Antizipatorische Entscheidungsfindung in Shared Mobility Systemen

Freie wissenschaftliche Arbeit

zur Erlangung des akademischen Grades

„Master of Science“

Studiengang: Betriebswirtschaftslehre

an der

Wirtschaftswissenschaftlichen Fakultät

der Universität Augsburg

– Lehrstuhl für Analytics & Optimization –

Eingereicht bei:
Betreuer:
Vorgelegt von:
Adresse:

Matrikel-Nr.:
E-Mail:

Augsburg, 13.12.2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DARP	Dial-a-Ride-Problem
DDARP	Dynamic Dial-a-Ride-Problem
LNS	Large Neighborhood Search
MDP	Markov Decision Process

Symbolverzeichnis

$\mathcal{L}$	Menge aller Orte
l	Ort der Menge aller Orte $\mathcal{L}$
i	Ort der Menge aller Orte $\mathcal{L}$
j	Ort der Menge aller Orte $\mathcal{L}$
p_r	Zeit, die für die Bedienung einer Anfrage $r \in \mathcal{R}$ benötigt wird
$c_{i,j}$	Fahrtzeit des direkten Weges von Ort $i \in \mathcal{L}$ zu Ort $j \in \mathcal{L}$
$\mathcal{R}$	Menge aller Kundenanfragen
r	Anfrage aus der Menge aller Kundenanfragen $\mathcal{R}$
t_r	Zeitpunkt, zu dem die Anfrage $r \in \mathcal{R}$ eintrifft
o_r	Abholort des Kunden $r \in \mathcal{R}$
d_r	Zielort des Kunden $r \in \mathcal{R}$
b_r	früheste Abholzeit des Kunden $r \in \mathcal{R}$

e_r	späteste Absetzzeit des Kunden $r \in \mathcal{R}$
α	Parameter, der die maximal erlaubte Verspätung angibt
$\mathcal{V}$	Flotte eingesetzter Fahrzeuge
v	Fahrzeug der Fahrzeugflotte $\mathcal{V}$
$\mathcal{K}$	Menge aller Entscheidungsperioden
k	Entscheidungsperiode
s_0	Initialzustand des Systems zum Zeitpunkt 0
l_v	Ort, an dem sich Fahrzeug $v \in \mathcal{V}$ zum Zeitpunkt 0 befindet
s_k	Vorentscheidungszustand der Entscheidungsperiode $k \in \mathcal{K}$
r_k	Anfrage, die in Entscheidungsperiode $k \in \mathcal{K}$ eintrifft
l_k^v	Position des Fahrzeuges $v \in \mathcal{V}$ in Entscheidungsperiode $k \in \mathcal{K}$
$\mathcal{O}_k^v$	angenommene Anfrage, die von Fahrzeug $v \in \mathcal{V}$ in Entscheidungsperiode $k \in \mathcal{K}$ aktuell bedient wird
$\mathcal{U}_k$	Menge an angenommenen Anfragen, deren Bedienung in Entscheidungsperiode $k \in \mathcal{K}$ noch aussteht
$A^\pi(s_k)$	Aktion, die nach dem Vorentscheidungszustand s_k durch die Wahl einer Politik $\pi \in \Pi$ ausgeführt wird
Π	Menge aller Politiken
π	Politik der Menge aller Politiken Π
π^*	optimale Politik aus der Menge aller Politiken Π, die die erwartete kumulative Vergütung $v^\pi(s_0)$ maximiert
x_k	binäre Entscheidungsvariable, die angibt, ob eine Anfrage in Entscheidungsperiode $k \in \mathcal{K}$ angenommen wurde oder nicht
z_o	geplanter Zusteigezeitpunkt am Abholort o_r des Kunden $r \in \mathcal{R}$
z_d	geplanter Absetzzeitpunkt am Zielort d_r des Kunden $r \in \mathcal{R}$
$\mathcal{F}_x$	Menge aller zulässigen Tourenpläne, die nach der Akzeptanzentscheidung x_k gebildet werden können
y_k	Tourenplan, der aus der Menge aller zulässigen Tourenpläne $\mathcal{F}_x$ in Entscheidungsperiode $k \in \mathcal{K}$ ausgewählt wurde
s_k^a	Zustand nach der in Entscheidungsperiode $k \in \mathcal{K}$ durchgeführten Aktion
W_{k+1}	stochastischer Übergang von Entscheidungsperiode $k \in \mathcal{K}$ zu Entscheidungsperiode $k+1 \in \mathcal{K}$, der durch das Eintreffen der nächsten Anfrage $r_{k+1} \in \mathcal{R}$ ausgelöst wird
$v^\pi(s_0)$	kumulative Vergütung, die im Initialzustand s_0 bei der Wahl der Politik $\pi \in \Pi$ erwartet wird
B_k	Vergütung, die in Entscheidungsperiode $k \in \mathcal{K}$ erzielt wird

w	Lösung im LNS-Verfahren
w_0	Initiallösung zu Beginn des LNS-Verfahrens
w_{new}	neue Lösung, die im Rahmen eines LNS-Durchlaufs kreiert wurde
w_{best}	derzeit beste gefundene Lösung des LNS-Verfahrens
n_w	Tourenplan der Lösung w
n_{w_0}	Tourenplan der Initiallösung w_0
$n_{w_{new}}$	Tourenplan der neuen Lösung $n_{w_{new}}$
$n_{w_{best}}$	Tourenplan der besten Lösung $n_{w_{best}}$
$\lvert n_w \rvert$	Anzahl aller in Tourenplan n_w angenommenen Anfragen
$\lvert n_{w_{new}} \rvert$	Anzahl aller in Tourenplan $n_{w_{new}}$ angenommenen Anfragen
m_w	Menge der Anfragen $r \in \mathcal{R}$, die in der Lösung w nicht im Tourenplan n_w eingeplant sind
m_{w_0}	Menge der Anfragen $r \in \mathcal{R}$, die in der Initiallösung w_0 nicht eingeplant sind
$m_{w_{new}}$	Menge der Anfragen $r \in \mathcal{R}$, die in der Lösung w_{new} nicht im Tourenplan $m_{w_{new}}$ eingeplant sind
z_l^v	geplante Zeitpunkte, an denen Fahrzeug $v \in \mathcal{V}$ am Ort $l \in \mathcal{L}$ ankommt
$c(n_w)$	gesamte Fahrtzeit, die für das Absolvieren des Tourenplans n_w benötigt wird
$c(n_{w_{new}})$	gesamte Fahrtzeit, die für das Absolvieren des Tourenplans $n_{w_{new}}$ benötigt wird
$c(n_{w_{best}})$	gesamte Fahrtzeit, die für das Absolvieren des Tourenplans $n_{w_{best}}$ benötigt wird
β	Parameter, der die maximale Anzahl an Iterationen festlegt
γ	Parameter, der festlegt, wie viel Prozent der Anfragen des Tourenplans entfernt werden sollen
q	Zufallswert, der die exakte Anzahl der zu entfernenden Anfragen festlegt
δ	Parameter, der die Intensität der Verzerrung kontrolliert

1 Einleitung

Diese Masterarbeit beschäftigt sich mit dem Thema der antizipatorischen Entscheidungsfindung in Shared Mobility Systemen. Mobilitätsformen solcher Art gewinnen angesichts des immer größer werdenden Anpassungsdrucks auf die klassische Mobilität von heute zunehmend an Bedeutung. Die treibenden Kräfte des Mobilitätswandels sind hierbei ein gesellschaftlicher Wandel und technische Innovationen. Im Bereich des gesellschaftlichen Wandels stellt beispielsweise die Urbanisierung durch die Verdichtung der Städte und der damit einhergehenden Verknappung des Raumes ein erhebliches Problem für die Straßen und den Stadtverkehr dar. Neue Mobilitätsformen müssen somit auf ein erhöhtes Aufkommen im Personenverkehr reagieren und gleichzeitig eine effizientere Raumnutzung ermöglichen. Zudem sorgt auch das Thema Nachhaltigkeit, welches angesichts des sich verschärfenden Klimawandels sowohl von gesellschaftlicher als auch von politischer Seite zunehmend an Bedeutung gewinnt, für Veränderungen. Die hohe Umweltbelastung der gegenwärtigen Mobilität stellt für die Erreichung der Klimaziele ein großes Problem dar. Zukünftige Mobilitätsformen sollten also im Vergleich zur traditionellen Mobilität nachhaltiger sein und sich durch Minderungen von Emissionen sowie Ressourcenverbrauch auszeichnen. Neben gesellschaftlichen Aspekten führen auch technologische Innovationen zu Veränderungen im Bereich der Mobilität. Hier ist vor allem die fortschreitende Digitalisierung zu nennen. Das Smartphone als grundlegende technische Innovation des letzten Jahrzehnts ermöglicht eine einfache Nutzung neuer Plattformen. Diese Plattformen können von Anbietern neuartiger Mobilitätsdienste als zentrale Schnittstelle zu den Kunden aufgebaut werden. Gleichzeitig können wichtige Informationen wie der Standort des Kunden oder das Aufkommen von Nachfrage für Mobilität erfasst werden.

Im Bereich der Shared Mobility Systeme gibt es viele neuartige Mobilitätsformen, und meistens ist die Verwendung der Begrifflichkeiten nicht immer genau (vgl. Zwiers et al. (2021, S. 52)). Deshalb soll nun eine eindeutige Festlegung auf die in dieser Arbeit betrachtete Mobilitätsform und die untersuchte Problemstellung erfolgen. Im Rahmen dieser Masterarbeit soll sich mit dem Konzept des Ridesellings als zentrale Mobilitätsform beschäftigt werden. Unter diesem Begriff versteht man „bedarfsorientierte, d.h. nachfrageabhängige (on demand) und in der Regel gewerblich ausgerichtete Pkw-Fahrdienstleistungen für Fahrten „von Tür zur Tür“, die unter das Personen-beförderungsgesetz (PBefG) fallen“ (Zwiers et al. (2021, S. 53)). Dem Anbieter des Mobilitätsservices wird hierbei ein kommerzielles

Interesse, also eine Gewinnerzielungsabsicht, unterstellt. Das bedeutet, dass Fahrten nur dann tatsächlich stattfinden, falls eine entsprechende Nachfrage vorliegt. Der Anbieter stellt hierbei in der Regel den Fahrer und die Fahrzeuge zur Verfügung, und die Kunden teilen dem Serviceanbieter über eine App mit, zu welcher Uhrzeit sie von ihrem aktuellen Standort zu ihrem gewünschten Zielort befördert werden möchten. Bei einem On-Demand Rideselling System treffen die Kundenanfragen live ein, also während die Fahrten durchgeführt werden. Die Anfragen sind somit nicht zwingend im Vorhinein bekannt und haben deshalb eine stochastische Komponente. Von der reinen Definition her scheint das Geschäftsmodell des Ridesellings dem klassischen Taxi sehr ähnlich zu sein. Ein wesentlicher Unterschied besteht jedoch darin, dass beim Rideselling die Möglichkeit des Poolings von Anfragen besteht, was in Fachsprache auch Ridepooling genannt wird. Unter dem Begriff des Poolings versteht man das Bündeln von Kundenanfragen, wodurch mehrere Anfragen von einem Fahrzeug gleichzeitig bedient werden können. Gepoolte Kunden sollen gemeinsam anstatt einzeln befördert werden, für die Befriedigung dieser Kunden kann folglich auf den Einsatz weiterer Fahrzeuge verzichtet werden. Neben einer effizienteren Ausnutzung der Fahrzeugkapazität bewirkt dies auch eine Reduzierung der Kosten, da weniger Fahrzeuge eingesetzt werden müssen und zusätzlicher Fahrweg eingespart bzw. reduziert werden kann (vgl. Zwiers et al. (2021, S. 53)).

Inwieweit das Pooling von Anfragen gelingen kann, ist letztendlich auch vom verwendeten mathematischen Modell abhängig. Das zentrale Modell dieser Masterarbeit ist die von Haferkamp und Ehmke (2020) vorgestellte Herangehensweise an die Entscheidungsfindung in Ridesharing-Services. Haferkamp und Ehmke zerlegen den Entscheidungsprozess eines Ridesharing-Anbieters in zwei Teilprobleme. Zum einen muss entschieden werden, ob eine eingehende Nachfrage angenommen wird, zum anderen muss im Anschluss die angenommene Anfrage sinnvoll und möglichst effizient in eine bestehende Tour eingefügt werden. Die Schwierigkeit besteht darin, dass die Kundenanfragen in Echtzeit eintreffen und der Kunde eine schnelle Antwort haben möchte, ob sich ihm eine Mitfahrgelegenheit anbietet oder nicht. Im Sinne der Kundenzufriedenheit muss folglich schnell entschieden werden, ob ein Kunde in eine zur selben Zeit bereits ausgeführte Tour mit aufgenommen wird. In Rahmen dieser zwei Entscheidungsprobleme kann es dabei vorteilhaft sein, anstatt einem myopischen Ansatz einen antizipatorischen Ansatz zu verwenden. Im Gegensatz zu einem myopischen Ansatz, bei dem eine Kundenanfrage unweigerlich akzeptiert wird, falls der Kunde in eine laufende Tour integriert werden kann, ist bei einem antizipatorischen Ansatz die Annahmeentscheidung nicht nur ausschließlich vom aktuellen Tourenplan abhängig,

sondern auch von zukünftig erwarteten Anfragen. Falls es als sinnvoll erachtet wird, kann bei einer antizipatorische Entscheidungsfindung eine vorliegende Anfrage auch abgelehnt werden (vgl. Haferkamp und Ehmke (2020, S. 6)).

Ein antizipatorisches Modell wird der dynamischen Struktur eines Ride-Selling-Systems auf dem ersten Blick deutlich besser gerecht als der myopische Ansatz und bietet dadurch vielversprechende Vorteile. Im Rahmen dieser Masterarbeit wird daher auf antizipatorische Ansätze in der Entscheidungsfindung für eine Problemstellung aus dem Bereich der Shared Mobility Systeme eingegangen. Ziel dieser Arbeit ist es, einen Einblick zu geben, wie antizipatorische Ansätze beim Lösen einer beispielhaften Modellinstanz angewendet werden können. Hierfür soll neben dem Aufzeigen unterschiedlicher Herangehensweisen in der Literatur der Ansatz von Haferkamp und Ehmke (2020) im Fokus stehen. Dieser zentrale Ansatz soll anhand von eigens kreierten Fallbeispielen implementiert und evaluiert werden.

Die vorliegende Masterarbeit gibt in Kapitel 2 einen Überblick über bisherige Beiträge zur Problemstellung aus der Forschung. Im Rahmen eines kleinen Literaturüberblicks sollen hier unterschiedliche Methoden für eine antizipatorische Entscheidungsfindung aufgezeigt und kurz beschrieben werden. In Kapitel 3 folgt die Modellierung dynamisch-stochastischer Optimierungsprobleme. Hierzu soll das von Haferkamp und Ehmke (2020) vorgestellte Modell als eine beispielhafte Herangehensweise an antizipatorische Entscheidungsprobleme didaktisch aufbereitet werden. Kapitel 4 beschäftigt sich mit den konkreten antizipatorischen Lösungsverfahren des im vorherigen Kapitel vorgestellten zentralen Modells dieser Arbeit. Dafür sollen die unterschiedlichen Lösungsansätze anhand einer eigens erstellten kleinen Beispielinstanz mithilfe der Programmiersprache Python modelliert und anschließend gelöst werden. Anschließend soll auf die Ergebnisse der Beispielumsetzungen eingegangen werden. Hierzu soll neben einem Vergleich der Ergebnisse auch auf den Mehrwert und die Risiken, die aus der Verwendung antizipativer Strategien resultieren, eingegangen werden. Kapitel 5 rundet mit Schlussfolgerungen und einem Ausblick die Arbeit ab.

2 Literaturüberblick

Im folgenden Kapitel soll ein keiner Überblick gegeben werden, wie das Thema der antizipatorischen Entscheidungsfindung in der Fachliteratur behandelt wird. Hierzu lässt sich generell sagen, dass die Verwendung von Antizipation im Bereich der Tourenplanung zunehmend populärer wird. So lassen sich mittlerweile auch vor allem im Bereich des klassischen Vehicle Routing Problems zahlreiche Lösungsmethoden finden, welche eine antizipatorische Strategie beinhalten. Im Rahmen dieses Literaturüberblicks soll der Fokus jedoch ausschließlich auf Problemstellungen des Dial-a-Ride-Problems gelegt werden. Dieser Überblick hat dabei keinen Anspruch auf Vollständigkeit. Vielmehr sollen weitere Lösungsmethoden gezeigt werden, in welcher Form die Verwendung einer antizipatorischen Entscheidungsfindung in Shared Mobility Systemen erfolgen kann.

Horn (2002) versuchte mit dem aktiven Ablehnen von ungünstigen Kundenanfragen als Erster, antizipatorische Ansätze bei der Lösung von DDARP's einzubringen. Nachdem eine erste Implementierung für jede Anfrage durchgeführt wurde, werden im Anschluss verschiedene lösungsverbessernde Prozeduren durchgeführt. Eine davon ist die Rank-Homing-Heuristik, eine Methode, in welcher zukünftige Nachfragemuster in den Lösungsprozess miteingebunden werden. Hierbei werden inaktive Fahrzeuge in Bereiche geschickt, in denen in zukünftigen Perioden eine hohe Nachfrage vermutet wird. Jedem dieser Bereiche wird jeweils ein Knoten zugeordnet. Als Nächstes bestimmt die Heuristik anhand der Anzahl der geschätzten Anfragen für jeden Knoten, zu welchem Knoten das leere Fahrzeug geschickt werden soll. Da jedoch nicht garantiert werden kann, dass eine antizipierte Anfrage tatsächlich eintrifft, besteht die Gefahr, dass der bereits zurückgelegte Weg eines Fahrzeugs zu einem Knoten mit antizipierter Nachfrage sich als unproduktiv erweist. Deswegen müssen bereits bei der Entscheidung, ein Fahrzeug zu einem Knoten provisorisch loszuschicken, die anfallenden Kosten mit dem geschätzten Ertrag durch die Bedienung der Anfrage abgewogen werden.

Ichoua et al. (2006) fokussieren sich auf eine antizipative Tourenplanungsstrategie und beziehen bei der Erstellung der Fahrzeugrouten Informationen über zukünftige Nachfrage ein. Genauer gesagt werden zur Abbildung zukünftiger Nachfragen Dummy-Variablen verwendet. Eine Dummy-Variable bildet hierbei einen Dummy-Kunden ab, für den in Zukunft eine Nachfrage vorhergesagt wird. Mithilfe einer Tabu-Search-Heuristik wird ein erster Tourenplan auserkoren, der im Anschluss durch nicht zugeordnete Kundenanfragen ergänzt wird. Mithilfe einer Vehicle-Waiting-Heuristik wird bezweckt, dass ein Fahrzeug nach dem

Erfüllen seines aktuellen Auftrages an seiner derzeitigen Position wartet, falls die Wahrscheinlichkeit ausreichend hoch ist, dass in der geografischen Nachbarschaft des Fahrzeugs in naher Zukunft eine neue Anfrage eintreffen wird. Falls bestimmt wird, dass das Fahrzeug an seiner aktuellen Position warten soll, wird ein Dummy-Kunde generiert, um eine gewisse Wartezeit zu erzwingen. Somit kann das Fahrzeug auf die in seiner Nähe vermuteten eingehenden Aufträge reagieren, falls diese tatsächlich eintreffen. Die letztendliche Entscheidung, ob eine Anfrage angenommen oder abgelehnt wird, ist in dieser Methode von der antizipierten Nachfrage abhängig. Falls diese in den jeweiligen Gebieten nämlich nicht ausreichend hoch ist, wird das entsprechende Gebiet nicht oder nicht ausreichend mit Fahrzeugen abgedeckt, was dazu führt, dass eventuell auftretende Anfragen nicht bedient werden können und somit abgelehnt werden müssen.

Xiang et al. (2008) entwerfen ein flexibles Ablaufschema, um den Herausforderungen eines DDARP gerecht zu werden. Solche Herausforderungen stochastischer Natur sind zum Beispiel Veränderungen der Reisezeiten, das Eintreffen neuer Anfragen, das Nichtantreten von Kunden zu Fahrtbeginn, Fahrzeugausfälle, Stornierungen von Anfragen und Verkehrsstau. Um beim Eintreten solcher stochastischen Events entsprechend reagieren zu können, wird eine schnelle Heuristik vorgestellt, welche das Ablaufschema re-optimiert. Diese Heuristik verwendet auf der Ebene der Akzeptanzentscheidung eine Insertion-Heuristik, welche ungünstige Kundenanfragen ablehnt. Die Betrachtung einer Anfrage als ungünstig hängt davon ab, ob die inkrementellen Kosten der Anfrage eine bestimmte Grenze überschreiten. Die zukünftig zu erwartende Nachfrage spielt hierbei also noch keine Rolle. Die darauffolgende Routing-Entscheidung geschieht durch eine lokale Suche und eine zusätzliche Zielfunktion, um bei der Lösungssuche nicht in lokalen Optima festzuhängen. Durch verschiedene Simulationen wird die Performance der Lösungsmethode und der Einfluss der unterschiedlichen stochastischen Events gemessen. Dabei ist festzustellen, dass diese Heuristik allgemein sehr gute Ablaufschemen generieren kann und dabei auch gut mit dem Eintreten stochastischer Events zurechtkommt.

