucks_outbound}")
474                     print(f"Total trucks used extra inbound door:
{total_trucks_inbound}")
475
476             # Lateness summary
477             lateness_values = []
478             for status in df_merged["Commodity Status"]:
479                 try:
480                     parts = status.split(";")
481                 except:
482                     continue
483
484                 for part in parts:
485                     if "Late by" in part:
486                         try:
487
lateness_values.append(float(part.split("Late by")[1]))
488                         except:
489                             pass
490
491             if lateness_values:
492                 total_lateness = sum(lateness_values)
493                 average_lateness = total_lateness /
len(lateness_values)
494             else:
495                 total_lateness = 0
496                 average_lateness = 0
497
498             print(f"Total lateness across commodities:
{total_lateness:.2f} periods")
499             print(f"Average lateness per commodity:
{average_lateness:.2f} periods")
500
501             # Output paths
502             out_path =
f"/Users/johannesschumann/Documents/Uni_Wien/4.Semester/Masterarbeit/D
ata-File/output/queue_results-iteration_{iteration}.csv"
503             out_path2 =
f"/Users/johannesschumann/Documents/Uni_Wien/4.Semester/Masterarbeit/D
ata-File/output/simulation_results-iteration_{iteration}.csv"
504
505             # Queue-table
506             df_queue.to_csv(out_path, index=False)
507             # Truck-table
508             df_merged.to_csv(out_path2, index=False)
509
510             # Open csv (Mac)
511             subprocess.call(["open", out_path, out_path2])
```

Anhang 3: Python-Modul „Main“

```
import FileHandler
from Simulation import TruckSimulation

# Processing data
FileHandler.read_arcs()
FileHandler.read_trucks()
combined_data_path = FileHandler.combine_files()
inbound_capacities, outbound_capacities =
FileHandler.read_capacities()
commodity_data_path = FileHandler.assign_commodities()
loading_periods, waiting_periods, disc_minutes =
FileHandler.read_times()
node_properties = FileHandler.read_nodes()

# Get deviation percentages
print("\nDeviation of the travel time:")
travel_time_down = float(input("Enter the percentage for downward
deviation of the travel time (in %): "))
travel_time_up = float(input("Enter the percentage for upward
deviation of the travel time (in %): "))

print("\nDeviation of the processing time:")
processing_time_down = float(input("Enter the percentage for the
downward processing time (in %): "))
processing_time_up = float(input("Enter the percentage for the
upward processing time (in %): "))

print("\nBatching time for decision on resources:")
batching_delay = float(input("Enter the time that a resource
should wait for a priority decision (in periods): "))

print("\nExtra doors at the terminals:")
door_decision = input("Should it be possible to open more doors?
(Yes/No): ").strip().lower()
if door_decision == "yes":
    door_number = int(input("How many additional doors can be
opened at most? (Number): "))
else:
    door_number = 0

# ================================================
# Start Simulation
# ================================================

# Run first iteration (Standard)
print("\nStandard Simulation Model:")
simulation1 = TruckSimulation(
    combined_data_path,
    inbound_capacities,
    outbound_capacities,
    travel_time_down,
    travel_time_up,
    loading_periods,
    processing_time_down,
    processing_time_up,
    commodity_data_path,
    batching_delay,
    door_number,
    optimized_mode=False,
```

```
        node_properties=node_properties
    )
    simulation1.run_iteration1()

    # Run second iteration (Optimized)
    print("\nOptimized Simulation Model:")
    simulation2 = TruckSimulation(
        combined_data_path,
        inbound_capacities,
        outbound_capacities,
        travel_time_down,
        travel_time_up,
        loading_periods,
        processing_time_down,
        processing_time_up,
        commodity_data_path,
        batching_delay,
        door_number,
        optimized_mode=True,
        node_properties=node_properties,
        truck_deviations=simulation1.truck_deviations   # Use same
deviations as in iteration 1
    )
    simulation2.run_iteration2()
```