Schilde et al. (2011) untersuchen, ob die Nutzung von Vergangenheitsdaten einen positiven Einfluss auf die Lösungsqualität der Tourenplanung haben. Das hier untersuchte dynamisch-stochastische Dial-a-Ride-Problem bezieht sich auf den Transport von Patienten zum Krankenhaus beziehungsweise den Rücktransport dieser Patienten nach Hause. Die historischen Daten liefern Informationen über den Rücktransport von Patienten aus dem Krankenhaus und können genutzt werden, um die Wahrscheinlichkeit des Auftretens eines Rücktransportes vorhersagen zu können. Die Frage ist hier also, ob und in welchem Ausmaß

Informationen über zukünftige Rücktransporte ausgenutzt werden sollen. Zu beachten ist jedoch, dass es im Sinne der Kundenzufriedenheit eines Krankentransport-Services nicht erlaubt ist, einkommende Anfragen abzulehnen. Das heißt, dass in dieser Methode zwar eine antizipative Strategie bei der Tourenplanung verfolgt wird, jedoch ist es hier nicht möglich, dass gewisse Kundenanfragen aktiv abgelehnt werden können, falls sie als ungünstig erachtet werden. Zur Lösung der Problemstellungen werden insgesamt 4 verschiedene Metaheuristiken verwendet. Neben den myopischen Methoden Dynamic Variable-Neighborhood-Search und Multiple-Plan-Approach stehen die beiden antizipativen Methoden Dynamic-Stochastic Variable-Neighborhood-Search und Multiple-Scenario-Approach im Fokus. Der Grund für die Ähnlichkeit der verwendeten Methoden besteht darin, dass die Untersuchungsergebnisse ausschließlich vom Umgang mit Zukunftsinformationen abhängig sein sollen und nicht etwa von konzeptionellen Unterschieden zwischen den einzelnen Methoden. Während bei den beiden myopischen Ansätzen die stochastischen Anfragen als dynamische Anfragen behandelt werden, wird bei den antizipatorischen Ansätzen die stochastische Information über zukünftige Anfragen ausgenutzt.

Hosni et al. (2014) modellieren das Shared-Taxi-Problem als Mixed Integer Program. Mithilfe einer Lagrange-Zerlegung wird das Problem in kleine Teilprobleme zerlegt und diese werden dann jeweils unabhängig voneinander gelöst. Neben einer Lagrange-Heuristik wird zudem noch eine weitere Heuristik verwendet, welche auf dem Prinzip der Inkrementkosten basiert. Außerdem wird ausdrücklich darauf hingewiesen, dass Kundenanfragen abgelehnt werden dürfen, falls sie als unprofitabel erachtet werden. Allerdings spielt die erwartete zukünftige Nachfrage bei der Beurteilung des Wertes von Kundenanfragen hier keine Rolle. Der Wert einer Anfrage wird vielmehr aus dessen inkrementellen Kosten bestimmt, also aus den Kosten, die zusätzlich anfallen, falls eine einkommende Anfrage angenommen wird und dieser neue Kunde folglich in den Tourenplan eines Fahrzeugs aufgenommen wird. Falls die Inkrementkosten für die Bedienung einer Kundenanfrage niedriger sind als der erwartete Ertrag aus der Erfüllung dieser Anfrage, wird die Anfrage dieses Kunden akzeptiert. Gleichzeitig wird jeder Kunde dem Fahrzeug mit den niedrigsten Inkrementkosten zugeordnet. Für ein jedes Fahrzeug des Fuhrparks wird somit ein Kostenminimierungsansatz verfolgt. Alternativ zur Ablehnung einer Anfrage kann dem Kunden ein höherer Preis angeboten werden, welcher die inkrementellen Kosten einer Bedienung dieses Kunden übersteigt.

Alonso-Mora et al. (2017) präsentieren eine Methode, die durch das Verwenden von Vergangenheitsdaten die Performance eines Netzwerkes aus autonomen Fahrzeugen verbessern soll. Eine Optimierungsmethode soll hierbei die Kundenanfragen den Fahrzeugen zuordnen

und gleichzeitig die erwarteten Kosten minimieren. Basierend auf den Vergangenheitsdaten werden Wahrscheinlichkeitsfunktionen erstellt, die wiederum in den weiteren Methoden der Kundenzuordnung und der Tourenplanung genutzt werden. Genauer gesagt werden die in der Zukunft erwarteten Anfragen gebündelt und im Zuge einer jeden neuen Iteration in das Zuordnungs- und Tourenplanungsproblem mit aufgenommen. Innerhalb dieses Iterationsablaufs werden die derzeit vorliegenden Kundenanfragen und die antizipierten Anfragen mit den sich im Einsatz befindlichen Fahrzeugen paarweise miteinander verknüpft, falls die Nebenbedingungen und Kapazitätsbeschränkungen dies zulassen. Darauf aufbauend werden in einem weiteren Graphen alle zulässigen Touren mit den Fahrzeugen, die diese jeweiligen Touren unter Erfüllung aller Nebenbedingungen ausführen können, miteinander verbunden. Ein Integer Linear Program ermittelt aus diesem Graphen die bestmögliche Zuordnung von Fahrzeugen und zulässigen Touren miteinander, wodurch ein Tourenplan entsteht. Als letzter Schritt des Iterationsablaufs werden mithilfe eines Linear Programs die nicht zugeordneten Fahrzeuge, die sich derzeit nicht im Einsatz befinden, in Gebieten platziert, die einen Mangel an Fahrzeugen und ein hohes Aufkommen von Anfragen vorweisen. Der Iterationsablauf beginnt mit jeder weiteren Zuordnungsrunde wieder von neu.

Lois et al. (2018) untersuchen, ob die Operationskosten eines Online Demand Responsive Transportation Systems gesenkt werden können, wenn während des Optimierungsprozesses probabilistische Anfragen verwendet werden. Die grundlegende Idee dahinter ist, mithilfe der probabilistischen Anfragen tatsächlich eintreffende Anfragen vorhersagen zu können. Falls hinreichend genaue Vorhersagen getroffen werden können, kann die Fahrzeugflotte neu zugewiesen in einen besseren Zustand versetzt werden. Die grundlegende Annahme besagt, dass die Online-Abhängigkeit eines Transportsystems reduziert werden kann und eine bessere Lösung gefunden werden kann, falls genug historische Daten über vergangene Kundenanfragen verfügbar sind. Die vorgestellte Methode beginnt mit der Erstellung einer Wahrscheinlichkeitsverteilung der Nachfrage basierend auf realen historischen Datensätzen, um das Nachfrageverhalten des Transportsystems beschreiben zu können. Anschließend wird, während des Durchführungszeitraums, für jede eintreffende Anfrage ein Set aus zusätzlichen probabilistischen Anfragen erstellt und eine Initiallösung berechnet. Die probabilistischen Anfragen werden aus der im ersten Schritt erstellten Wahrscheinlichkeitsverteilung gewonnen. Im darauffolgenden Schritt werden die probabilistischen Anfragen aus der Lösung entfernt und die vorliegende Lösung weiter optimiert. Zuletzt werden die durch probabilistische Anfragen beeinflussten Lösungen mit den Lösungen, bei denen auf keine probabilistischen Anfragen zurückgegriffen wurde, verglichen. Die Studie zeigt, dass durch

die Verwendung probabilistischer Anfragen die Lösungen in Bezug auf die Zielfunktionskosten verbessert werden konnten. Gleichzeitig erhöht sich jedoch die Anzahl der eingesetzten Fahrzeuge, was vor allem für Anbieter, die eine eigene Fahrzeugflotte verwenden, problematisch sein kann. Zudem wurde nicht hinreichend genau untersucht, welchen Effekt eine Veränderung der Länge der Zeithorizonte oder die Verwendung dynamischer anstatt fixer Zeithorizonte mit sich bringen. Denkbar ist auch, den vorgestellten probabilistischen Ansatz in weiteren Optimierungsalgorithmen zu verwenden. Beispielsweise eignen sich hierfür Algorithmen, die Lösungen in einer großen Nachbarschaft suchen, wie die Tabu-Suche oder das Simulated Annealing. Deshalb sind in diesem Bereich noch weitere zukünftige Untersuchungen notwendig.

Wei et al. (2018) stellen eine Reinforcement Learning-Methode für ein Shared-Taxi-System vor. Der Hauptfokus liegt dabei nicht auf dem gewöhnlichem Shared-Taxi-Problem, in dem neue Kundenanfragen in den Tourenplan so integriert werden, dass die totale Wartezeit und die gesamte Umweg-Distanz minimiert wird. Vielmehr zielen die Autoren darauf ab, den Level-of-Service zu verbessern. Der Level-of-Service wird definiert als die totale Zahlungsbereitschaft der Kunden minus der totalen Strafkosten für Wartezeiten der Kunden und für gefahrene Umwege der Fahrzeuge. Falls ein Anbieter von Shared-Taxi-Systemen sich bei dem Einfügen einer neuen Kundenanfrage in den Tourenplan nur auf die Minimierung der Strafkosten für Wartezeiten und Umwege fokussiert, besteht die Gefahr, dass die Gesamtkapazität dieses Systems für eine gewisse Zeit ausgeschöpft ist, da das Problem in dieser Form nur myopisch betrachtet wird und somit zukünftig eintreffende Anfragen nicht ausreichend bedient werden können. Dies könnte für Anfragen in der Zukunft zu einer hohen Ablehnungsrate führen, was sich wiederum negativ auf den Level-of-Service auswirkt. Falls wiederum eine reine Fokussierung auf die Operationskosten, also der für den Serviceanbieter anfallenden Kosten für die insgesamt zurückgelegte Wegstrecke, erfolgt, resultiert dies in längeren Wartezeiten und Zusatzwegen für die Kunden, die sich bereits im System befinden. Unter dem Umstand der unsicheren Nachfrage in der Zukunft ist es schwierig, eine Zielfunktion zu finden, welche neue Kundenanfragen in das System integriert und dabei gleichzeitig den globalen Level-of-Service optimiert. Die von Wei et al. vorgestellte Reinforcement Learning-Methode beachtet die Unsicherheit zukünftiger Nachfragen und verbessert durch eine Look-Ahead-Insertion-Policy auch den globalen Level-of-Service eines Shared-Taxi-Systems. Die Kundenanfragen werden hierbei unter der Einhaltung einer Balance zwischen den Operationskosten und den Strafkosten für Wartezeiten und Umwege in das System integriert. Diese Balance wird durch einen anpassbaren Parameter gesteuert.

Der Wert des Balance-Parameters wird mithilfe eines Reinforcement Learning-Ansatzes in Abhängigkeit vom Zustand des Systems bestimmt. Die Autoren schlagen hierbei vor, die totale Besetzungszeit der Fahrzeuge als Beschreibung des Systemzustands zu verwenden, da diese Herangehensweise in der Praxis einfach anwendbar ist und die aktuelle Verfügbarkeit des Systems effektiv widerspiegelt. Die abschließenden Testergebnisse zeigen, dass die vorgeschlagene Methode von Wei et al. die Ablehnungsrate von Kundenanfragen reduzieren kann und gleichzeitig den globalen Level-of-Service eines Shared-Taxi-Problems verbessern kann.

Guo et al. (2020) zeigen ein Real-Time Service Vehicle Dispatching Framework für Online Ride Hailing Plattformen, welches auf die Hauptprobleme eines DDARP Rücksicht nimmt. Eine solche Herausforderung ist beispielsweise, eine gute Balance zwischen Angebot und Nachfrage zu finden, bei der nicht allzu viele Kundenanfragen unbedient bleiben. Deshalb soll verhindert werden, dass Fahrzeuge sich in Regionen bewegen, in denen wenig Nachfrage aufkommt. Die Fahrzeuge sollen also in Regionen gelotst werden, in denen viel Nachfrage prognostiziert wird. Um zu verhindern, dass die Fahrzeuge auf ihrem Weg in die Regionen mit hoher Nachfrage durch das Aufkommen neuer Anfragen abgelenkt werden, bedarf es einer guten Führung. Hierzu wird ein neues mathematisches Modell eingeführt, welches mithilfe einer räumlich-zeitlichen Wärme den Versendungsprozess der Fahrzeuge unterstützt. Das Konzept der räumlich-zeitlichen Wärme wird genutzt, um die Dichte der Nachfrage in einer Region innerhalb eines Zeitintervalls abzubilden. Die räumlich-zeitliche Wärme ist von vielfachen mehrdimensionalen Faktoren abhängig und kann deshalb nicht im Voraus bestimmt werden. Deshalb ist es wichtig, dass sie möglichst genau vorausgesagt werden kann. Hierfür schlagen die Autoren eine effektive Machine Learning Methode vor, mit der durch das Verwenden von historischen Daten die räumlich-zeitliche Wärme geschätzt werden kann. Grundlegende Faktoren räumlicher Dimension, die die Nachfrage einer Region beeinflussen, sind beispielsweise die Anzahl an Krankenhäusern, Bahnhöfen oder Flughäfen, die sich in der entsprechenden Region befinden. Auch Faktoren zeitlicher und meteorologischer Dimensionen spielen hier eine Rolle, beispielsweise der aktuelle Wochentag, Ferien und Feiertage oder das Wetter. Anschließend werden Entscheidungsregeln verwendet, um die Länge der Zeitfenster dynamisch festlegen zu können, und ein Kuhn-Munkres-basierter Matching-Algorithmus entwickelt, um die Anfragen eines Zeitfensters optimal zu den eingesetzten Fahrzeugen zuordnen zu können. Die Entscheidungsregeln müssen dabei ein gutes Gleichgewicht zwischen der Größe eines Zeitfensters und der für den anschließenden Prozess benötigten Berechnungszeit schaffen, damit die finale

Zuordnungslösung einen gut durchführbaren Zuordnungsplan darstellt. Die abschließenden Experimente zeigen, dass Prinzip der räumlich-zeitlichen Wärme die Balance zwischen Angebot und Nachfrage signifikant verbessern kann. Die Reallokation der Fahrzeuge funktioniert im Rahmen des vorgestellten Frameworks ebenfalls problemlos, sowohl für sich im Einsatz befindliche als auch untätige Fahrzeuge. Die Autoren geben an, dass in weiteren Untersuchungen versucht werden soll, die Effizienz bei der Berechnungszeit zu erhöhen, bei der Bestimmung der optimalen Zeitfenster die Betrachtung von einem individuellen Zeitfenster zu mehreren aufeinanderfolgenden Zeitfenstern auszuweiten, und den Ansatz in weiteren Anwendungsbereichen des Transportproblems zu verwenden.

Haferkamp und Ehmke (2020) betrachten, inwieweit Ride-Sharing-Systeme von einer antizipatorischen Entscheidungsfindung profitieren können. Hierbei unterteilen sie den kompletten Entscheidungsprozess eines Service-Anbieters in zwei Teilbereiche, nämlich die Annahmeentscheidung der Anfrage und den Routing-Prozess des Dienstleistungsfahrzeugs. Diese Betrachtung ermöglicht, in beiden Teilbereichen jeweils entscheiden zu können, ob eine antizipatorische Strategie verwendet werden soll oder nicht. Man betrachtet also verschiedene Zwischenstufen, mit denen die Intensität der Verwendung von Antizipation festgelegt werden kann, von myopisch über teils-antizipativ bis hin zu vollkommen antizipatorisch. Wie die Autoren zeigen, bringt die Verwendung von Antizipation generell einen Vorteil gegenüber einer myopischen Herangehensweise. Beachtenswert ist hierbei jedoch, dass eine vollständige Antizipation nicht automatisch immer die beste Wahl darstellt, sondern auch ein Blick auf eine teils-antizipative Entscheidungsstrategie vorteilhaft sein kann. Die bevorzugte Intensität der Antizipationsverwendung ist also von der spezifischen Service-Struktur eines Ride-Sharing-Anbieters abhängig und sollte für jeden Anbieter individuell bestimmt werden.

3 Vorstellung eines dynamisch-stochastischen Optimierungsproblems

In diesem Kapitel soll ein dynamisch-stochastisches Optimierungsproblem eingeführt werden. Hierfür soll der Modellierungsansatz von Haferkamp und Ehmke (2020) verwendet werden. Im ersten Teil des Kapitels sollen zunächst die Grundlagen dieses Modells eingeführt werden. Anschließend sollen Verfahren gezeigt werden, mit denen solch ein Optimierungsproblem gelöst werden kann.

3.1 Grundlagen der Modellierung

Zunächst sollen die für die Modellierung benötigten Grundlagen erläutert werden. Hierfür wird zuerst das Dynamic Dial-a-Ride-Problem gezeigt, anschließend soll der Markov Decision Process erklärt werden. Zuletzt wird das grundlegende Prinzip einer antizipatorischen Entscheidungsfindung erklärt.

3.1.1 Dynamic Dial-a-Ride-Problem

Das DDARP setzt sich aus Indizes und Parametern zusammen, die nun aufgezählt werden sollen. Die Menge $\mathcal{L}$ bildet alle geografischen Orte ab, die sich im Service-Gebiet eines Ride-Sharing-Dienstleisters befinden. An jedem Ort dieser Menge $l \in \mathcal{L}$ können Kunden zu- bzw. aussteigen. Für diesen Prozess wird eine deterministische Service-Zeit p_r festgelegt, der die Dauer dieses Zu- bzw. Aussteigevorgangs bemisst. Die eingesetzten Fahrzeuge bewegen sich zwischen den Orten der Menge $\mathcal{L}$. Für alle möglichen Paare von Orten $(i, j) \in \mathcal{L}$ wird eine deterministische Fahrzeit $c_{i,j}$ bestimmt. Die auftretende Nachfrage wird in Form von eintreffenden Kundenanfragen $r \in \mathcal{R}$ abgebildet. Eine jede einzelne Anfrage wird durch kundenindividuelle Daten charakterisiert. So bestimmt t_r die Zeit, zu der die Kundenanfrage eintrifft. Der Abholort $o_r \in \mathcal{L}$ ist der Standort, an dem der Kunde von einem der Fahrzeuge abgeholt werden muss. Der Zielort $d_r \in \mathcal{L}$ stellt den Ort dar, an den der Kunde befördert werden möchte. An diesem Ort muss das Fahrzeug den Kunden folglich abladen. Das Zeitfenster $[b_{r,} e_r]$ eines Kunden setzt sich aus der frühesten Abholzeit b_r und der spätesten Absetzzeit e_r dieses Kunden zusammen. Es wird angenommen, dass der früheste Abholzeitpunkt b_r gleich dem Eintreffzeitpunkt t_r der Anfrage ist. Das bedeutet, dass ein Kunde r für die Abreise bereit sein muss, sobald er die Anfrage für eine Beförderung stellt. Reservierungen oder Vorbuchungen sind somit ausgeschlossen, es treffen also nur Anfragen in Echtzeit

ein. Die späteste Absetzzeit e_r setzt sich aus der Summe der frühesten Abholzeit b_r, der direkten Fahrtzeit vom Abholort c_{o_r,d_r} zum Zielort eines Kunden, und einem Parameter α zusammen. Der Parameter α bildet die maximale Verspätung ab, die von den Kunden für die Ankunft am gewünschten Zielort toleriert wird. Verspätungen für Kunden entstehen durch Wartezeiten bei der Mitnahme am Abholort und durch das Fahren von Umwegen durch Bundling. Um die auftretenden Anfragen zu bedienen, wird eine Flotte identischer Fahrzeuge $\mathcal{V}$ eingesetzt. Wichtig ist zu beachten, dass die Kapazität eines Fahrzeuges in diesem Modell keine Rolle spielt, da ein Fahrzeug durch das Festlegen enger Bedienzeitfenster niemals voll besetzt sein wird (vgl. Haferkamp und Ehmke (2020, S. 4f)).

3.1.2 Markov Decision Process

Der MDP setzt sich aus einer Serie an Entscheidungsperioden $k \in \mathcal{K}$ zusammen, die den betrachteten Planungshorizont des DDARP abdecken. Zu Beginn des Planungshorizontes befindet sich der Service im Initialzustand s_0. Im Initialzustand befinden sich ein jedes Fahrzeug $v \in \mathcal{V}$ im untätigen Zustand an seiner Startposition $l_v \in \mathcal{L}$. Zusätzlich befinden sich zum Start noch keine Anfragen im System, alle Anfragen werden also im weiteren Verlauf innerhalb des Planungshorizontes eintreffen. Jede Entscheidungsperiode $k \in \mathcal{K}$ wird durch eine stochastisch eintreffende Anfrage $r_k \in \mathcal{R}$ ausgelöst, was zu einem Vorentscheidungszustand s_k führt. Im Vorentscheidungszustand werden alle Informationen, die für die anschließende Entscheidungsfindung relevant sind, abgebildet. Der Vorentscheidungszustand s_k wird definiert durch den Zeitpunkt t_r zu dem die neue Anfrage r_k beim Service-Anbieter eintrifft. Zudem enthält s_k den Zustand der Ressourcen, der durch das Tupel $(l_k^v, \mathcal{O}_k^v)$ abgebildet wird. Für jedes Fahrzeug $v \in \mathcal{V}$ stellen $l_k^v \in \mathcal{L}$ die aktuellen Positionen der Fahrzeuge und $\mathcal{O}_k^v \subset \mathcal{R}$ die Menge an akzeptierten Kundenanfragen dar, deren Bedienung aktuell ausgeführt wird. Zuletzt enthält s_k die Nachfrage, welche durch das Tupel $(r_k, \mathcal{U}_k)$ abgebildet wird. r_k entspricht der neu eingetroffenen Anfrage und $\mathcal{U}_k \subset \mathcal{R}$ stellt die Menge der akzeptierten Kundenanfragen dar, deren Bedienung noch aussteht. Die Definition des Vorentscheidungszustands s_k lautet folglich $s_k = (t_r, (l_k^v, \mathcal{O}_k^v), (r_k, \mathcal{U}_k))$.

Basierend auf dem Vorentscheidungszustand s_k wird eine Aktion $A^\pi(s_k)$ durch die Wahl einer Politik $\pi \in \Pi$ durchgeführt. Jede Aktion besteht aus zwei hierarchisch abhängigen Entscheidungen. Zuerst muss entschieden werden, ob die neue Anfrage r_k angenommen oder abgelehnt werden soll. Diese Akzeptanzentscheidung wird durch die binäre Entscheidungsvariable $x_k \in \{0, 1\}$ dargestellt, hier steht $x_k = 1$ für die Annahme und $x_k = 0$ für die

Ablehnung einer Anfrage. Innerhalb des Entscheidungsfindungsprozesses kontrolliert die Entscheidungsvariable x_k die Höhe der zu bedienenden Nachfrage. Die zweite zu treffende Entscheidung ist die Auswahl eines zulässigen Tourenplans. Das Routing setzt die zur Verfügung stehenden Ressourcen an Fahrzeugen ein, um die angefragten Fahrten effizient zu bedienen. Ein Tourenplan wird als zulässig erachtet, wenn alle im Vorfeld bereits akzeptierten Kundenanfragen einem Fahrzeug zugeordnet worden sind. Dabei gelten allerdings die im Folgenden aufgezählten Nebenbedingungen. Für alle akzeptierten Anfragen deren Bedienung noch aussteht $r \in \mathcal{U}_k$ und die neue Anfrage r_k gilt, dass die Abholung am Ursprungsort o_r vor der Absetzung am Zielort d_r für dasselbe Fahrzeug $v \in \mathcal{V}$ geplant werden muss, falls $x_k = 1$ ist. Des Weiteren gilt für alle aktuell ausgeführten Bedienungen von Anfragen $r \in \mathcal{O}_k^v$, dass die Absetzung des jeweiligen Kunden am Zielort d_r für dasselbe Fahrzeug $v \in \mathcal{V}$ geplant werden muss, in welchem der Kunde zugestiegen ist. Für alle Ursprungsorte gilt, dass der geplante Zusteigezeitpunkt z_o nicht vor dem frühesten Abholzeitpunkt b_r liegen darf. Für alle Zielorte gilt, dass der geplante Absetzzeitpunkt z_d nicht nach dem entsprechenden spätesten Absetzzeitpunkt e_r stattfinden darf.

Die Routingentscheidung wird durch die Entscheidungsvariable $y_k \in \mathcal{F}_x$ getroffen. $\mathcal{F}_x$ beinhaltet die endliche Menge an allen zulässigen Tourenplänen, die sich aus der vorrangigen Akzeptanzentscheidung x_k ergeben können. Die Akzeptanzentscheidung x_k muss auf eine solche Weise getroffen werden, dass es im Anschluss noch möglich ist, einen zulässigen Tourenplan zu finden. Die Menge $\mathcal{F}_x$ darf folglich nicht leer sein. Das Ausführen einer Aktion $A^\pi(S_k)$ löst einen deterministischen Übergang vom Vorentscheidungszustand s_k zu einem Zustand nach der Entscheidung $s_k^a = (y_k)$ aus. Der Zustand s_k^a enthält den letztendlich ausgewählten zulässigen Tourenplan y_k, welcher das konkrete Routing der Fahrzeuge bis zur nächsten Entscheidungsperiode $k + 1$ festlegt. Die nächste Entscheidungsperiode wird durch den stochastischen Übergang W_{k+1} ausgelöst, wenn der Serviceanbieter die nächste Anfrage $r_{k+1} \in \mathcal{R}$ empfangen hat.

Das Ziel bei dieser Methode ist, eine optimale Politik $\pi^* \in \Pi$ zu finden, die die erwartete kumulative Vergütung $v^\pi(s_0) = max^\pi \mathbb{E}\{\Sigma_{k=0}^K B_k(s_k, A^\pi(s_k), W_{k+1}|s_0\}$ über alle Entscheidungsperioden $k \in \mathcal{K}$ maximiert. Für die anteilige Vergütung B_k einer Entscheidungsperiode $k \in \mathcal{K}$ gilt, dass B_k denselben Wert wie die Entscheidungsvariable x_k annimmt. Die kumulative Vergütung $v^\pi(s_0)$ entspricht somit der Akzeptanzrate, welche sich aus der Division der Anzahl der eingetroffenen Anfragen durch die Menge der akzeptierten und somit erfüllten Kundenanfragen ergibt (vgl. Haferkamp und Ehmke (2020, S. 5)).

3.1.3 Einführung in die antizipatorische Entscheidungsfindung

Die zentrale Herausforderung des in diesem Kapitel formulierten Problems liegt in der Ungewissheit der zukünftigen Nachfrage. Diese Ungewissheit wird durch die stochastisch eintreffenden Nachfragen $r \in \mathcal{R}$ verursacht und hat zur Folge, dass die Entscheidungen der Entscheidungsperioden $k \in \mathcal{K}$ unter unvollständiger Informationslage getroffen werden müssen. Die Höhe der kumulativen Vergütung $v^{\pi}(s_0)$ ist somit ebenfalls unsicher. Um dieser Ungewissheit zu entgehen, können die zukünftige Nachfrage und deren Auswirkungen auf den Entscheidungsfindungsprozess antizipiert werden. Antizipation in Bezug auf den Entscheidungsfindungsprozess bedeutet, dass zukünftige Stochastizität, beispielsweise in Form von historischen Daten oder Prognosen, berücksichtigt wird, um die erwartete kumulative Vergütung $v^{\pi}(s_0)$ zu maximieren. Dem gegenüber steht die myopische Entscheidungsfindung, bei der für die Maximierung der kumulativen Vergütung ausschließlich bestätigte Information, also tatsächlich eingetroffene Anfragen, in Betracht gezogen werden soll. Die verschiedenen Ebenen, auf denen die Verwendung von Antizipation erfolgen kann, werden im Folgenden genauer erläutert.

Jede Entscheidungsperiode $k \in \mathcal{K}$ des zugrunde liegenden Problems beinhaltet sowohl die Akzeptanzentscheidung x_k als auch die Routingentscheidung y_k. Beide Entscheidungen können unabhängig voneinander entweder auf antizipatorische oder myopische Weise getroffen werden. Auf der Ebene der Akzeptanzentscheidung x_k gilt, dass bei einer myopischen Herangehensweise die vorliegende Anfrage akzeptiert wird, falls ein Fahrzeug verfügbar ist und somit noch Kapazität für die Beförderung dieses Kunden vorhanden ist, da diese Aktion die unmittelbare Vergütung um B_k erhöht. In diesem Fall basiert die Akzeptanz einer Anfrage ausschließlich auf den Routingentscheidungen, die in den vorherigen Perioden getroffen wurden, da diese die Kapazität in der aktuell vorliegenden Periode beeinflussen. Diese durch die vorhandenen Kapazitäten beeinflusste Akzeptanzentscheidung wird in der Literatur auch Zulässigkeitsprüfung genannt, bei der bestimmt wird, ob ein zulässiger Tourenplan gefunden werden kann. Die Prüfung der Zulässigkeit ist ebenfalls die Grundlage bei der antizipatorischen Herangehensweise, aber hier ist die Akzeptanzentscheidung komplexer, da die Möglichkeit besteht, eine vorliegende Anfrage unter Berücksichtigung potentiell eintreffender Anfragen in der Zukunft aktiv abzulehnen. Diese Ablehnung von Anfragen tritt auf, falls die erwartete Kapazitätseinsparung einer abgelehnten Anfrage zu einer Erhöhung der im Planungshorizont insgesamt angenommenen Anfragen führt, was die erwartete kumulative Vergütung $v^{\pi}(s_0)$ maximiert. Auf der Ebene der Routingentscheidung y_k gilt, dass

die myopische Herangehensweise dazu führt, dass letztendlich der Tourenplan ausgewählt wird, dem die geringste Kapazitätsnutzung unterliegt, also beispielsweise die Lösung eines DARP's, die aus einem Vorentscheidungszustand s_k resultiert. Eine antizipative Strategie im Routing erfordert die Berücksichtigung von zukünftiger Nachfrage im Routingprozess. Hier können sich die verschiedenen Ansätze im Umfang des Einflusses zukünftiger Nachfrage deutlich unterscheiden. Denkbar wären neben einfachen Methoden wie beispielsweise einer antizipatorischen Versetzung untätiger Fahrzeuge (vgl. Pureza & Laporte, 2008) auch anspruchsvolle Herangehensweisen, bei denen die zukünftige Nachfrage bei der Erfüllung von Anfragen aktiv berücksichtigt wird (vgl. Ferrucci et al., 2013).

Jede zur Verfügung stehende Politik $\pi \in \Pi$ kann einer der vier Antizipationsebenen zugeordnet werden. Nicht-antizipatorisch schließt alle Politiken ein, welche in jeglicher Form auf die Verwendung von Antizipation verzichten. Stattdessen erfolgen beide Entscheidungen, die Akzeptanz- und die Routingentscheidung, auf myopische Weise. Eine nicht-antizipatorische Herangehensweise wäre vorteilhaft, wenn beispielsweise die zukünftig auftretende Nachfrage unvorhersehbar ist oder eine unzureichende Datenlage von Vergangenheitsinformationen vorliegt. Außerdem könnte der höhere Rechenaufwand von antizipatorischen Modellen dazu führen, dass vollkommen myopische Politiken bevorzugt werden. Nichtsdestotrotz birgt die Verwendung solcher myopischen Politiken das Risiko, durch das Vorliegen unvollkommener Information unvorteilhafte Entscheidungen zu treffen.

Antizipatorische Akzeptanz und antizipatorisches Routing bezeichnen Politiken, welche Antizipation auf nur einer Entscheidungsebene verwenden. Genauer gesagt wird bei einer Politik auf der Ebene der antizipatorischen Akzeptanz die Verwendung von Antizipation ausschließlich bei der Akzeptanzentscheidung x_k erfolgen. Solche Politiken führen zu einer Verbesserung von Akzeptanzentscheidungen, indem der langfristige Wert einer Anfrage unter Berücksichtigung von zukünftiger Nachfrage bewertet wird. Dadurch, dass der Fokus auf die weniger komplexe Akzeptanzentscheidung x_k gelegt wird, reduziert diese Art von Politik den Herausforderungsaufwand einer antizipatorischen Entscheidungsfindung. Gleichwohl erfordern solche Politiken zuverlässige Information über zukünftige Nachfrage, um erfolgreich implementiert werden zu können. Der Fokus von antizipatorischen Akzeptanz-Politiken liegt auf der Akzeptanz der meistbevorzugten Anfragen, die eine relativ geringe Nutzung von Kapazität benötigen, also beispielsweise Anfragen, welche einfacher gebündelt werden können. Allerdings birgt eine jede antizipatorische Politik auch die Gefahr, dass unvorteilhafte Entscheidungen getroffen werden können. Beispielsweise könnte die zukünftige Nachfrage überschätzt werden, sodass das tatsächliche zukünftige

Nachfragevolumen kleiner ist als das Volumen, welches prognostiziert wurde. In Vorbereitung auf die zukünftige Nachfrage wird bei einer Überschätzung folglich zu viel Kapazität für zukünftige Entscheidungsperioden bereitgestellt, als für die tatsächliche zukünftige Nachfrage nötig wäre. Um diese Kapazitätseinsparungen bewerkstelligen zu können, werden in der Gegenwart möglicherweise zu viele Anfragen abgelehnt. Schlussendlich werden die verfügbaren Fahrzeugkapazitäten bei einer Überschätzung der zukünftigen Nachfrage nicht vollständig genutzt. Zusätzlich kann die Verwendung von antizipatorischen Akzeptanzentscheidungen zu Nachteilen in geschäftlichen Überlegungen führen. Als Beispiel sind hier Ablehnungen von Kundenanfragen zu nennen. Proaktive oder unverständliche Ablehnungen von Anfragen könnten dazu führen, dass die Kundenzufriedenheit sinkt. Des Weiteren führt das kontinuierliche Ablehnen gewisser Anfragen dazu, dass diese Anfragen auf lange Frist gesehen möglicherweise nicht mehr gestellt werden, selbst wenn diese Anfragen nach einer gewissen Zeit vom Serviceanbieter nicht mehr als unvorteilhaft angesehen werden.

Im Gegensatz zur Antizipatorischen Akzeptanz erfolgt beim Antizipatorischen Routing die Verwendung von Antizipation ausschließlich auf der Ebene der Routingentscheidung y_k. Der Kern bei der Verwendung Politiken solcher Art liegt in der effizienten gegenwärtigen und zukünftigen Nutzung der Fahrzeugkapazitäten, indem die zukünftige Nachfrage beim Routingprozess berücksichtigt wird. Das Antizipatorische Routing muss hierbei mit der Komplexität der Routingentscheidung zurechtkommen, diese ist jedoch bereits bei nichtantizipatorischen Modellen schon herausfordernd genug, insbesondere, falls große Modellinstanzen gelöst werden müssen. Um Antizipatorisches Routing ermöglichen zu können, bedarf eine solche Politik genaue räumliche und zeitliche Informationen über die zukünftige Nachfrage. Eine beispielhafte Politik könnte darin bestehen, untätige Fahrzeuge zu verlegen, um prognostizierter zukünftiger Nachfrage besser entgegnen zu können. Gleichzeitig können antizipierte Anfragen beim Bundling effizienter berücksichtigt werden. Jedoch können wieder auch in solchen Politiken Fehleinschätzungen zu einer ineffizienten Nutzung von Kapazitäten führen, indem beispielsweise Fahrzeuge zu antizipierten Anfragen zugeordnet werden, welche niemals realisiert werden. Aus geschäftlicher Sicht kann dies aufgrund von unnötigen Zusatzwegen oder längeren Wartezeiten zu Unzufriedenheit unter den Kunden führen.

Das Konzept der Vollkommenen Antizipation bezeichnet Politiken, welche sowohl bei der Akzeptanz- als auch bei der Routingentscheidung Antizipation verwenden. Diese Techniken stellen rechnerisch gesehen die größte Herausforderung unter den vorgestellten vier

Herangehensweisen dar, bieten jedoch auch das größte Potential im Bereich des Dynamic Fleet Managements (vgl. Haferkamp und Ehmke (2020, S. 6f)).

3.2 Lösungsverfahren

Dieser Abschnitt führt konkrete Lösungsverfahren ein, welche verwendet werden können, um den Einfluss der verschiedenen Antizipationsebenen auf das Dynamic Fleet Management in Ride-Selling-Dienstleistungen herauszustellen. Da die Umsetzung der im vorherigen Kapitel dargestellten Herangehensweisen sehr komplex ist, wird im Folgenden ein vereinfachtes System vorgestellt, mit dem die Wirkweise des Einsatzes von Antizipation auf verschiedenen Ebenen gezeigt werden soll. Nach der Schilderung des grundlegenden Prinzips antizipativer Lösungsansätze wird eine etablierte Large Neighborhood Search verwendet, um die unterschiedlichen antizipatorischen Ansätze abzubilden und miteinander vergleichbar machen zu können.

3.2.1 Antizipatorische Lösungsverfahren

Im Folgenden sollen unterschiedliche antizipative Lösungsansätze gezeigt werden. Für eine jede vorgestellte Antizipationsebene unterscheidet sich hierbei der Umfang der zur Verfügung stehenden Informationen für die Akzeptanzentscheidung x_k und die Routingentscheidung y_k.

Bei der Nicht-Antizipatorischen Ebene wird das Problem auf myopische Weise gelöst und diese dient somit als Orientierungspunkt für den vollkommenen Verzicht auf Antizipation. Genauer gesagt wird bei der entsprechenden Politik das DDARP durch eine Zulässigkeitsprüfung und anschließende Re-Optimierung gelöst. Die Zulässigkeitsprüfung der Akzeptanzentscheidung x_k erfolgt durch eine Insertion Heuristik, welche kontrolliert, ob eine eintreffende Anfrage $r_k \in \mathcal{R}$ in den obliegenden Tourenplan y_{k-1} eingefügt werden kann, wobei y_0 den initialen leeren Tourenplan darstellt. Falls das Einfügen gelingt, wird die Anfrage akzeptiert. Der aus dem erfolgreichen Einfüge-Schritt resultierende Tourenplan wird anschließend in Hinblick auf die Routingentscheidung y_k re-optimiert. Zu diesem Zweck wird ein statisches DARP für alle akzeptierten Anfragen gelöst. Die Zielfunktion besteht hierbei in der Minimierung der gesamten benötigten Reisezeit. Wichtig ist zu beachten, dass sowohl für die Akzeptanz- als auch für die Routingentscheidung gilt, dass bereits erfüllte Anfragen und die aktuell von den Fahrzeugen angefahrenen Orte nicht neu geplant werden können. Genauer gesagt bedeutet das, dass Fahrzeuge nicht von ihrem Weg zu einer

geografischen Position $l \in \mathcal{L}$ nicht abgelenkt werden dürfen. Dies verringert zwar die Flexibilität in der Planung des Prozesses, ermöglicht allerdings auch eine Verringerung des benötigten Rechenaufwandes. Darüber hinaus gewährleistet dies, dass die Fahrer und die Kunden bzw. Passagiere verlässlich über die nächste Haltestelle informiert werden können, ohne dass häufige Änderungen des nächsten Halts aufgrund von Ablenkungen zu anderen Orten auftreten.

Im Rahmen der Antizipatorischen Akzeptanz wird angenommen, dass für die Akzeptanzentscheidung x_k vollkommene Information über die zukünftige Nachfrage zur Verfügung steht. Diese Entscheidung wird für jede eintreffende Anfrage $r_k \in \mathcal{R}$ in zwei Schritten getätigt. Zuerst erfolgt, wie im Nicht-Antizipatorischen Ansatz, eine Zulässigkeitsprüfung durch eine Insertion Heuristik. Falls die Prüfung der Zulässigkeit erfolgreich ist, wird im zweiten Schritt die Vorteilhaftigkeit der Anfrage untersucht. Um vorteilhafte Anfragen ausfindig zu machen, wird ein statisches Team Orienteering Problem mit gleichen Wertungen für eine jede Anfrage gelöst. Das TOP ist eine bekannte Variante des statischen Tourenplanungsproblems, bei den nur die profitabelsten Orte besucht werden (vgl. Chao et al. (1996)). Das Ziel des TOP besteht darin, die optimale Menge der besuchten Orte zu finden, die den Gewinn des Dienstleistungsbetreibers maximieren. Als Grundlage für das TOP dienen alle Anfragen des obliegenden Tourenplans y_{k-1}, die aktuelle Anfrage r_k und alle in der Zukunft erwarteten Anfragen. Damit der final gefundene Tourenplan die Anzahl der insgesamt eingeplanten Anfragen maximiert, werden alle Anfragen als gleichwertig eingeschätzt. Alle Anfragen des obliegenden Tourenplans y_{k-1} müssen abgedeckt werden, dann identifiziert das TOP unter den aktuellen und allen zukünftigen Anfragen die als vorteilhaft erachteten Anfragen. Schlussendlich wird eine Anfrage r_k akzeptiert, wenn diese im besten gefundenen Tourenplan enthalten ist. Nachdem die Akzeptanzentscheidung durchgeführt wurde, wird ein neuer Tourenplan y_k bestimmt. Dies geschieht, wie auch schon aufseiten der Nicht-Antizipatorischen Ebene, indem ein statisches DARP ohne Berücksichtigung von Anfragen in der Zukunft gelöst wird.

Das Antizipatorische Routing folgt bei der Entscheidung über die Auftragsannahme x_k dem Nicht-Antizipatorischen Ansatz. Genauer gesagt wird die Entscheidung dynamisch durchgeführt, indem für eine jede eintreffende Anfrage $r_k \in \mathcal{R}$ eine Zulässigkeitsprüfung durch eine Insertion Heuristik erfolgt. Nichtsdestotrotz wird angenommen, dass alle Anfragezeitfenster $[b_r, e_r]$ sich auf die Erfüllung am darauffolgenden Tag beziehen, sodass der obliegende Tourenplan y_{k-1} noch flexibel umgestaltet werden kann. Das bedeutet, damit der Ansatz des Antizipatorischen Routings funktionieren kann, muss die Planung des

aktuellen Tages am vorherigen Tag im Voraus geschehen. Dies ermöglicht das Treffen einer dynamischen Akzeptanzentscheidung x_k ohne Information über zukünftige Nachfrage, während gleichzeitig die Routingentscheidung y_k mit vollkommener Information über alle eintreffenden Anfragen getätigt werden kann. Nachdem eine Zulässigkeitsprüfung erfolgreich durchgeführt wurde, erfolgt kein weiterer Re-Optimierungsschritt des Tourenplans. Eine finale Routingentscheidung y_k wird erst untersucht, sobald alle Entscheidungen aufseiten der Auftragsannahme vollbracht wurden. Der final auszuführende Tourenplan wird schlussendlich ermittelt, indem das aus der Menge der angenommenen Anfragen resultierende statische DARP gelöst wird.

Für den Vollkommen-Antizipatorischen Ansatz wird angenommen, dass perfekte Information über die zukünftige Nachfrage gegeben ist. Dies ermöglicht, dass alle dynamischen Entscheidungen im Voraus getroffen werden können. Zu diesem Zweck wird das Problem als statisches TOP gelöst, bei dem jede Anfrage $r \in \mathcal{R}$ als gleichwertig angesehen wird. Dies führt zu einem Tourenplan, der die Anzahl der angenommenen Anfragen auf eine solche Weise maximiert, dass die zu akzeptierenden Anfragen und die zu fahrenden Routen einander entsprechend optimiert werden können (vgl. Haferkamp und Ehmke (2020, S. 7f)).

3.2.2 Large Neighborhood Search

Nun soll ein LNS-Verfahren eingeführt werden, mit der die Effekte der Verwendung von Antizipation auf unterschiedlichen Entscheidungsebenen bewertet werden können. Um eine gewisse Vergleichbarkeit der Ergebnisse zu ermöglichen, wird beim Durchspielen der unterschiedlichen Ansätze stets die gleiche Heuristik verwendet. Die im Folgenden vorgestellte LNS basiert auf der Adaptive Large Neighborhood Search von Ropke und Pisinger (2006). Diese zeigte bereits in der Vergangenheit eine gute Anwendbarkeit bei einer Vielzahl komplexer Tourenplanungsprobleme und erzielte dabei durchweg gute Ergebnisse in Hinblick auf die Berechnungszeit. Speziell für die Auftragsannahmeentscheidung ist eine solche kurze Berechnungszeit von großer Bedeutung.

Das grundlegende Konzept einer LNS ist, gefundene Lösungen wiederholt zu zerstören und anschließend wieder zu reparieren. Für das zugrundeliegende DDARP besteht eine Lösung w aus einem Tourenplan n_w und einer Menge an derzeit nicht eingeplanten Kundenanfragen $m_w \in \mathcal{R}$, deren Bedienung aktuell nicht im Tourenplan n_w berücksichtigt wird. Ein Tourenplan n_w stellt den Plan für ein jedes Fahrzeug $v \in \mathcal{V}$ dar. Dieser Plan gibt die Reihenfolge der zu besuchenden Orte $l \in \mathcal{L}$ und die geplanten Ankunftszeiten z_l^v für alle

Fahrzeuge an. Die LNS zielt darauf ab, die Anzahl der geplanten Auftragserfüllungen $|n_w|$ zu maximieren oder die erforderliche gesamte Fahrzeit $c(n_w)$ zu minimieren.

Im Folgenden soll der von Haferkamp und Ehmke (2020) erstellte Pseudocode des LNS-Verfahrens skizziert werden, um das grundlegende Verständnis dieser Methodik vermitteln zu können.

Function $LNS(w_0)$

| $w = w_0$

| $w_{best} = w_0$

| ***while*** *termination criterion is not met* ***do***

| | $w_{new} = w$

| | remove requests from $n_{w_{new}}$ to $m_{w_{new}}$

| | insert requests from $m_{w_{new}}$ into $n_{w_{new}}$

| | **if** w_{new} is accepted **then**

| | | $w = w_{new}$

| | | **if** w_{new} is an improvement to w_{best} **then**

| | | | $w_{best} = w_{new}$

| | | **end**

| | **end**

| **end**

| **return** w_{best}

Im Initialzustand startet die LNS mit einer ersten Lösung w_0 als Input. Diese Lösung w_0 wird als die derzeit zugrunde liegende Lösung w und zugleich als die aktuell bestmögliche Lösung w_{best} abgespeichert. Im Anschluss wird eine iterative Suche nach einer besseren Lösung durchgeführt. Dies geschieht so lange, bis ein Stoppkriterium erreicht wurde. Als Stoppkriterium wird eine maximale Anzahl an Iterationen β festgelegt. Je nach Zweck der Suche können auch noch weitere Kriterien in Betracht gezogen werden.

Jede Iteration der LNS beginnt mit dem Erstellen einer neuen Lösung. Dieser Prozess findet innerhalb einer while-Schleife statt, diese wird erst unterbrochen, wenn ein Stoppkriterium erreicht wurde. Im Rahmen dieser Schleife wird die derzeit vorliegende Lösung w als neue

Lösung w_{new} gespeichert. Genauer gesagt wird versucht, eine neue Lösung auf Grundlage der aktuell vorliegenden Lösung zu finden.

Danach wird die neue Lösung w_{new} durch einen Destroy-Operator zerstört. Dieser Operator entfernt zwischen γ_1 und γ_2 Prozent der im Tourenplan $n_{w_{new}}$ eingeplanten Anfragen und ordnet sie der Menge der nicht eingeplanten Anfragen $m_{w_{new}}$ zu. Allerdings gilt für jeden Kunden, dessen Ursprungsort o_r bereits angefahren wurde, dass der entsprechende Zielort des Kunden d_r nicht mehr aus dem Tourenplan entfernt werden darf. Die Kunden, die also bisher in ein Fahrzeug bereits aufgenommen wurden, dürfen nicht mehr durch einen Destroy-Operator aus dem Tourenplan entfernt werden. Die exakte Anzahl der zu entfernenden Anfragen wird für jede Iteration individuell durch einen Zufallswert q bestimmt. Für q gilt $\{ q \in \mathbb{N} \mid (\gamma_1 \times |n_{w_{new}}|) \leq q \leq (\gamma_2 \times |n_{w_{new}}|) \}$.

Im anschließenden Schritt wird ein Repair-Operator verwendet, der so viele Anfragen wie möglich aus der Menge der nicht eingeplanten Anfragen $m_{w_{new}}$ nimmt und in den Tourenplan $n_{w_{new}}$ einfügt. Im Gegensatz zu einer klassischen ALNS werden bei dieser LNS die für den Destroy- und den Repair-Vorgang benötigten Operatoren für eine jede Iteration zufällig ausgewählt. Das ist deshalb notwendig, da die LNS im vorliegenden Fall in einer dynamischen Umgebung implementiert wird. In dieser werden vielfache Suchen über wenige Iterationen hinweg durchgeführt, sodass eine automatische Anpassung der Operatorauswahl während der Suche weder zulässig noch vorteilhaft ist.

Die Operatoren für die Entfernung von Kunden aus dem Tourenplan entsprechen denen der ALNS von Ropke und Pisinger (2006). Dafür stehen insgesamt 3 Operatoren zur Auswahl. Der Random-Removal-Operator wählt die zu entfernenden Anfragen nach dem Zufallsprinzip aus. Somit bietet er ein Maximum an Diversifikation hinsichtlich der durch ihn ausgewählten Menge an Anfragen, da schlussendlich alle Anfragen mit der gleichen Wahrscheinlichkeit ausgewählt werden können.

Der Worst-Removal-Operator versucht, schlecht platzierte Anfragen zu entfernen. Um diese schlechten Anfragen ausfindig zu machen, wird für eine jede im Tourenplan enthaltene Anfrage bestimmt, wieviel Fahrtzeit eingespart werden kann, falls diese Anfrage aus dem Tourenplan entfernt wird. Im Anschluss werden die potentiellen Zeitersparnisse in einer Liste festgehalten und absteigend sortiert. Um zu verhindern, dass immer die gleichen oder ähnliche Gruppen an Anfragen aus dem Tourenplan entfernt werden, wird bei der Auswahl der Anfragen das Prinzip der Verzerrung verwendet. Hierbei wird der Formulierung von Ropke und Pisinger (2006) gefolgt, bei der die Listenposition der nächsten zu entfernender

Anfrage mit der Formel $q_1^{\delta} \times |list|$ ermittelt wird. In dieser Formel steht q für einen zufälligen Wert, für den $\{q \in \mathbb{Q} \,|0 \leq q \leq 1\}$ gilt. Der Parameter δ_1 kontrolliert den Stärkegrad der Verzerrung.

Der Shaw-Removal-Operator, ursprünglich von Shaw (1998) eingeführt, entfernt Anfragen, die einander ähnlich sind. Diese können nämlich aufgrund ihrer Ähnlichkeit zueinander leichter vertauscht werden, wodurch das Finden von besseren Tourenplänen wahrscheinlicher wird. In einem ersten Schritt wird eine Anfrage hierfür zufällig ausgewählt. Alle anderen Anfragen werden dann in aufsteigender Reihenfolge nach ihrer Ähnlichkeit zur zufällig ausgewählten Anfrage sortiert und der Sortierung entsprechend entfernt. Die Ähnlichkeit zwischen zwei Anfragen r_1 und r_2 wird ermittelt, indem die Entfernung zwischen den jeweiligen Ursprüngen $c_{a_{r_1},a_{r_2}}$ und Zielorten $c_{d_{r_1},d_{r_2}}$ der beiden Anfragen gemessen wird. Zusätzlich wird, sowohl für die Ankunft am Ursprungs- als auch am Zielort der beiden Kunden, die Differenz der geplanten Ankunftszeiten $\triangle\left(z_{a_{r_1}}, z_{a_{r_2}}\right) + \triangle\left(z_{d_{r_1}}, z_{d_{r_2}}\right)$ ermittelt. Bevor die beiden geografischen und zeitlichen Werte addiert werden, werden sie min-max-normalisiert.

Für das anschließende Einfügen von entfernten Anfragen lässt sich sagen, dass es eine große Auswahl an Operatoren gibt. Im Rahmen dieses Modells soll jedoch nur der vielversprechende Regret-2-Operator behandelt werden. Dieser wird sowohl ohne Verzerrung als auch mit dem Prinzip der Verzerrung ausgeführt. Die Regret-Insertion-Heuristik wurde von Potvin und Rousseau (1993) für das Tourenplanungsproblem mit Zeitfenstern eingeführt. Mit dieser Heuristik soll herausgefunden werden, an welchen Positionen Anfragen in den Tourenplan wieder eingefügt werden sollen. Hierbei wird für eine Anfrage ermittelt, wie groß der Verlust wäre, falls die bestmögliche Einfügeposition dieser Anfrage nicht länger zulässig wäre. Das Ziel dieser Heuristik ist letztendlich, diejenigen Anfragen an ihre jeweiligen besten Positionen in den Tourenplan wieder einzufügen, die den größten Verlust aufweisen, falls dieses Einfügen nicht erfolgen würde. Für die Regret-2-Variante wird der Verlust ermittelt, indem die Differenz zwischen der aus Kostensicht betrachteten bestmöglichen und der zweitbestmöglichen Einfügeposition in den Tourenplan gebildet wird. Die Einfügepositionen müssen hierbei logischerweise stets zulässig sein. Die auftretenden Kosten entsprechen hierbei der zusätzlichen Fahrtzeit, die sich daraus ergeben würde, falls eine Anfrage an einer bestimmten Position in den Tourenplan eingegliedert werden würde. Es wird also versucht, für eine Anfrage diejenige Einfügeposition zu finden, die den geringsten Zuwachs an zusätzlicher Fahrtzeit mit sich bringt. Falls für eine Anfrage nur eine zulässige

Einfügeposition im Tourenplan gefunden werden kann, wird die Differenz zwischen der bestmöglichen Einfügeposition und dem maximal möglichen Integer-Wert gebildet. Dieser Integer-Wert ist von seiner Wirkweise her mit einem Big-M zu vergleichen, das heißt, es muss eine hinreichend große Zahl bestimmt werden. Bei der Auswahl, welche Anfrage als nächstes dem Tourenplan hinzugefügt werden soll, wird der Verlust-Wert einer jeden aktuell nicht eingeplanten Anfrage $r \in m_{w_{new}}$ berechnet, die Verlust-Werte werden dann in absteigender Reihenfolge in einer Liste sortiert. Der Regret-2-Operator wird in diesem Modell in zwei Varianten verwendet, eine mit und eine ohne dem Prinzip der Verzerrung. Bei letzterer Variante wird die Anfrage mit den größten Verlust-Wert an ihre zulässige kosten-effizienteste Position in den Tourenplan eingefügt. Der Regret-2-Operator hingegen wählt die nächste einzufügende Anfrage auf dieselbe Weise aus, wie der im Vorhinein bereits vorgestellte Worst-Removal-Operator. Der Grad der Verzerrung wird hierbei durch den Parameter β_2 bestimmt. Dieses Prinzip soll also, wie auch schon bei der Worst-Removal, verhindern, dass im Laufe der Iterationen immer wieder ausschließlich die gleichen Anfragen eingefügt werden. Beispielsweise wäre dies denkbar, wenn sich die geografische Position einer Anfrage sehr weit weg von den restlichen Anfragen befinden würde. Diese ließe sich nur sehr schwer in den Tourenplan einfügen und würde deswegen aufgrund ihres hohen Verlustwertes immer zuerst zur Auswahl stehen. Durch das Prinzip der Verzerrung kann somit mehr Variation in den Auswahlprozess der einzufügenden Anfragen gebracht werden und der Lösungsraum tiefgründiger untersucht werden. Zusammenfassend lässt sich über die Regret-Insertion sagen, dass ihr Konzept darauf abzielt, zuerst die Kunden in den Tourenplan wiederaufzunehmen, für die sich das Finden von alternativen Einfügepositionen am schwersten gestaltet. Falls ein kleiner Verlustwert ermittelt wird, lässt sich die betrachtete Anfrage auch leichter an alternativen Positionen in die Tour eingliedern. Eine sofortige Integration in ihre bestmögliche Position ist also nicht dringend erforderlich. Bei Anfragen, die einen hohen Verlustwert aufweisen, ist das sofortige Einfügen wichtiger, da diese sich schwerer in alternative Positionen eingliedern lassen. Deswegen stehen Anfragen mit hohem Verlustwert in der Prioritätenliste weiter oben. Zudem wird mit dem Konzept des Big-M erzwungen, dass Aufträgen, die nur eine einzige zulässige Einfügeposition besitzen, die höchste Priorität zugewiesen wird, da ihnen durch das Big-M ein sehr viel höherer Verlustwert zugeordnet wird als Anfragen mit zwei oder mehreren Einfügepositionen. Dies ist insofern sinnvoll, da der Hauptzweck des Modells darin besteht, möglichst viele Anfragen zu akzeptieren und somit möglichst viele Kunden in den Tourenplan aufzunehmen. Falls mehrere Anfragen nur eine zulässige Einfügeposition besitzen, steht aufgrund der Art der Differenzbildung diejenige

Anfrage am Anfang der Liste, die die geringste zusätzliche Fahrtzeit benötigt und aufgrund dessen am leichtesten in die Tour eingegliedert werden kann.

Nach dem Destroy- und dem anschließenden Repair-Prozess wird die hierdurch neu generierte Lösung w_{new} akzeptiert, wenn die Anzahl der eingeplanten Anfragen $|n_{w_{new}}|$ gleich bleibt oder größer ist als die Anzahl eingeplanter Anfragen $|n_w|$ der derzeit vorliegenden Lösung w. Da meist Services mit vollständigem Nutzungsgrad untersucht werden, und die Flexibilität des Routing-Prozesses dadurch eingeschränkt wird, hat dieses Akzeptanzkriterium den Vorteil, maximale Diversifikation in Bezug auf die gesamte Fahrtzeit zu ermöglichen und gleichzeitig zu verhindern, dass die Anzahl der angenommenen Anfragen sich verringert. Nachdem die neue Lösung w_{new} akzeptiert wurde, wird sie als die derzeit vorliegende Lösung w gespeichert, die in der nächsten Iteration weiterverwendet wird. Nun kann zusätzlich auch noch untersucht werden, ob die neue Lösung w_{new} eine Verbesserung zur derzeit bestbekannten Lösung w_{best} bietet. Dies ist der Fall, wenn die Anzahl der eingeplanten Anfragen der neuen Lösung $|n_{w_{new}}|$ größer ist als die der bestbekannten Lösung $|n_{w_{best}}|$. Falls beide eine gleich hohe Anzahl an akzeptierten Anfragen aufweisen, wird die neue Lösung zur neuen bestbekannten Lösung, falls die gesamte Fahrtzeit $c(n_{w_{new}})$ geringer als die der aktuell bestbekannten Lösung $c(n_{w_{best}})$ ist. Nachdem die neue Lösung w_{new} untersucht worden ist, kann im Anschluss eine neue Iteration durchgeführt werden. Dies geschieht so lange, bis die Suche beendet und die dann bestbekannte Lösung w_{best} zurückgegeben wird (vgl. Haferkamp und Ehmke (2020, S. 8ff)).

3.2.3 Umsetzung unterschiedlicher Antizipationsebenen im Rahmen einer LNS

Im Folgenden soll kurz erörtert werden, in welcher Form die LNS in einem jeden Ansatz eingesetzt werden kann, um die Verwendung von Antizipation auf unterschiedlichen Ebenen bewerten zu können.

Im Nicht-Antizipatorischen Ansatz wird die LNS als Insertion-Heuristik eingesetzt, um bei der Auftragsannahmeentscheidung als Zulässigkeitsprüfung fungieren zu können. Bei der Routing-Entscheidung wird die LNS für die Re-Optimierung des Tourenplans verwendet. Grundlage für die Insertion-Heuristik sind die Initiallösung w_0 und die Menge der nicht eingeplanten Aufträge m_{w_0}, welche in diesem Fall nur aus der neuen aktuell eingetroffenen Anfrage $r_k \in \mathcal{R}$ besteht. Der Tourenplan n_{w_0} deckt nur die Orte $l \in \mathcal{L}$ ab, deren Summe aus

geplanter Ankunftszeit z_l^v und der Service-Zeit p_l gleich groß oder größer als der Zeitpunkt des Eintreffens der aktuell eingetroffenen Anfrage t_{r_k} ist. Das führt dazu, dass die erste Location eines Fahrzeugplans des Tourenplans n_{w_0} den aktuell angefahrenen Ort beziehungsweise den als Nächstes anzufahrenden Ort eines Fahrzeugs darstellt. Dieser darf, wie vorher bereits erwähnt, nicht verändert werden. Basierend auf diesen Input sucht die LNS nun nach einer neuen Lösung w_{new}, bei der alle Anfragen in den Tourenplan $n_{w_{new}}$ eingefügt werden. Die Suche endet entweder, wenn eine solche Lösung gefunden werden konnte, oder falls ein Maximum an Iterationen, gesteuert durch den Parameter β, ausgeführt wurde. Falls eine erfolgreiche Zulässigkeitsprüfung nicht geglückt ist, wird die zurückgegebene Lösung verworfen. Hier wird die Initiallösung w_0 wiederverwendet und diese dient somit als Initiallösung für die Zulässigkeitsprüfung der nächsten Anfrage r_{k+1}. Falls die Zulässigkeitsprüfung erfolgreich gelingen konnte, wird die gefundene Lösung für das Routing als Initiallösung für den Re-Optimierungsschritt verwendet. Die LNS führt hierbei eine Re-Optimierung unter der Zielfunktion, die gesamte Fahrtzeit zu minimieren, in β Iterationen aus.

Auf der Ebene der Antizipatorischen Akzeptanz findet ebenfalls eine Zulässigkeitsprüfung bei der Auftragsannahmeentscheidung und eine Re-Optimierung für die Routingentscheidung statt, das Grundgerüst ähnelt somit der Herangehensweise des Nicht-Antizipatorischen Konzepts. Die Annahmeentscheidung beinhaltet hier jedoch neben der Prüfung der Zulässigkeit zusätzlich auch noch eine Prüfung der Vorteilhaftigkeit. Das heißt, es wird beobachtet, ob das Annehmen einer Anfrage entsprechend den Nebenbedingungen des Modells gültig und zusätzlich in Hinblick auf die Zielfunktion des Modells auch noch vorteilhaft ist. Wenn die Annahme anderer Anfragen als wertvoller erachtet wird, können folglich auch Anfragen abgelehnt werden, obwohl deren Bedienung potentiell möglich wäre. Um diese Prüfung der Vorteilhaftigkeit durchführen zu können, wird das Prinzip des TOP verwendet. Als Basis dient hier wieder die Initiallösung w_0, die aus demselben Tourenplan wie bei der Zulässigkeitsprüfung des Nicht-Antizipatorischen Ansatzes besteht. Die Menge der nicht eingeplanten Anfragen m_{w_0} enthält nun zusätzlich, neben der neu eingetroffenen Anfrage r_k, alle Anfragen, die in den folgenden Entscheidungsperioden noch eintreffen werden. Auf Grundlage dieser Informationen wird mithilfe der LNS versucht, die Anzahl der eingeplanten und somit angenommenen Anfragen $|n_w|$ zu maximieren. Damit eine neue Lösung w_{new} als beste gefundene Lösung w_{best} akzeptiert werden kann, wird ein zusätzliches Kriterium angewendet, welches prüft, ob alle im initialen Tourenplan n_{w_0} eingeplanten Anfragen auch

im neuen Tourenplan $n_{w_{new}}$ enthalten sind. Die Suche endet, sobald entweder eine Lösung w_{new} gefunden wurde, bei der alle in der Suche berücksichtigten Anfragen in den Tourenplan aufgenommen werden konnten, oder nachdem die Iterationsgrenze β erreicht wurde. Sobald die Suche beendet wurde, wird überprüft, ob die derzeit vorliegende Anfrage r_k in dem von der Suche zurückgegebenen Tourenplan $n_{w_{best}}$ enthalten ist. Falls dies der Fall ist, bedeutet das, dass die Anfrage die Prüfung der Vorteilhaftigkeit erfolgreich bestanden hat und somit angenommen werden kann.

Beim Antizipatorischen Routing wird die LNS primär genutzt, um die Zulässigkeitsprüfung der Akzeptanzentscheidungen durchführen zu können. Die Zulässigkeitsprüfung beginnt mit einer Initiallösung w_0, die sich aus einem leeren Tourenplan n_{w_0} oder einem Tourenplan, der aus der letzten erfolgreichen Zulässigkeitsprüfung resultiert, zusammensetzt. Zusätzlich zum Tourenplan wird die Menge der nicht eingeplanten Anfragen m_{w_0}, die in diesem Fall nur die neue Anfrage $r_k \in \mathcal{R}$ beinhaltet, betrachtet. Nun werden die Zulässigkeitsprüfungen für alle dem Modell zugrundeliegenden Anfragen nacheinander durchgeführt. Bei der Zulässigkeitsprüfung der ersten Anfrage wird ein leerer Tourenplan verwendet. Für alle darauffolgenden Anfragen gilt das im Folgenden erklärte Prinzip. Falls das Einfügen einer Anfrage erfolgreich gelungen und somit zulässig ist, wird der aus dem Einfügen resultierende Tourenplan als Grundlage für die Zulässigkeitsprüfung der darauffolgenden Anfrage verwendet. Falls das Einfügen einer Anfrage nicht gelungen ist, da dieser Schritt nicht zulässig ist, ändert sich am Tourenplan der Zulässigkeitsprüfung der nächsten Anfrage nichts. Der initiale Tourenplan ist in diesem Fall derselbe wie der initiale Tourenplan der vorherigen nicht zulässigen Anfrage. Das heißt, es existiert im gesamten Verlauf der Zulässigkeitsprüfung nur ein initialer Tourenplan. Dieser beginnt mit einer leeren Tour und wird jedes Mal überschrieben, falls eine neue Anfrage in den Tourenplan eingefügt werden kann. Nachdem alle Zulässigkeitsprüfungen durchgeführt wurden, wird die LNS zum Bilden der finalen Routingentscheidung verwendet. Das Ziel ist hierbei, die Lösung, die aus der letzten erfolgreichen Zulässigkeitsprüfung resultiert, weiter zu verbessern. Genauer gesagt wird versucht, durch das Anwenden der LNS die gesamte Fahrtzeit dieses Tourenplans zu minimieren.

Im Vollkommen-Antizipatorischen Ansatz wird die LNS angewendet, um das TOP lösen zu können. Die initiale Lösung w_0 besteht aus einem leeren Tourenplan n_{w_0} und der Menge der nicht eingeplanten Anfragen m_{w_0}, die in diesem Fall alle Kundenanfragen $r \in \mathcal{R}$ des Modells enthält. Die Lösung w wird dann unter Berücksichtigung der Anzahl aller eingeplanten

Anfragen $|n_w|$ und der gesamten Fahrtzeit $c(n_w)$ in β Iterationen optimiert (vgl. Haferkamp und Ehmke (2020, S. 10f)).

4 Beispielhafte Implementierung antizipatorischer Lösungsansätze

In diesem Kapitel sollen nun konkrete Lösungsverfahren gezeigt werden. Hierzu dient das im vorherigen Kapitel vorgestellte Modell als Grundlage, um eine eigens kreierte Beispielinstanz lösen zu können. Im ersten Teil dieses Kapitels sollen die Beispielinstanz und die beispielhaft implementierten Modelle vorgestellt werden. Der zweite Teil des Kapitels widmet sich anschließend den daraus resultierenden Ergebnissen. Abschließend werden die Vor- und Nachteile der Verwendung antizipativer Strategien diskutiert.

4.1 Programmierung der Beispiele in Python

Zuerst sollen die eigens erstellten Programme vorgestellt werden. Diese wurden mithilfe der Programmiersprache Python umgesetzt. Die Programme dienen dazu, eine eigens kreierte Beispielinstanz lösen zu können. Die Beispielinstanz soll im ersten Teil dieses Unterkapitels genauer vorgestellt werden. Anschließend wird erklärt, wie die unterschiedlichen Antizipationsansätze in Programmcode umgesetzt werden konnten.

4.1.1 Einführung der Beispielinstanz

Die zentralen Daten der Beispielinstanz können in Tabelle 1 und Abbildung 1 eingesehen werden.

Kunde	Eintreffzeit	Zeitfenster	Kunde	Eintreffzeit	Zeitfenster
0	2	[2, 10.24]	5	11	[11, 22]
1	4	[4, 12.83]	6	13	[13, 21.24]
2	8	[8, 18.12]	7	15	[15, 23.24]
3	9	[9, 18]	8	18	[18, 25]
4	10	[10, 17]	9	22	[22, 32]

Tabelle 1: Kundendaten der Beispielinstanz

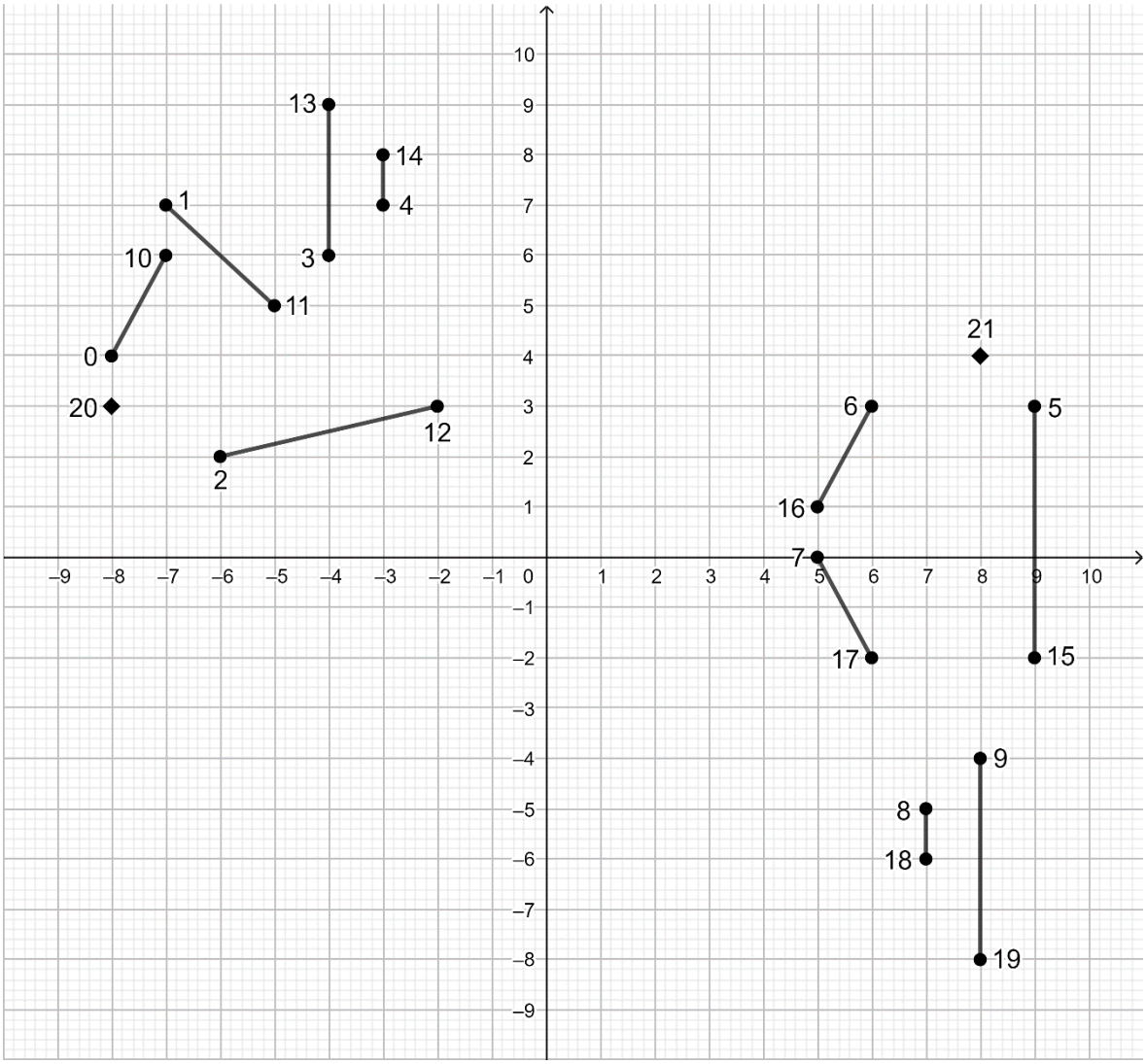

Abbildung 1: Geografische Positionen der Knoten der Beispielinstanz

Damit ein jeder Ansatz auf die Beispielinstanz angewendet werden kann, beginnt ein jedes jeweiliges Programm mit der Einführung der grundlegenden Daten der Beispielinstanz. Wie die Umsetzung der Beispielinstanz in Programmcode erfolgen kann, soll nun für alle Ansätze exemplarisch erläutert werden.

Nach den benötigten Import-Befehlen werden die Inputdaten der Beispielinstanz eingeführt. Zuerst wird mit der Funktion $random.seed()$ ein Startpunkt für das Generieren von Zufallszahlen festgelegt. Dieser gewährleistet, dass bei jedem erneuten Durchlauf des Skripts stets dieselbe identische Abfolge an Zufallszahlen gezogen wird. Hiermit wird gewährleistet, dass die Ergebnisse gleichbleiben, und nicht unterschiedliche Zufallszahlen zu möglicherweise veränderten Ergebnissen führen. Im Anschluss wird in der Variable $knoten$ eine Liste mit den Zahlen 0 bis 21 angelegt, diese Liste stellt die Knoten der Beispielinstanz dar. Die Knoten 0 bis 9 stehen hierbei für die Ursprünge der Kunden 0 bis 9, die Knoten 10 bis 19 stellen die entsprechenden Zielorte der Kunden 0 bis 9 dar, Knoten 20 und 21 sind die Standorte der beiden Fahrzeuge. Diese spezielle Notation resultiert zum einen aus der in Python verwendeten Zählweise, die standardmäßig mit der Zahl 0 beginnt, zum anderen erleichtert eine solche Notation an einigen Stellen die Programmierung. In der nun folgenden Variable $locations$ werden die geografischen Positionen der Knoten des Modells festgehalten. Die Koordinaten werden jeweils in Tupel-Schreibweise in Klammern notiert. Der erste Wert einer jeden Klammer ist die entsprechende x-Koordinate, der zweite Wert in der Klammer steht für die zugehörige y-Koordinate. Über den Index der Liste der Variable $locations$ kann nun das Koordinaten-Tupel eines jeden Knotens abgefragt werden, da die Elemente der Liste der Variable $knoten$ den Indizes beziehungsweise den Positionen der Liste der Variable $locations$ entsprechen. Um beispielsweise den Ursprung und den Zielort der nullten Anfrage ermitteln zu können, müssen die Positionen 0 und 10 der in $locations$ gespeicherten Liste abgefragt werden. Für die anderen Knoten der Liste $knoten$ erfolgt dies analog. Die Variable $anzLocations$ ermittelt mit der Funktion $len(locations)$ die Länge der Liste $locations$ und stellt somit die Anzahl aller Knoten des Modells dar.

Im Anschluss wird eine neue Funktion $distanzberechnung()$ erstellt, welche die euklidische Distanz zwischen zwei Knoten berechnen soll. Hierfür wird mit dem Schlüsselwort def eine neue Funktion mit dem Namen $distanzberechnung$ definiert, in der anschließenden Klammer befinden sich die beiden Argumente der Funktion, die innerhalb der Funktion mit den Buchstaben a und b bezeichnet werden. Die beiden Argumente sind die beiden Koordinaten-Tupel der beiden Knoten, deren Distanz zueinander berechnet werden soll. Beim

Funktionsaufruf werden diese beiden Tupel der Funktion übergeben, diese berechnet dann die Differenz zwischen den beiden x- und den y-Koordinaten und speichert sie in den Variablen dx und dy. Zurückgegeben wird die Wurzel aus der Summe von dx^2 und dy^2, also die Länge der Strecke des direkten Weges von einem zum anderen Knoten. Im Anschluss wird die Distanz- beziehungsweise Kostenmatrix c_ij erstellt, indem ein Array in der Größe der Anzahl aller Knoten des Modells erstellt wird. Im Anschluss wird das Array befüllt, indem für eine jede mögliche Kombination zweier Knoten des Modells die Distanz zueinander berechnet wird. Hierfür wird dem Array in der Zeile i und der Spalte j mithilfe der Funktion $distanzberechnung()$ der Wert der euklidischen Distanz vom Knoten i zum Knoten j übergeben. Der Funktion $distanzberechnung()$ muss hierfür das jeweilige Koordinaten-Tupel der beiden Knoten i und j übergeben werden. Nach dem Durchlauf beider for-Schleifen stellt c_ij letztendlich die Distanzmatrix dar, in der durch den Zugriff auf die entsprechenden Zeilen und Spalten des Arrays die Distanzen aller Knoten zueinander abgefragt werden können.

Anschließend wird eine hinreichend große Zahl Big-M festgelegt, welche an manchen Stellen die Programmierung der Nebenbedingungen erleichtern soll. In der Variable $anfragen$ wird eine Liste von 0 bis 9 angelegt, diese bezeichnet alle Kunden des Modells. Die Variable $fahrzeuge$ enthält die Liste aller eingesetzten Fahrzeuge des Modells, also die Fahrzeuge 0 und 1. In der Variable t_r werden nun die konkreten Eintreffzeitpunkte der Kundenanfragen in einer Liste festgehalten. Der Einfachheit halber wurden die konkreten Eintreffzeitpunkte so gewählt, dass die Kunden entsprechend ihrer Nummer in aufsteigender Reihenfolge in das System eintreten. Das heißt, dass zuerst die Anfrage des Kunden 0 eintreffen wird, anschließend die des Kunden 1, dann die des Kunden 2, et cetera. Die Variable $alpha$ enthält den Parameter für die maximale Verspätung, die bei der Erfüllung einer jeden Kundenanfrage toleriert wird. Die maximal erlaubte Verspätung beträgt in diesem Fall 6 Zeiteinheiten.

Die Variable $früest$ beinhaltet die Liste der frühestmöglichen Bedienzeiten aller Kunden, hierfür wird ihr der Wert der Variable t_r zugewiesen. Folglich stellt das Eintreffen einer Anfrage in das System gleichzeitig den Zeitpunkt dar, ab dem der Kunde auf seine Beförderung zum jeweiligen Zielort wartet und somit bedient werden kann. Zur Vereinfachung im späteren Programmablauf, in dem über die Liste $früest$ iteriert wird, werden der Liste auch Werte für die Knoten 10 bis 19 und die Knoten 20 und 21 der beiden Fahrzeuge angehängt. Aus logischer Sicht betrachtet haben diese angehängten Werte jedoch keine Aussagekraft, da für die Zielorte der Kunden die Werte der Variable $spätest$ relevant sind.

Diese Variable enthält die Liste der spätestmöglichen Bedienzeiten der Kundenanfragen, das heißt zu diesen jeweiligen Zeitpunkten muss der jeweilige Kunde an seinem jeweiligen Zielort abgesetzt worden sein, ansonsten ist die Bedienung des Kunden nicht zulässig. Die späteste Bedienzeit eines Kunden errechnet sich aus der Addition der frühesten Bedienzeit des Kunden, der Distanz des direkten Weges von diesem Kunden zu seinem entsprechenden Zielort und des maximalen Verspätungsparameters $alpha$. Für eine jede Anfrage aus der Liste $anfragen$ wird die späteste Bedienzeit ermittelt und anschließend der in $spätest$ gespeicherten Liste angehängt. Anschließend wird zur Vereinfachung, analog zu $früh est$, für jeden Knoten des Modells ein spätester Bedienzeitpunkt ermittelt, auch für die Knoten 0 bis 9. Die Knoten der Fahrzeuge, Knoten 20 und 21, erhalten den Wert Big-M, damit beim Programmdurchlauf keine Probleme auftreten.

4.1.2 Nicht-Antizipatorischer Ansatz

Nachdem die Inputdaten der Beispielinstanz beleuchtet wurden, kann nun der Programmablauf für den Nicht-Antizipatorischen Ansatz genauer erläutert werden. Dieser ist im Python-Skript *NoneAnt.py* hinterlegt.

Begonnen wird mit dem Anlegen der Variable $tourenplan$, dieser wird eine Liste übergeben, welche zwei weitere Listen enthält. Die gesamte Liste stellt den gesamten Tourenplan des vorliegenden Modells dar, wobei die erste innere Liste den Tourenplan des ersten Fahrzeuges, also Fahrzeug 0, und die zweite innere Liste den Tourenplan des zweiten Fahrzeuges, also Fahrzeug 1, beinhaltet. Die beiden Tourenpläne erhalten als Startwert die Knoten, von denen die beiden Fahrzeuge aus starten, also die Knoten 20 und 21 für die Fahrzeuge 0 respektive 1. Im weiteren Verlauf des Skripts werden dem Tourenplan nun nach und nach weitere Knoten aus der Liste $knoten$ angehängt, die beiden inneren Listen stellen also die Abfolge an Knoten dar, die von dem jeweiligen Fahrzeug nacheinander abgehandelt wird. Der Variable $finalTourenplan$ wird eine Liste mit zwei leeren inneren Listen übergeben, diese Variable stellt somit den finalen Tourenplan der Endlösung dar. Dieser Liste werden im Verlauf des Programmdurchlaufs die vergangenen Ereignisse des Tourenplans übergeben. Das sind die Anfragen, welche bereits bedient wurden und somit abgeschlossen worden sind. Nach vollständigem Durchlauf des Programms kann der finale Tourenplan der Endlösung aus der Variable $finalTourenplan$ abgelesen werden. Die Variable $zeitenplan$ funktioniert analog zu $tourenplan$ und enthält die Ankunftszeiten, zu denen ein Fahrzeug an einem besuchten Knoten ankommt. Beiden inneren Listen wird der Wert 0 übergeben,

somit starten die Touren der beiden Fahrzeuge am jeweiligen Startknoten zum Zeitpunkt 0. Im Verlauf des Programms werden im Zeitenplan die Ankunftszeiten festgehalten, zu denen ein Fahrzeug am entsprechenden Knoten des Tourenplans ankommt. Die Listen $tourenplan$ und $zeitenplan$ stimmen in ihrer Länge beide überein, somit kann in $tourenplan$ eingesehen werden, in welcher Reihenfolge verschiedene Knoten der Beispielinstanz besucht werden, und in $zeitenplan$ kann abgelesen werden, zu welchen Zeitpunkten die Ankunft am entsprechenden Knoten geschieht. Die Variable $finalZeitenplan$ funktioniert wie $finalTourenplan$ und enthält den finalen Zeitenplan der Endlösung. Aus $finalZeitenplan$ kann somit am Ende des Programmdurchlaufs der endgültige Zeitenplan abgelesen werden, der die Zeitpunkte der Besuche der in $finalTourenplan$ festgehaltenen Knoten darstellt. Die Variable $abgelehnteAnfragen$ enthält die Liste der Anfragen, welche abgelehnt werden, und die Menge der angenommenen Anfragen wird in der in $angenommeneAnfragen$ angelegten Liste festgehalten. Logischerweise sind beide Listen zum Start des Programms leer, und im Laufe des Modelldurchlaufs wird dynamisch bestimmt, welche Anfragen angenommen oder abgelehnt werden.

Nun soll der konkrete Programmablauf des Nicht-Antizipatorischen Ansatzes erfolgen. Hierfür wird eine for-Schleife verwendet, welche über die Werte der in t_r gespeicherten Liste iteriert. Die for-Schleife wiederholt sich für jeden Wert aus t_r, die Zählvariable $eintreffzeit$ hält somit den Zeitpunkt des Eintreffens einer jeden Anfrage fest. Hiermit wird gewährleistet, dass mit jedem Eintreffen einer neuen Anfrage eine neue Entscheidungsperiode eingeleitet wird und so unmittelbar über diese entschieden werden kann. Für die Zeiträume zwischen den Eintreffzeitpunkten wird der Tourenplan der letzten Entscheidungsperiode weiterverwendet. Zum Start des Codeblocks innerhalb der for-Schleife muss zuerst die fortgeschrittene Zeit berücksichtigt werden, welche zwischen den Eintreffzeitpunkten vergangen ist. Deswegen müssen vergangene Ereignisse aus dem aktuellen Tourenplan entfernt werden. Hierfür wird die Variable $zuEntfernen$ mit einer leeren Liste erstellt. In dieser werden die zu entfernenden Knoten des aktuellen Tourenplans festgehalten. Mithilfe zweier verschachtelter for-Schleifen wird über die Tourenpläne der beiden Fahrzeuge iteriert, damit jeder Knoten des Tourenplans untersucht werden kann. Allerdings ist zu beachten, dass die innere for-Schleife nur bis zum vorletzten Element der Tourenplan-Liste läuft. Das letzte Element der Tourenplan-Liste wird somit ausgelassen, da dies die aktuelle Position des Fahrzeuges ist, falls alle vorherigen Stationen des Tourenplans entfernt werden können. Untersucht werden soll, ob der Besuch eines Knotens durch das Fortschreiten der Zeit inzwischen ein vergangenes Ereignis ist. Dies ist der Fall, falls der Ankunftszeitpunkt

eines Fahrzeuges am jeweiligen Knoten vor dem Eintreffzeitpunkt der neuen Anfrage liegt. Um dies überprüfen zu können, wird eine if-Bedingung verwendet, welche für einen Knoten des Tourenplans den entsprechenden Wert des Zeitenplans ermittelt und überprüft, ob dieser kleiner ist als der Eintreffzeitpunkt der neu eingetroffenen Anfrage. Ist dies der Fall, ist die Bedingung erfüllt und die darauffolgende Code-Zeile kann ausgeführt werden, da der Ankunftszeitpunkt am besuchten Knoten vor dem Eintreffen der neu eingetroffenen Anfrage liegt und somit bereits stattfand. Der Knoten kann nun bei Erfüllung der if-Bedingung entfernt werden, hierfür wird der Liste $zuEntfernen$ der Wert der Zählvariable $kunde$, also der derzeit untersuchte Knoten des Tourenplans, angefügt. Nach dem Durchlaufen beider for-Schleifen wurden nun alle vergangenen Ereignisse des Tourenplans identifiziert und finden sich in der Liste $zuEntfernen$ wieder. Die Liste $zuEntfernen$ kann nun wiederum mithilfe zweier for-Schleifen durchlaufen werden, um den aktuellen Tourenplan bereinigen zu können. Die Variable i enthält den Index des derzeit untersuchten Fahrzeuges, die Variable $index$ hält den derzeit untersuchten Knoten des Tourenplans fest. Nun wird mithilfe der ermittelten Indizes der zu entfernende Knoten des Tourenplans aus der Liste der Variable tourenplan entfernt, eine gleichzeitige Verschachtelung mit der append-Funktion ermöglicht, dass dieser Knoten direkt in die Liste der Variable $finalTourenplan$ eingefügt werden kann. Dasselbe geschieht mit den entsprechenden Ankunftszeiten der zu entfernenden Knoten, die auf diese Weise aus dem aktuellen Zeitenplan entfernt und in den finalen Zeitenplan eingefügt werden können. Die Codezeile für den Zeitenplan muss hierbei vor der Zeile des Tourenplans stehen, da es ansonsten zu Fehlern kommt. Nach dem Durchlaufen beider for-Schleifen wurden alle vergangenen Ereignisse des Tourenplans aus dem aktuellen Tourenplan entfernt und nun kann der weitere Programmverlauf erfolgen.

Der erste Wert der Liste tourenplan stellt nun entweder die aktuelle Position des dazugehörigen Fahrzeugs oder das Ziel, das gerade vom Fahrzeug angefahren wird, dar. Anschließend wird in der Variable $aktuelleAnfrage$ die Nummer der gerade eingetroffenen Anfrage festgehalten, diese Nummer wird ermittelt, indem die Position des aktuellen Eintreffzeitpunktes in der Liste der Eintreffzeitpunkte t_r bestimmt wird. Diese korrespondiert mit der Anzahl der Anfragen, somit stellt der ermittelte Index der Liste die Nummer der aktuell eingetroffenen Anfrage dar. Die Variable $ungeplanteAnfragen$ stellt die Liste der Anfragen dar, welche zur aktuellen Zeit noch nicht in den Tourenplan eingeplant wurden. Der in $ungeplanteAnfragen$ angelegten Liste wird die neu eingetroffene Anfrage übergeben, da über diese Anfrage im Folgenden entschieden werden muss, ob sie angenommen und somit in den Tourenplan integriert oder letztendlich abgelehnt wird. Im Anschluss wird in $beta$

die Anzahl an Iterationen der folgenden while-Schleife festgelegt. Des Weiteren wird eine Variable $newTourenplan$ angelegt, mit der Deepcopy-Funktion wird eine Kopie der Variable $tourenplan$ erstellt und in $newTourenplan$ gespeichert. Hier ist zu beachten, dass mit dieser Funktion ein neues eigenständiges Objekt erzeugt wird, eine Veränderung in $newTourenplan$ oder $tourenplan$ beeinflusst den Inhalt der jeweils anderen Variable somit nicht. Der Grund für das Anlegen einer Kopie des aktuellen Tourenplans ist der nachfolgende Programmablauf, innerhalb dem der Tourenplan nämlich verändert wird. In der Variable $newTourenplan$ kann somit ein neuer Tourenplan kreiert werden, da versucht wird, die neu eingetroffene Anfrage in den aktuellen Tourenplan zu integrieren. Dieselbe Prozedur erfolgt auch für den Zeitenplan, auch hier wird in $newZeitenplan$ eine eigenständige Kopie des Inhalts von $zeitenplan$ gespeichert.

Nun kann die while-Schleife erläutert werden, innerhalb der der Destroy- und Repair-Prozess erfolgt. Die while-Schleife läuft durch, solange $beta$ nicht 0 ist und zugleich die in $ungeplanteAnfragen$ hinterlegte Liste nicht leer ist. Die Schleife wird folglich ausgeführt, bis entweder das Iterationskriterium $beta$ erreicht wurde, also 100 Iterationen durchgeführt wurden, oder bis keine ungeplanten Anfragen mehr vorliegen, also alle untersuchten Kundenanfragen in den Tourenplan integriert werden konnten. Nach einer jeden Iteration werden diese beiden Bedingungen überprüft. Innerhalb einer Iteration der while-Schleife geschieht der Destroy- und Repair-Prozess. Zuerst wird im Rahmen des Destroy-Schritts eine zufällige Anfrage aus dem Tourenplan entfernt. Hierfür wird in $destroy$ eine leere Liste angelegt. Nun wird mittels zweier for-Schleifen untersucht, welche Anfragen überhaupt aus den Tourenplan entfernt werden können. Die innere Schleife iteriert erst ab der zweiten Position der Tourenplan-Liste über diese Liste, da die erste Position des Tourenplans den aktuellen Standort beziehungsweise das aktuell angefahrene Ziel des Fahrzeugs darstellt und somit nicht entfernt werden kann. Anschließend wird mittels einer if-Bedingung ermittelt, ob sich aktuell sowohl der Ursprungsort als auch der Zielort eines Kunden in dem Tourenplan befindet. Mit dieser Bedingung wird sichergestellt, dass nur ganze Anfragen aus dem Tourenplan entfernt werden können. Falls die Bedingung erfüllt ist, wird der Liste in $destroy$ der aktuell untersuchte Knoten hinzugefügt. Da in der oberen Bedingung zur Nummer des Knotens auch noch den Wert 10 hinzuaddiert wird, können sich in der Liste $destroy$ nur die Knoten 0 bis 9 befinden, diese entsprechen den Anfragen 0 bis 9. In der darauffolgenden if-Bedingung wird ermittelt, ob die Länge der Liste $destroy$ größer als 0 ist. Falls dies der Fall ist, kann mindestens eine Anfrage entfernt und im folgenden Abschnitt die zu entfernende Anfrage bestimmt werden. Falls die Länge von $destroy$ gleich 0 ist, ist

die Bedingung nicht erfüllt und es kann auch keine Anfrage vollständig, also samt ihres Ursprungs und ihres Zielorts, aus den Tourenplan entfernt werden. Ist die Bedingung erfüllt, wird aus der Liste $destroy$ ein zufälliger Wert ermittelt und in der Variable d gespeichert, diese Variable enthält somit die zu entfernende Anfrage. Mit einer for-Schleife werden die Tourenpläne der beiden Fahrzeuge durchsucht. Mit einer if-Bedingung wird ermittelt, in welchem der beiden Fahrzeuge sich der zu entfernende Kunde befindet, damit er aus dem Tourenplan entfernt werden kann. Hierfür werden in den folgenden 3 Zeilen der Index des entsprechenden Fahrzeuges und die Position des Ursprungsknotens und des Zielknotens der zu entfernenden Anfrage in der Tourenplanliste bestimmt. Hiermit kann im Anschluss mit der Remove-Funktion der Ursprungsknoten des zu entfernenden Kunden aus der in $newTourenplan$ gespeicherten Liste entfernt werden. Das Gleiche geschieht auch mit dem Zielknoten des zu entfernenden Kunden, dieser wird angesprochen, indem der Ursprungsknoten mit dem Wert 10 addiert wird. Auch im Zeitenplan muss der zu entfernende Kunde entfernt werden, hierfür wird mithilfe des Key-Words *del* der entsprechende Eintrag aus $newZeitenplan$ gelöscht. Wichtig ist, dass zuerst der Ankunftszeitpunkt am Zielort des Kunden entfernt wird, da ansonsten die Indizes der Liste nicht mehr übereinstimmen und der falsche Eintrag der Liste gelöscht wird. Anschließend kann der Ankunftszeitpunkt am Ursprung des Kunden aus $newTourenplan$ gelöscht werden, da dieser einen kleineren Index besitzt als die Ankunftszeit am Zielort, kann er auch problemlos gefunden und entfernt werden. Da die zufällig ausgewählte Anfrage nun aus den Tourenplan entfernt wurde, ist sie nun nicht mehr eingeplant und wird deswegen in die Liste der ungeplanten Anfragen eingefügt. Zum Schluss wird mithilfe des Keywords *break* der Ausbruch aus der for-Schleife erzwungen und die Schleife somit beendet. Zum einen, da pro Iteration nur maximal eine Anfrage zufällig ausgewählt und entfernt wird, zum anderen, da es ansonsten zu einer Fehlermeldung kommen würde, da sich der Inhalt der Liste, über die in der for-Schleife iteriert wird, verändert hat.

Im Anschluss an den Destroy-Prozess kann jetzt der Repair-Schritt erfolgen. Die Strategie ist hierbei, so viele Anfragen wie möglich aus der Liste der ungeplanten Anfragen in den Tourenplan einzufügen. Zunächst wird die Liste der ungeplanten Anfragen mithilfe der Funktion $random.shuffle()$ zufällig durchgemischt. Das führt dazu, dass die Anfragen in gewisser Weise zufällig ausgewählt werden, damit nicht immer dieselben Anfragen priorisiert integriert werden. Die Variable eingefügt besteht aus einer leeren Liste und stellt im weiteren Verlauf diejenigen Anfragen dar, welche erfolgreich in den Tourenplan eingefügt werden konnten. Die anschließende for-Schleife läuft über alle ungeplanten Anfragen, die

Zählvariable $origin$ stellt den Ursprungsknoten der in der jeweiligen Iteration untersuchten Anfrage dar. In der Variable $destination$ wird der Zielort festgehalten, der dem Ursprungsort der untersuchten ungeplanten Anfrage zugehörig ist. Die Variable $einfügenZulässig$ wird der Wert *False* zugewiesen. Dieser Wert ändert sich zum Wert *True*, sobald das Einfügen der untersuchten Anfrage sich als zulässig herausstellt. Die Variable $zusatzkostenOpt$ speichert den Wert der geringsten Zusatzkosten, dieser wird standardmäßig auf den Wert des Big-M gesetzt. Falls im weiteren Verlauf eine zulässige Einfügestelle mit geringeren Zusatzkosten gefunden werden kann, wird dieser Wert überschrieben, wodurch $zusatzkostenOpt$ zu jeder Zeit den Wert der günstigsten Einfügestelle darstellt. Zwei weitere for-Schleifen sorgen dafür, dass jede mögliche Einfügestelle des neu zu kreierenden Tourenplans betrachtet wird. Die äußere Schleife geht nacheinander die Tourenpläne der beiden Fahrzeuge durch, während bei der inneren Schleife die Zählvariable m eine jede Einfügestelle des jeweiligen Fahrzeugtourenplans darstellt. Betrachtet werden alle Einfügestellen nach dem ersten Element der Tourenplanliste, da dieses die aktuelle Position beziehungsweise das nächste Ziel des Fahrzeuges darstellt. Alle Listenpositionen dahinter können als Einfügestellen berücksichtigt werden. Der Einfachheit halber wird für eine Anfrage nur jeweils eine Einfügestelle ermittelt, das heißt, dass beim Einfügen einer Anfrage der Ursprung und der Zielort gemeinsam in die Einfügestelle eingefügt werden. Denkbar wäre beispielsweise auch, dass der Zielort an eine unterschiedliche Position in den Tourenplan eingefügt wird und nicht direkt hinter den Ursprungsort. Da dies allerdings zusätzliche Komplexität verursachen würde, wurde auf eine solche Einfügemöglichkeit verzichtet. Eine anschließende Fallunterscheidung bei der Ermittlung der Zusatzkosten untersucht, welche Art von Einfügestelle aktuell vorliegt. Falls die if-Bedingung erfüllt ist, wird aktuell eine Einfügestelle untersucht, die sich mitten in der Tourenplanliste befindet. Falls die Bedingung nicht erfüllt ist, befindet sich die aktuelle Einfügestelle am Ende der Tourenplanliste. Für die Berechnung der Zusatzkosten ist das wichtig, denn im ersten Fall wird die einzufügende Anfrage mitten in die Tour integriert. Das heißt, dass sich die Zusatzkosten hier aus der direkten Strecke vom Vorgängerknoten der Einfügestelle zum Ursprungsknoten der einzufügenden Nachfrage, der direkten Strecke vom Ursprungsknoten zum Zielknoten der einzufügenden Nachfrage, und der Strecke vom ebengenannten Zielort zum Nachfolgerknoten der Einfügestelle zusammensetzen. Abgezogen werden muss hiervon noch die Strecke vom Vorgängerknoten der Einfügestelle zum Nachfolgerknoten. Im zweiten Fall wird die Anfrage einfach ans Ende des Tourenplans angehängt, die Zusatzkosten errechnen sich hier durch das Addieren der Wegstrecke vom letzten Knoten der Tourenplanliste zum

Ursprungsknoten der einzufügenden Anfrage und der Distanz vom Ursprungsknoten zum dazugehörigen Zielknoten der einzufügenden Anfrage. Nach der Berechnung der Zusatzkosten prüft eine weitere if-Bedingung, ob eine neue beste Einfügestelle gefunden worden ist. Dies ist der Fall, wenn die soeben ermittelten Zusatzkosten geringer sind, als die Zusatzkosten der bisher besten gefundenen Lösung. Falls eine neue aus Kostensicht betrachtet beste Lösung gefunden wurde, wird eine leere Liste in der Variable $zulässig$ angelegt, die später noch gebraucht wird. Zudem wird die Variable $provisionsTourenplan$ erstellt, dieser wird eine Kopie von $newTourenplan$ übergeben, allerdings nur der Tourenplan des geradeeben untersuchten Fahrzeugs k. Dieser provisorische Tourenplan dient im weiteren Verlauf der Überprüfung, ob das Einfügen der Anfrage in die neue günstigste Einfügestelle überhaupt zulässig ist. Um dies zu untersuchen, wird die Anfrage an der neu ermittelten günstigsten Einfügestelle in den entsprechenden Tourenplan integriert. Damit die Reihenfolge stimmt, wird zuerst der Zielort, dann der Ursprungsort integriert. Für die Variable $provisionsZeitenplan$ wird eine Kopie des entsprechenden Zeitenplans angefertigt, allerdings nur mit dem Eintreffzeitpunkt der ersten Station des Tourenplans, dieser ist ja wie bereits gesagt unveränderlich. Alle anderen Eintreffzeitpunkte der darauffolgenden Anfragen müssen aufgrund des Integrierens einer neuen Anfrage natürlich neu berechnet werden. Eine anschließende for-Schleife iteriert über alle Stationen des provisorischen Tourenplans, damit der zugehörige neue provisorische Zeitenplan berechnet werden kann. Hierfür wird in $indexHaltestelle$ die Position des aktuell betrachteten Knotens innerhalb der Tourenplanliste festgehalten. In der Variable $zeitVorgänger$ wird ermittelt, zu welcher Zeit sich das Fahrzeug am Vorgänger des derzeit betrachteten Knotens befindet. Aus den beiden Werten in der Klammer, die im provisorischen Zeitenplan festgelegte Ankunftszeit am Vorgängerknoten und der Eintreffzeitpunkt der neu eingetroffenen Anfrage, muss nun das Maximum ermittelt werden, damit bestimmt werden kann, wann sich das Fahrzeug am Vorgängerknoten befindet und zum aktuell betrachteten Knoten losfahren kann. Falls die Ankunftszeit größer als der Eintreffzeitpunkt ist, befindet sich das Fahrzeug noch auf dem direkten Weg zum Vorgängerknoten und kann erst nach der Ankunft weiterfahren, falls die Ankunftszeit kleiner ist als der Eintreffzeitpunkt, wartet das Fahrzeug aktuell am Vorgängerknoten auf weitere Anweisungen und kann sich somit direkt auf den Weg zum derzeit untersuchten Knoten begeben. Die Bestimmung der Variable $zeitVorgänger$ ist wichtig für die darauffolgende Variable $mindestZeit$, diese ermittelt nämlich, wann das Fahrzeug am derzeit betrachteten Knoten ankommen wird. Hierfür benötigt man die gerade eben berechnete Ankunftszeit am Vorgängerknoten und addiert zu dieser noch die Strecke des direkten Weges

vom Vorgängerknoten zum derzeit betrachteten Knoten hinzu. Die errechnete Ankunftszeit kann nun im provisorischen Zeitenplan angehängt werden, nach dem Durchlaufen der for-Schleife hat man somit für jeden Knoten des provisorischen Tourenplans eine neue Ankunftszeit berechnet.

Nun kann für den provisorischen Tourenplan überprüft werden, ob sich die Ankunftszeiten an jedem Knoten innerhalb der Zeitfenster der Kunden befinden. Eine for-Schleife sorgt dafür, dass jede Station des provisorischen Tourenplans auf diese Bedingung geprüft wird. Zuerst wird in $indexHaltestelle$ die Position des aktuell untersuchten Knotens in der Tourenplanliste bestimmt. Danach prüft eine if-Bedingung, ob der im provisorischen Zeitenplan festgehaltene Ankunftszeitpunkt an einem Knoten größer oder gleich groß wie der früheste Bedienzeitpunkts und zugleich kleiner oder gleich groß wie der späteste Bedienzeitpunkt des zugehörigen Kunden ist. Wenn dem so ist, dann befindet sich der Bedienzeitpunkt des Kunden innerhalb der individuellen Bedienzeitfenster und die Bedienung der Anfrage ist zulässig. Wie bereits im Vorhinein bei den Inputdaten der Beispielinstanz erklärt wurde, hilft die Struktur der verwendeten Listen dabei, die Zeitfensterbedingung in einer einzigen Zeile abzufragen. Die in $frühest$ und $spätest$ gespeicherten Listen hängen jedem Knoten, egal ob Ursprungs- oder Zielknoten eines Kunden, das individuelle Zeitfenster des Kunden an. So ist beispielsweise in der Liste $frühest$ der Wert vom Kunden 0 derselbe wie der des Kunden 10, das gleiche ist bei der Liste in $spätest$ der Fall, wo die Werte an der Stelle 0 und 10 identisch sind. Falls ein Kunde innerhalb seines Bedienzeitfensters vom Ursprungsknoten abgeholt und zum Zielknoten befördert wird, ist diese if-Bedingung erfüllt, falls dies nicht innerhalb des Zeitfensters geschehen kann, ist die Bedingung nicht erfüllt und die Codezeile nach dem Keyword *else* wird ausgeführt. Nach erfüllter if-Bedingung wird der Liste in zulässig der Wert *True* angehängt, da das Zeitfenster dieses Kunden eingehalten wurde, die Bedienung dieses Kunden an dieser Position im Tourenplan zur vorgesehenen Zeit ist zulässig. Die Codezeile nach dem Keyword *else* hängt der Liste in zulässig den Wert *False* an, da bei Nicht-Erfüllung der if-Bedingung das kundenindividuelle Zeitfenster nicht eingehalten werden konnte. Eine weitere for-Schleife soll überprüfen, ob die Anfragen, welche in den vorherigen Entscheidungsperioden bereits angenommen aber im Destroy-Schritt aus der Tour entfernt wurden, in den Tourenplan wieder aufgenommen wurden. Hierzu wird eine for-Schleife verwendet, die über die Liste $angenommeAnfragen$ iteriert. Es wird für alle bisher angenommenen Anfragen überprüft, ob diese sich mittlerweile in der Liste der ungeplanten Anfragen befinden. Dies geschieht innerhalb einer if-Bedingung. Falls dem so ist, wurde eine Anfrage identifiziert, die in einer vergangenen Entscheidungsperiode

angenommen, im vorigen Destroy-Schritt jedoch wieder entfernt wurde. Eine weitere if-Bedingung prüft, o sich dieser identifizierte Kunde im provisorischen Tourenplan befindet. Falls dem so ist, wird der Liste zulässig der Wert *True* angehängt, da der entfernte Kunde wieder in die Tour aufgenommen wurde und dieser Tourenplan somit hinsichtlich dieses Aspekts zulässig ist. Falls die if-Bedingung verletzt wird, wird der Liste zulässig der Wert *False* angehängt, da sich der entfernte Kunde nicht im Tourenplan befindet, obwohl dieser im Vorhinein angenommen wurde und somit bedient werden muss. Die Art und Weise, wie diese Nebenbedingung im vorliegenden Fall programmiert wurde, ist zugegebenermaßen sehr simpel. Es gibt bestimmt bessere Wege, mit denen die Bedingung in diesen Programmablauf eingebettet werden kann. Um das Modell jedoch nicht allzu komplex werden zu lassen, wurde hier versucht, die Bedingung auf eine sehr einfache Art und Weise in das Python-Skript aufzunehmen. Diese ist nämlich sehr wichtig, damit Lösungen unterbunden werden können, in denen einem Kunden die zugesagte Bedienung kurze Zeit später verwehrt wird. Mit der hier verwendeten Art kann es beispielsweise passieren, dass eine neu eingetroffene Anfrage in den Tourenplan integriert wird, bevor die im Destroy-Schritt aus dem Tourenplan entfernte Anfrage wieder eingefügt wurde. Das kann durchaus der Fall sein, da die einzufügenden Kunden der Liste $ungeplanteAnfragen$ entnommen wird und diese vor einem jeden Repair-Prozess zufällig durchgemischt wird. In diesem Fall kann die neu eingetroffene Anfrage nicht in den Tourenplan integriert werden, da die bereits angenommene entfernte Anfrage bisher nicht wieder eingefügt wurde. Es könnte aber sein, dass möglicherweise bessere Lösungen gefunden werden können, wenn zuerst die neue Anfrage und danach erst die bereits angenommene Anfrage in die Tour eingefügt werden. Diese Lösungen können also im vorliegenden Skript leider nicht untersucht werden, und es kann nicht verhindert werden, dass bereits angenommene Anfragen bevorzugt wieder eingefügt werden. Im Rahmen der vorliegenden kleinen Beispielinstanz ist das aber verschmerzbar.

Im letzten Teil dieses Code-Abschnitts prüft eine weitere if-Bedingung, ob sich in der Liste zulässig der Wert *False* befindet. Falls die Liste zulässig keinen einzigen Wert *False* beinhaltet, ist die Bedingung erfüllt und der folgende Abschnitt kann ausgeführt werden. Das Erfüllen dieser Bedingung bedeutet, dass der provisorische Tourenplan alle im Vorhinein erklärten Nebenbedingungen erfüllt hat und somit ein zulässiger Tourenplan ist. Im weiteren Verlauf kann dieser provisorische Tourenplan als neue beste Lösung gespeichert werden. Hierzu wird mit $kOpt$ das gerade untersuchte Fahrzeug und $mOpt$ die derzeit betrachtete Einfügestelle festgehalten. In $zusatzkostenOpt$ wird der Wert der Zusatzkosten der besten Lösung überschrieben, da eine neue beste Lösung gefunden wurde. Der neue Wert dieser

Variable entspricht dem Wert der Zusatzkosten, der für das Einfügen des Kunden in den Tourenplan an der ermittelten Stelle berechnet wurde. Der Wert der Variable, der im vorigen Abschnitt weiter oben auf den Wert *False* gesetzt wurde, wird nun auf den Wert *True* geändert, da das Einfügen des Kunden an der vorgesehenen Stelle zulässig ist. Die Variable $zeitenplanOpt$ wird benötigt, um den Zeitenplan außerhalb der Schleife wieder vervollständigen zu können. Hierfür wird in $zeitenplanOpt$ der provisorische Zeitenplan, also $provisionsZeitenplan$, gespeichert. Nachdem alle for-Schleifen bis auf die äußerste for-Schleife in Zeile 111 vollständig durchgelaufen sind, sind alle möglichen Einfügepositionen einer ungeplanten Anfrage durchgegangen worden. Mit der anschließenden if-Bedingung wird überprüft, ob für die ungeplante Anfrage eine zulässige Einfügestelle in der Tour gefunden wurde. Falls dem so ist, werden die gespeicherten Positionen der besten Lösung benutzt, um die Anfrage in den Tourenplan einzufügen. Hierzu wird zuerst der Zielort der einzufügenden Anfrage in die Liste in $newTourenplan$ eingefügt, danach erst der Ursprungsort der einzufügenden Anfrage. Dies hat mit der Funktionsweise von Python zu tun, durch die hier verwendete Schreibweise kann die gedachte Reihenfolge eingehalten werden. Das Einfügen in $newTourenplan$ erfolgt in das in der besten Lösung vorgesehene Fahrzeug, $kOpt$, an der besten ermittelten Einfügestelle, $mOpt$. Wie bereits vorher erklärt, wird die gesamte Anfrage in die Einfügestelle integriert. Im Anschluss wird die Anfrage an die Liste eingefügt angehängt, was bedeutet, dass die ungeplante Anfrage erfolgreich eingefügt werden konnte. Hierfür reicht es aus, der Liste eingefügt den Wert $origin$ anzuhängen, da die Knoten 0 bis 9 den Anfragen 0 bis 9 entsprechen. Durch das Einfügen des ungeplanten Kunden in den Tourenplan muss nun auch der Zeitenplan korrigiert werden. Hierzu muss zuerst der Zeitenplan des Fahrzeugs, dem der ungeplante Kunde zugeordnet wurde, in der Variable $newTourenplan$ gelöscht werden. Das geschieht durch die clear-Funktion. Anschließend können die Werte des provisorischen Tourenplans einfach übernommen werden. Durch eine for-Schleife wird über die Liste des Zeitenplans der besten gefundenen Einfügestelle iteriert, ein jeder einzelner Wert dieser Liste wird mit der Zählvariable i festgehalten. Anschließend kann ein jeder Wert i der Liste an die Liste $newZeitenplan$ für das zugeordnete Fahrzeug $kOpt$ angehängt werden. Nach Ablauf der Schleife besitzt der veränderte Teil von $newTourenplan$ einen korrekten Zeitenplan, der in $newZeitenplan$ nun wieder abgelesen werden kann. Nach gesamtem Ablauf der äußeren for-Schleife in Zeile 111 wurde für eine jede ungeplante Anfrage der Liste $ungeplanteAnfragen$ versucht, diese in den Tourenplan zu integrieren. Falls dies gelingen konnte, wurde die Anfrage in der Liste eingefügt gespeichert. Im Anschluss an die for-Schleife kann nun über die Liste eingefügt iteriert

werden, um die eingefügten Anfragen aus der Liste der ungeplanten Anfragen zu entfernen. Dies geschieht mit der remove-Funktion. Die Liste der ungeplanten Anfragen wurde somit für die nächste darauffolgende Iteration bereinigt. Der letzte Teil der while-Schleife in Zeile 88 besteht darin, den Iterationszähler $beta$ um den Wert 1 zu verringern. Dies symbolisiert, dass eine weitere Iteration vergangen ist und die while-Schleife nun eine neue Iteration beginnt, falls die zwei vorgegebenen Bedingungen weiterhin erfüllt sind.

Im Anschluss an die while-Schleife kann nun ermittelt werden, ob der Destroy- und Repair-Prozess dazu geführt hat, dass die neu eingetroffene Anfrage in den Tourenplan eingefügt werden konnte und diese ganze Operation somit erfolgreich war oder nicht. Hierzu wird eine if-Bedingung verwendet, die prüfen soll, ob die Menge der ungeplanten Anfragen nicht leer ist. Falls die Liste $ungeplanteAnfragen$ nicht leer ist, bedeutet das, dass die neu eingetroffene Anfrage nicht in den Tourenplan eingefügt werden konnte, da sie sich nach dem Destroy- und Repair-Prozess weiterhin in der Menge der ungeplanten Anfragen befindet. Die neu eingetroffene Anfrage muss folglich abgelehnt werden und wird in der folgenden Code-Zeile an die Liste $abgelehnteAnfragen$ angehängt. Die darauffolgende else-Bedingung wird ausgeführt, wenn die Liste $ungeplanteAnfragen$ leer ist. Dies ist gleichbedeutend mit dem Umstand, dass keine ungeplante Anfrage mehr existiert und somit alle Anfragen in den Tourenplan eingefügt werden konnten, darunter auch unweigerlich die neu eingetroffene Anfrage. Das bedeutet, dass die Anfrage angenommen werden kann. Der Abschnitt nach der else-Bedingung führt dazu, dass dem aktuellen Tourenplan eine eigenständige Kopie des neu erstellten Tourenplans, $newTourenplan$, übergeben wird. Dasselbe geschieht mit dem Zeitenplan. Der Liste der angenommenen Anfragen, $angenommeneAnfragen$, wird die neu eingetroffene Anfrage, $aktuelleAnfrage$, angehängt. Der Abschnitt von Zeile 164 bis 171 bewirkt, dass am Ende einer jeden Entscheidungsperiode untersucht wird, ob die Anfrage angenommen oder abgelehnt werden muss. Diese Entscheidung hat auch Einfluss auf die nachfolgenden Entscheidungsperioden. Wird die Anfrage nämlich angenommen, ändert sich auch der Tourenplan, und dieser muss in der nachfolgenden Entscheidungsperiode berücksichtigt werden. Deswegen wird in Zeile 167 bis 171 sichergestellt, dass der neu gefundene Tourenplan in den aktuellen Tourenplan, an dem sich die Fahrzeuge letztendlich orientieren, überführt wird. Der aktuelle Tourenplan dient dann als der Plan, an den sich gehalten werden muss, bis das Eintreffen einer neuen Anfrage eine neue Entscheidungsperiode auslöst. Falls die Anfrage abgelehnt wird, wird nur der Abschnitt von Zeile 164 bis 166 ausgeführt wird, in dem nachverfolgt wird, dass die Anfrage abgelehnt wird. Der nachfolgende Abschnitt wird nicht ausgeführt, das bedeutet,

dass sich der aktuelle Tourenplan nicht verändert, da die neu eingetroffene Anfrage abgelehnt werden musste. In der nachfolgenden Entscheidungsperiode kann das Programm mit dem unveränderten aktuellen Tourenplan fortgesetzt werden. Das Ergebnis einer Entscheidungsperiode, in der eine Anfrage abgelehnt wurde, wird schlussendlich also ignoriert. Des Weiteren ist noch zu beachten, dass nach dem erfolgreichen Einfügen einer Anfrage eigentlich noch ein Re-Optimierungsschritt folgen müsste, in dem untersucht wird, ob der aktuelle Tourenplan noch hinsichtlich des Routings verbessert werden könnte. In diesem Skript wurde jedoch darauf verzichtet, da nach Ablauf der Insertion-Heuristik die bestmögliche Lösung bereits gefunden werden konnte und somit eine Re-Optimierung schlicht nicht mehr nötig ist. Um die Komplexität dieses Programms nicht noch unnötig zu vergrößern, wurde letztendlich auf den Re-Optimierungsschritt verzichtet. Nach dem gesamten Durchlauf der for-Schleife aus Zeile 65 wurde das Eintreffen einer jeden einzelnen Anfrage der Beispielinstanz simuliert, zu jedem Eintreffzeitpunkt wurde eine neue Entscheidungsperiode ausgelöst, in der über die Annahme der Anfrage entschieden wurde.

Bevor nun die finale Lösung ausgegeben werden kann, ist noch ein weiterer kleiner Abschnitt notwendig. In diesem werden die im aktuellen Tourenplan übriggebliebenen Elemente in den finalen Tourenplan überführt. Während dem Ablauf des Hauptabschnitts des Programms geschieht dies ja bereits, indem Vergangenheitsereignisse in den finalen Tourenplan überführt werden. Dies soll nun auch für den aktuellen Tourenplan der letzten Entscheidungsperiode geschehen. Hierfür wird mittels zweier for-Schleifen über ein jedes Fahrzeug des aktuellen Tourenplans iteriert, anschließend werden sowohl der Index des Fahrzeugs als auch die Position des Knotens in der Tour festgehalten. Im Anschluss können so die Elemente des aktuellen Zeiten- und Tourenplanes dem finalen Zeiten- beziehungsweise Tourenplanes angehängt werden. Nach dem Durchlaufen beider for-Schleifen ist der finale Tourenplan vollständig, aus diesem können nun alle getroffenen Entscheidungen des Entscheidungszeitraums entnommen werden. Zum Schluss werden einige print-Funktionen verwendet, um die zentralen Ergebnisse hervorzuheben, die aus der Anwendung des Nicht-Antizipatorischen Ansatzes auf die Beispielinstanz resultieren.

4.1.3 Antizipatorische Akzeptanz

Der Antizipatorische Akzeptanz-Ansatz ist vom Aufbau und Prinzip identisch wie der Ansatz, bei dem keine Antizipation verwendet wird. Allerdings findet auf der Ebene der Akzeptanzentscheidung neben der Prüfung der Zulässigkeit zusätzlich noch eine Prüfung

der Vorteilhaftigkeit statt. Hierfür wird das bisher vorgestellte Python-Skript um einen weiteren Aspekt ergänzt, dies soll im Folgenden erläutert werden. Erklärungen der restlichen Bestandteile des Skripts *AntAcceptance.py* sind dem vorherigen Unterkapitel zu entnehmen. Um prüfen zu können, ob eine Anfrage vorteilhaft ist, wird in Zeile 51 eine neue Funktion namens $favorabilityCheck()$ erstellt. Diese wird in Zeile 203 aufgerufen, das Ergebnis des Funktionsausrufs wird in der Variable $favorable$ gespeichert. Eine Zeile zuvor wird die Variable $favorable$ erstellt, welche anzeigen soll, ob die aktuell vorliegende Anfrage vorteilhaft ist. Im Initialzustand bekommt diese Variable den Wert *False* zugewiesen. Die Funktion $favorabilityCheck()$ wird nach ihrem Durchlauf ebenfalls entweder den Wert *True* oder *False* zurückgeben und in $favorable$ speichern. In einem weiteren Einschub in Zeile 289 wird der Wert von $favorable$ geprüft, falls der Wert *True* ist, wird die aktuell vorliegende Anfrage in den Tourenplan aufgenommen, falls der Wert *False* ist, wird die Anfrage abgelehnt. Für eine Annahme einer Anfrage ist nun nicht mehr entscheidend, ob sie bedient werden kann, sondern auch, ob diese Bedienung unter Berücksichtigung von Anfragen, welche in zukünftigen Perioden noch eintreffen werden, vorteilhaft ist. Um bestimmen zu können, ob die aktuell untersuchte Anfrage vorteilhaft ist, wird der Funktion $favorabilityCheck()$ beim Funktionsaufruf der neu zu kreierende Touren- und Zeitenplan und die aktuell untersuchte Anfrage übergeben. Innerhalb der Funktion werden von diesen Inputdaten eigenständige Kopien angefertigt, damit die anschließenden Veränderungen an den Listen nur innerhalb der Funktion erfolgen und den weiteren Programmablauf außerhalb der Funktion nicht beeinflussen. Außerdem werden alle Variablennamen innerhalb der Funktion mit einem Unterstrich am Anfang des Variablennamens erweitert, damit keine doppelten Bezeichnungen auftreten und die Übersichtlichkeit bewahrt werden kann. Neben dem Anfertigen der Kopien werden noch weitere Variablen mit dem Namenszusatz *Best* eingeführt, diese sollen den Touren- und Zeitenplan, die Zahl der insgesamt angenommen Anfragen und die Gesamtdauer der Tour der besten Lösung festhalten. Der Menge der ungeplanten Anfragen wird die aktuelle Anfrage angehängt, im Anschluss wird bestimmt, welche aktuelle Anfragen ab den aktuell untersuchten Zeitpunkt noch eintreffen werden. Hierfür wird mit einer if-Bedingung bestimmt, welche Eintreffzeitpunkte der Anfragen größer sind, als der Eintreffzeitpunkt der aktuell eingetroffenen Anfrage. Ein Erfüllen der if-Bedingung ist gleichbedeutend damit, dass die Anfrage erst in einer zukünftigen Periode eintreffen wird. Diese Anfragen werden im Anschluss an die Liste der ungeplanten Anfragen angehängt. Hier tut sich ein großer Unterschied zum Nicht-Antizipatorischen-Ansatz auf, in dem nur die aktuelle Anfrage und alle bisher eingetroffenen Anfragen betrachtet wurden.

Nun sollen die aktuelle und alle zukünftigen Anfragen berücksichtigt werden, damit bestimmt werden kann, ob die aktuelle Anfrage als wertvoll erachtet wird. Der anschließende Ablauf ist weitestgehend aus den bisherigen Erläuterungen dieses Kapitels bekannt. Eine while-Schleife gewährleistet den Ablauf des Destroy- und Repair-Prozesses, der in diesem Fall aber erst beendet werden kann, wenn 100 Iterationen absolviert wurden. Im Destroy-Schritt wird abermals ein zufälliger Kunde aus der Tour entfernt und in die Menge der ungeplanten Anfragen aufgenommen. Der anschließende Repair-Schritt versucht, die nicht eingeplanten Anfragen in einer zufälligen Abfolge wieder in den Tourenplan zu integrieren. Das Einfügen erfolgt nach demselben Prinzip, das bereits im vorherigen Unterkapitel erläutert wurde. Basierend auf der Einfügestelle mit den geringsten Zusatzkosten werden wieder provisorische Tourenpläne erstellt, welche hinsichtlich der Zulässigkeit geprüft werden. Falls eine zulässige Lösung gefunden wurde, werden die zugehörigen Daten gespeichert. Nach Überprüfung aller möglichen Einfügestellen wird die Anfrage an der besten gefundenen zulässigen Einfügestelle in die Tour integriert. Nach dem versuchten Einfügen einer jeden nicht eingeplanten Anfrage werden alle Anfragen, die eingefügt werden konnten, aus der Menge der nicht eingeplanten Anfragen entfernt. Im Anschluss wird für den nach dem Repair-Prozess entstandenen Tourenplan ermittelt, wie viele Anfragen in diesem insgesamt angenommen wurden. Dazu wird die Variable $\_insgesamtAngenommen$ erstellt und mit dem Wert 0 initialisiert. Mehrere for-Schleifen iterieren über jeden Knoten des Tourenplans, und um zu ermitteln, wie viele Kunden im gesamten Tourenplan enthalten sind, genügt es, für einen jeden Knoten, der kleiner als 10 ist, die Anzahl der insgesamt angenommenen Anfragen um 1 zu erhöhen. Im Anschluss soll die gesamte zeitliche Dauer des gesamten Tourenplans ermittelt werden. Hierzu wird die Variable $\_gesamtdauer$ mit dem Wert 0 initialisiert. Um für eine jede Einzeltour bestimmen zu können, wie lange diese Tour insgesamt dauert, muss jeweils der letzte Wert der Zeitenplan-Liste betrachtet werden. Dieser sagt aus, wann ein Fahrzeug den letzten Knoten des Tourenplans erreicht. Mit dem Wert -1 in eckigen Klammern kann auf das letzte Element der Zeitenplan-Liste zugegriffen werden, und die Gesamtdauer wird um diesen Wert erhöht. Im letzten Teil einer einzelnen while-Schleifen-Iteration wird untersucht, ob in der aktuellen Iteration eine neue beste Lösung gefunden werden konnte. Primär wird betrachtet, wie viele Kunden in einen Tourenplan eingefügt werden konnten. Wenn der Tourenplan der aktuellen Iteration eine höhere Anzahl an insgesamt angenommenen Anfragen besitzt als die aktuell beste Lösung, wird der Tourenplan der aktuellen Iteration als neue beste Lösung gespeichert. Dies wird mithilfe einer if-Bedingung sichergestellt. Festgehalten werden der Touren- und Zeitenplan, die Anzahl

insgesamt angenommener Anfragen und die Gesamtdauer der Tour der besten Lösung. Eine weitere if-Bedingung ermittelt Fälle, bei denen der aktuelle Tourenplan und der Tourenplan der bisher besten gefundenen Lösung eine gleich hohe Anzahl an angenommenen Anfragen besitzen. Falls dem so ist, wird mit einer weiteren if-Bedingung geprüft, ob die aktuelle Lösung eine geringere Gesamtdauer aufweist als die bisher beste gefundene Lösung. Falls dem so ist, wird die aktuelle Lösung als neue beste Lösung gespeichert und die eben genannten Daten der besten Lösung festgehalten. Falls keine Verbesserung gefunden werden konnte, geschieht nichts. Der aktuelle Tourenplan wird in die nächste Iteration übernommen und wird in dieser durch einen weiteren Destroy- und Repair-Prozess weiterverwertet. Am Ende einer jeweiligen Iteration wird der Iterationszähler um den Wert 1 verringert. Im Anschluss an die while-Schleife in Zeile 67 wird geprüft, ob sich die aktuell eingetroffene Anfrage im Tourenplan der besten gefundenen Lösung befindet. Hierfür wird mit einer if-Bedingung geprüft, ob die aktuelle Anfrage entweder in der Einzeltour des ersten oder in der Einzeltour des zweiten Fahrzeugs befindet. Wenn dem so ist, kann die Anfrage angenommen werden, da sie sich im Tourenplan der besten gefundenen Lösung befindet, die Anfrage wird somit als wertvoll erachtet. Falls die Bedingung nicht erfüllt ist, wird die Annahme der Anfrage als nicht vorteilhaft erachtet und die Anfrage wird folglich abgelehnt. Basierend auf der if-Bedingung gibt die Funktion den Wert *True* an den Funktionsaufruf zurück, falls die Bedingung erfüllt ist. Falls sie nicht erfüllt ist, wird *False* an den Funktionsaufruf zurückgegeben.

4.1.4 Antizipatorisches Routing

Der Ansatz des Antizipatorischen Routings funktioniert hinsichtlich der Auftragsannahmeentscheidung auf dieselbe Art und Weise, wie der Nicht-Antizipatorische Ansatz, beide Ansätze unterscheiden sich jedoch auf der Ebene der Routingentscheidung. Beim Antizipatorischen Routing-Ansatz erfolgt der Routing-Prozess nun unter Berücksichtigung von Anfragen, die in zukünftigen Perioden eintreffen werden. In der im Folgenden vorgestellten beispielhaften Programmierung muss also sichergestellt werden, dass Informationen über zukünftige Anfragen in den Routing-Prozess einfließen können. Gleichzeitig muss gewahrt werden, dass die Entscheidung über die Annahme der Anfragen auf myopische Weise erfolgt. Aus diesen Gründen ist der Programmablauf der Python-Datei *AntRouting.py* bis einschließlich Zeile 177 derselbe, wie beim Nicht-Antizipatorischen Ansatz. Erläuterungen hierzu finden sich im ersten Unterkapitel dieses Hauptkapitels. Das bedeutet, dass auch beim Antizipatorischen Routing-Ansatz für eine jede Frage auf myopische Weise entschieden wird, ob sie angenommen oder abgelehnt wird. Wie bereits bekannt, wird eine jede Anfrage

angenommen, falls sie bedient werden kann. Nachdem alle Anfragen der Beispielinstanz eingetroffen sind, ergibt sich somit nach Durchlauf des Python-Skripts bis Zeile 177 der finale Tourenplan. Um die Verwendung von Antizipation ausschließlich auf die Routing-Ebene beschränken zu können, muss der finale Tourenplan neu konstruiert werden. Auf diese Weise werden bei der Neukonstruktion des Tourenplans nur die Kunden beachtet, über deren Annahme im Vorhinein bereits myopisch entschieden wurde. Gleichzeitig bedeutet das, dass diese Entscheidungen nicht mehr verändert werden können, alle im finalen Tourenplan enthaltenen Anfragen gelten weiterhin als angenommen und müssen somit auch unweigerlich im neu konstruierten Tourenplan enthalten sein. Letztendlich ändern sich im neu konstruierten Tourenplan nur die gefahrenen Strecken und die Ankunftszeiten an den jeweiligen Knoten, jedoch nicht die darin enthaltenen angenommenen Anfragen. Um den finalen Tourenplan in einer endgültigen Routing-Entscheidung verbessern zu können, müssen ab Zeile 182 zuerst neue Variablen eingeführt werden. Die Variablen $reoptTourenplan$ und $reoptZeitenplan$ werden erstellt und erhalten die Daten des finalen Touren- beziehungsweise Zeitenplanes. Die beiden anschließenden Variablen mit dem Namenszusatz *Best* stellen den bisher besten gefundenen Touren- und Zeitenplan dar. Im Initialzustand ist der finale Tourenplan die beste Lösung. Eine weitere Variable dient der Speicherung der Gesamtdauer des besten gefundenen Tourenplans. Die Gesamtdauer der aktuell besten gefundenen Lösung wird in der anschließenden for-Schleife bestimmt. Die Menge der ungeplanten Anfragen bleibt bei der Initialisierung leer, da alle Entscheidungsperioden bereits absolviert worden sind und somit keine neu eingetroffene Anfrage vorliegen kann. Im Laufe des folgenden Destroy- und Repair-Prozess werden Anfragen aus dem Tourenplan entfernt und anschließend wieder eingefügt, wodurch der finale Tourenplan auf mögliche Verbesserungen hinsichtlich des Routings untersucht werden soll. Dieser Prozess läuft 100 Iterationen innerhalb einer while-Schleife ab. Im Destroy-Prozess darf eine jede Anfrage zufällig aus der Tour entfernt werden, diese wird im anschließenden Repair-Schritt zulässig wieder eingefügt. Anschließend wird ab Zeile 265 bestimmt, wie viel Zeit für das Absolvieren des gesamten in dieser Iteration neu konstruierten Tourenplans benötigt wird. Im Anschluss wird mit einer if-Bedingung geprüft, ob die Gesamtdauer des neu konstruierten Tourenplans geringer ist, als die Gesamtdauer des Tourenplans der bisher besten gefundenen Lösung. Falls dem so ist, wird mit zwei weiteren if-Bedingungen geprüft, ob die Einzeltouren der beiden Fahrzeuge dieselbe Länge haben wie die beiden Einzeltouren des finalen Tourenplans. So kann sichergestellt werden, dass alle myopisch angenommenen Anfragen auch im endgültigen Tourenplan enthalten sind. Falls alle 3 Bedingungen erfüllt sind, wurde ein neuer

Tourenplan gefunden, der bei gleichbleibender Anzahl an bedienten Kunden kürzer ist als der bisher beste Tourenplan. Die beste Lösung wird gespeichert, indem der entsprechende Touren- und Zeitenplan und die Gesamtdauer, die für diese Tour benötigt wird, in weiteren Variablen festgehalten wird. Abschließend markiert das Verringern des Iterationszählers den Abschluss einer weiteren Iteration. Nach vollständigem Durchlaufen der while-Schleife kann die Lösung der endgültigen Routingentscheidung in der Variable $reoptTourenplanBest$ eingesehen werden. Dieser enthält dieselben Anfragen wie der Tourenplan der Variable $finalTourenplan$, unterscheidet sich jedoch möglicherweise hinsichtlich der Gesamtdauer, falls bessere Routingentscheidungen für die Fahrzeuge gefunden werden konnten. So wird ermöglicht, dass die individuellen Annahmeentscheidungen myopisch durchgeführt werden können, während im Routingprozess antizipiert werden kann, welche Anfragen myopisch angenommenen werden.

4.1.5 Vollkommen-Antizipatorischer Ansatz

Der Vollkommen-Antizipatorische Ansatz unterscheidet sich grundlegend vom Nicht-Antizipatorischen Ansatz. Bei vollkommener Antizipation erfolgt das Antizipieren von Anfragen sowohl auf der Ebene der Auftragsannahmeentscheidung als auch auf der Ebene der Routing-Entscheidung. Im Programm *FullyAnt.py* müssen nun folglich nicht mehr alle Entscheidungsperioden nacheinander durchlaufen werden, vielmehr sind die genauen geografischen Positionen der Anfragen und ihre jeweiligen Eintreffzeitpunkte bereits zum Zeitpunkt 0 bekannt, da sie antizipiert werden können. Um herausfinden zu können, auf welche Weise sich die Tourenpläne bei einer vollkommenen Antizipation der Anfragen verändern, muss der Code-Abschnitt nach der Einführung der Inputdaten der Beispielinstanz betrachtet werden. Dieser startet mit der Initialisierung der Touren- und Zeitenplänen, diese werden zum Start mit den beiden Fahrzeugstartknoten befüllt, an denen sich die Fahrzeuge zum Zeitpunkt 0 jeweils befinden. Des Weiteren werden ebenfalls Variablen benötigt, welche im Programmablauf die beste gefundene Lösung speichern. Das grundlegende Ziel im Vollkommen-Antizipatorischen Ansatz besteht darin, so viele Anfragen wie möglich anzunehmen. Hierfür ist die Variable $insgesamtAngenommenBest$ von wichtiger Bedeutung, diese speichert die Anzahl der angenommenen Anfragen der besten gefundenen Lösung. Falls sich an der Anzahl angenommener Anfragen nichts ändert, besteht ein weiteres Nebenziel darin, die Gesamtlänge des Tourenplans so klein wie möglich zu halten, die Gesamtdauer der besten gefundenen Lösung wird in $gesamtdauerBest$ gespeichert. Die Menge der nicht eingeplanten Anfragen wird in $ungeplanteAnfragen$ gespeichert, diese

wird zum Start mit allen Anfragen der Beispielinstanz befüllt, da alle Anfragen zum Startzeitpunkt antizipiert werden können. Anschließend führt eine while-Schleife 100 Iterationen des Destroy- und Repair-Prozesses durch, in diesem wird wie bereits geschildert eine zufällige Anfrage aus dem Tourenplan entfernt und in die Menge der ungeplanten Anfragen aufgenommen. Aus dieser Menge werden anschließend so viele Anfragen wie möglich in den Tourenplan integriert, falls dies zulässig ist. Danach wird ab Zeile 134 gezählt, wie viele Anfragen in den Tourenplan, der in der aktuellen Iteration gebildet wurde, aufgenommen werden konnten. Anschließend wird bestimmt, wie viel Zeit für das Absolvieren des aktuellen Tourenplanes benötigt wird. Zum Abschluss einer Iteration kann nun bestimmt werden, ob in der aktuellen Iteration ein Tourenplan gefunden wurde, der eine höhere Anzahl an angenommenen Anfragen vorweist. Ist dies der Fall und somit die erste if-Bedingung erfüllt, dann ist die aktuelle Lösung die neue beste Lösung. Zum Vergleich in den nächsten Iterationen werden der Touren- und Zeitenplan, die Anzahl der insgesamt angenommenen Anfragen und die Gesamtdauer der besten gefundenen Lösung in den entsprechenden Variablen gespeichert. Falls die Anzahl der insgesamt angenommenen Anfragen gleichgeblieben ist, liegt eine neue Lösung vor, falls die innere if-Bedingung erfüllt wird. Diese prüft nämlich, ob der aktuelle Tourenplan eine kürzere Gesamtdauer aufweist als der bisher gefundene beste Tourenplan. Falls dem so ist, stellt der aktuelle Tourenplan eine neue beste Lösung dar, und die entsprechenden Daten können wie eben bereits geschildert in den vorgesehenen Variablen gespeichert werden. Falls keine Verbesserung gefunden werden können, sind alle eben genannten if-Bedingungen nicht erfüllt und werden somit übersprungen. Der Tourenplan der aktuellen Lösung wird somit in der nächsten Iteration weiterverwendet und auf weitere Verbesserungspotentiale untersucht, egal ob sich der Tourenplan verbessert oder verschlechtert hat. So kann ein breiterer Teil des Lösungsraumes untersucht werden. Nach gesamtem Ablauf der while-Schleife aus Zeile 66 kann die Variable $tourenplanBest$ abgerufen werden, diese enthält den besten gefundenen Tourenplan des Vollkommen-Antizipatorischen Ansatzes. Um aus diesen auslesen zu können, welche Anfragen letztendlich angenommen und abgelehnt wurden, wird ein weiterer kurzer Abschnitt benötigt, in dem die Kundenanfragen der Liste $angenommeneAnfragen$ angehängt werden, wenn der jeweilige Kunde im Tourenplan der besten gefundenen Lösung enthalten ist. Falls dem nicht so ist, werden sie schlussendlich abgelehnt und der Liste $abgelehnteAnfragen$ angehängt. Abschließende Print-Statements geben die Ergebnisse des Vollkommen-Antizipatorischen Ansatzes aus.

4.2 Vergleich der Ergebnisse

Im Folgenden können nun die Ergebnisse verglichen werden, die aus der Anwendung der vier unterschiedlichen Ansätze auf die Beispielinstanz resultieren. Die grafischen Darstellungen der Lösungen sind in den jeweiligen Abbildungen einsehbar.

Beim Anwenden des Nicht-Antizipatorischen Ansatzes auf die Beispielinstanz ergibt sich die in Abbildung 2 dargestellte Lösung. In dieser bedient Fahrzeug 1 die Kunden 0, 1 und 2. Die Anfragen 3 und 4, für die das erste Fahrzeug ebenfalls zuständig ist, müssen abgelehnt werden, da das Fahrzeug zuerst Kunde 2 bedienen muss. Eine anschließende Bedienung von Kunde 3 und 4 kann nicht rechtzeitig innerhalb der entsprechenden Zeitfenster erfolgen, eine Annahme dieser beiden Anfragen wäre somit nicht zulässig.

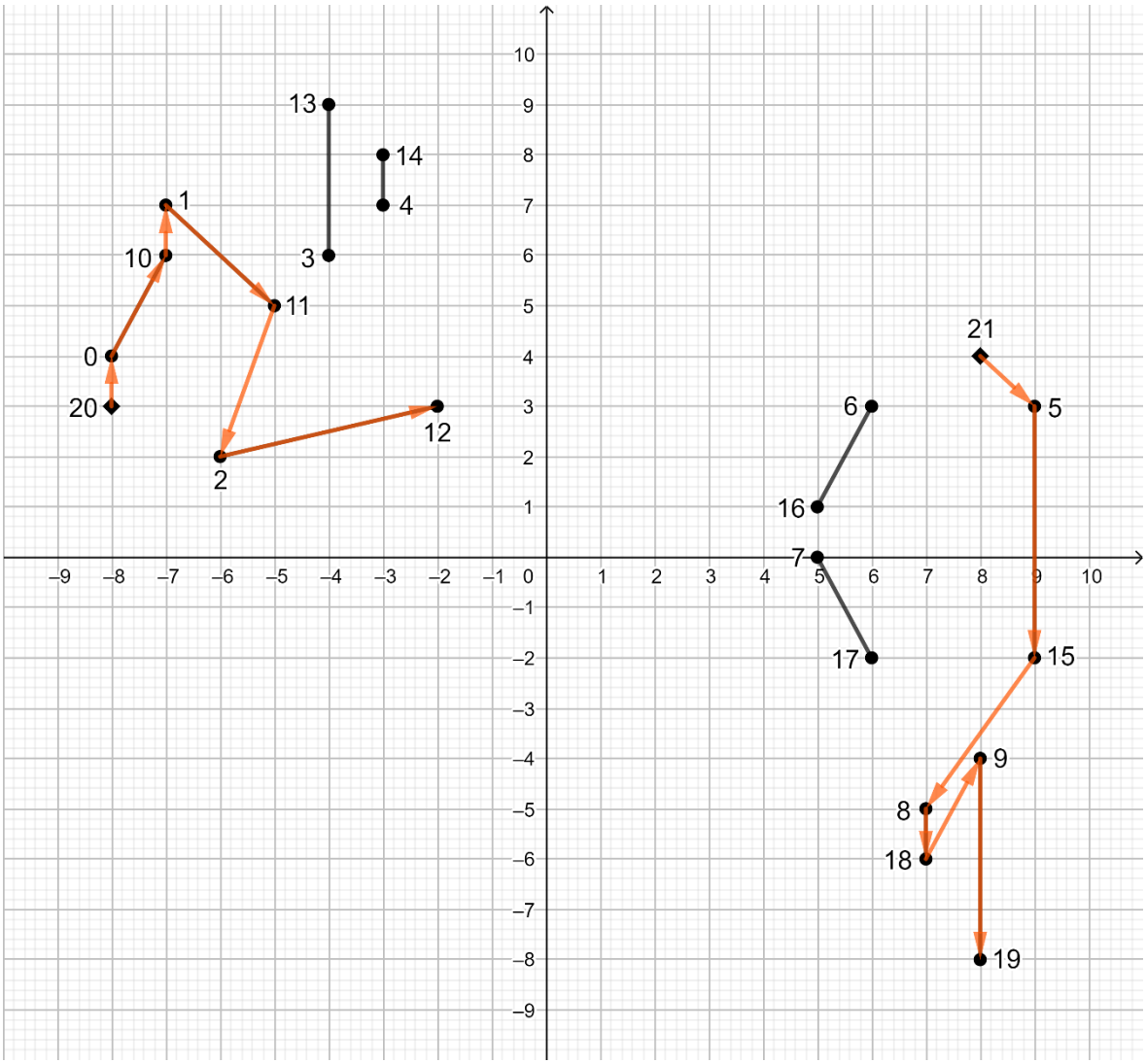

Abbildung 2: Ergebnis des Nicht-Antizipatorischen Ansatzes

Für Fahrzeug 2 kann aus dem Tourenplan abgelesen werden, dass die Kunden 5, 8 und 9 bedient werden. Die Anfragen 6 und 7 müssen abgelehnt werden, da das Fahrzeug zu dieser Zeit mit der Bedienung des fünften Kunden beschäftigt ist. Da die Anfrage 8 zuerst eintrifft, muss zuerst zu Knoten 8 gefahren werden, obwohl Kunde 9 ebenfalls bedient wird und dessen Ursprungsknoten näher an Knoten 15 liegt als der Ursprung des Kunden 8. Grund hierfür ist, dass Anfrage 9 erst 4 Zeiteinheiten nach Anfrage 8 eintreffen wird. Das Modell des Nicht-Antizipatorischen Ansatzes hat zum Zeitpunkt des Eintreffens der achten Anfrage keine Kenntnis darüber, dass die neunte Anfrage in einer zukünftigen Periode eintreffen wird. Deswegen muss zuerst Anfrage 8 vollständig erfüllt werden, bevor der neunte Kunde bedient werden kann.

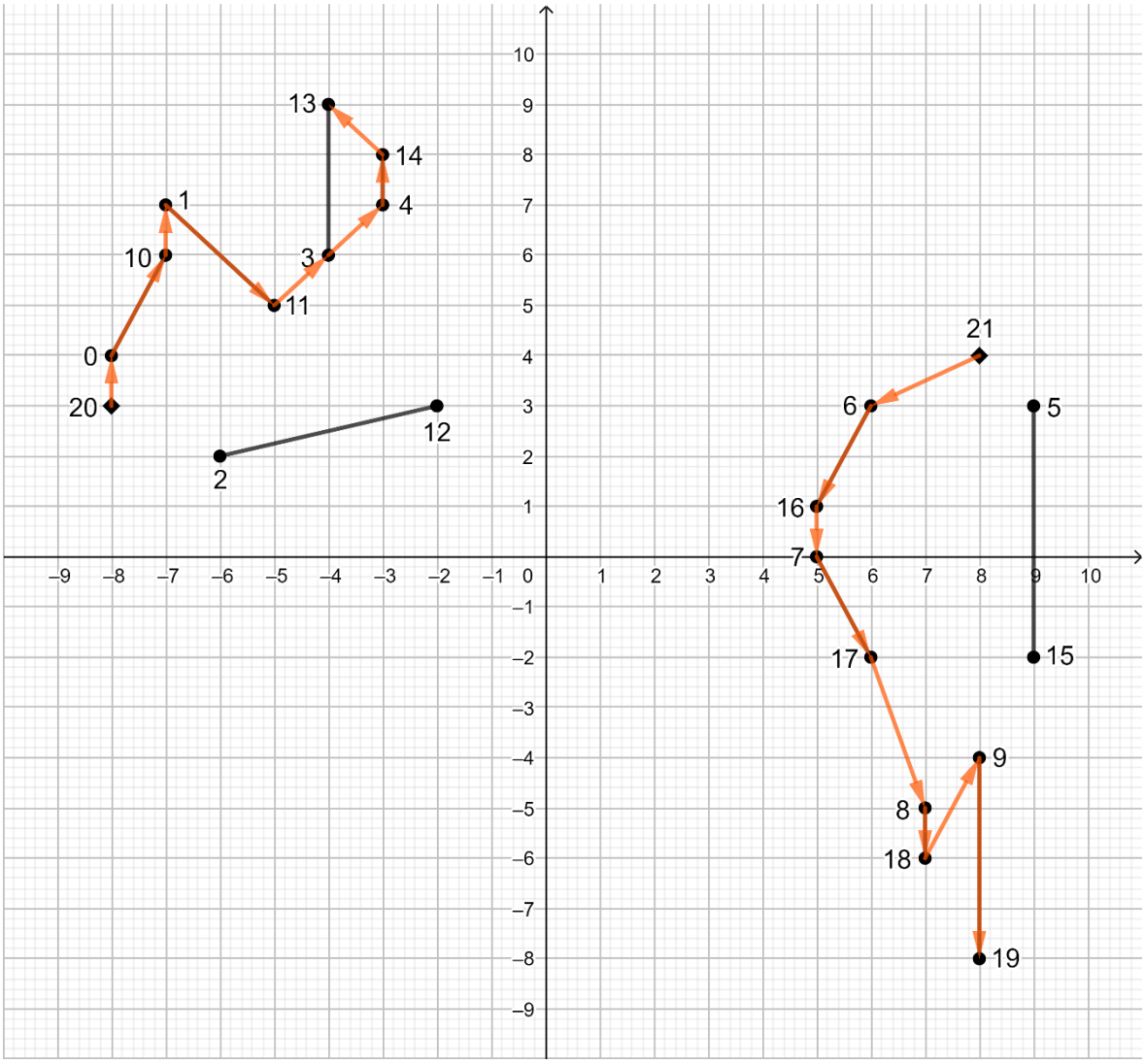

Abbildung 3: Ergebnis des Antizipatorischen Akzeptanz-Ansatzes

Abbildung 3 stellt das Ergebnis des Antizipatorischen-Akzeptanz Ansatzes dar. Aus der Lösung der Beispielinstanz mit diesem Ansatz ergibt sich, dass Fahrzeug 1 die Kunden 0, 1, 3 und 4 bedient. Im Gegensatz zu keiner Antizipation wird nun für eine jede eintreffende Anfrage bestimmt, ob deren Annahme unter Berücksichtigung zukünftig eintreffender Anfragen vorteilhaft ist. Kunde 0 und 1 werden als vorteilhaft erachtet und beide Anfragen werden daher angenommen. Kunde 2 wird als nicht vorteilhaft erachtet und wird daher abgelehnt, obwohl eine Bedienung der Anfrage möglich und zulässig wäre. Das aktive Ablehnen der Anfrage 2 führt dazu, dass die Kunden 3 und 4 bedient werden können. Der Vergleich zum Nicht-Antizipatorischen Ansatz zeigt, dass dort die Anfrage des zweiten Kunden angenommen wurde, was dazu geführt hat, dass die Bedienung der Anfragen 3 und 4 zeitlich gesehen nicht mehr möglich war und diese folglich abgelehnt werden mussten. Mithilfe der Verwendung von Antizipation kann nun zum Zeitpunkt des Eintreffens der zweiten Anfrage antizipiert werden, dass die Anfragen 3 und 4 zu den darauffolgenden Zeitpunkten 9 und 10 eintreffen werden. Das zweite Fahrzeug bedient nun die Kunden 6 bis 9. Hier fällt auf, dass die fünfte Anfrage abgelehnt wurde, obwohl sie hätte bedient werden können. Durch die Ablehnung dieser einen Anfrage ist es nun möglich, zwei weitere Anfragen in den Tourenplan aufzunehmen und somit die Zahl insgesamt angenommener Anfragen zu erhöhen. Die Reihenfolge, in der die Knoten der Kunden 8 und 9 angefahren werden, bleibt hier unverändert. Das liegt daran, dass sich die Verwendung von Antizipation bei diesem Ansatz auf die Ebene der Annahmeentscheidung beschränkt. An den Routingentscheidungen hat sich somit im Vergleich zum Nicht-Antizipatorischen Ansatz nichts verändert.

Der Ansatz des Antizipatorischen Routings kann der Abbildung 4 auf der folgenden Seite entnommen werden. Das Ergebnis dieses Ansatzes zeigt, verglichen mit dem Nicht-Antizipatorischen Ansatz, für den Tourenplan des ersten Fahrzeugs keine Veränderungen auf. Auch hier werden die Kunden 0,1 und 2 bedient und die Anfragen 4 und 5 abgelehnt. Bei der Betrachtung des Zeitenplans zeigen sich jedoch erste Änderungen. Das erste Fahrzeug kommt nun nach 2 Zeiteinheiten am Ursprungsknoten 0 an, gleichzeitig trifft die Anfrage 0 zum Zeitpunkt 2 ein. Das liegt daran, dass die Anfrage des Kunden 0 antizipiert werden konnte. Das erste Fahrzeug ist zu Beginn des Modelldurchlaufs untätig und benötigt nur eine Zeiteinheit, um zum Ursprungsknoten 0 zu gelangen. Somit fährt das Fahrzeug zum Ursprungsknoten 0, obwohl die Anfrage 0 noch nicht eingetroffen ist, damit der Kunde direkt zu Beginn des Bedienzeitfensters, welches mit dem Eintreffzeitpunkt der Anfrage beginnt, bedient werden kann. Deshalb können auch die darauffolgenden Kunden des Tourenplans

zu früheren Zeitpunkten bedient werden. Letztendlich ist die gesamte Tour des ersten Fahrzeugs beim Antizipatorischen Routing um eine Zeiteinheit kürzer, obwohl dieselben Wege gefahren werden, wie beim Nicht-Antizipatorischen Ansatz. Der Blick auf das zweite Fahrzeug zeigt, dass auch hier nach wie vor die Kunden 5, 8 und 9 bedient werden. Der Ankunftszeitpunkt am Knoten 5 ist nun zum Zeitpunkt 11 und somit ebenfalls früher als beim Nicht-Antizipatorischen Ansatz, bei dem die Ankunft zum Zeitpunkt 12.41 erfolgt.

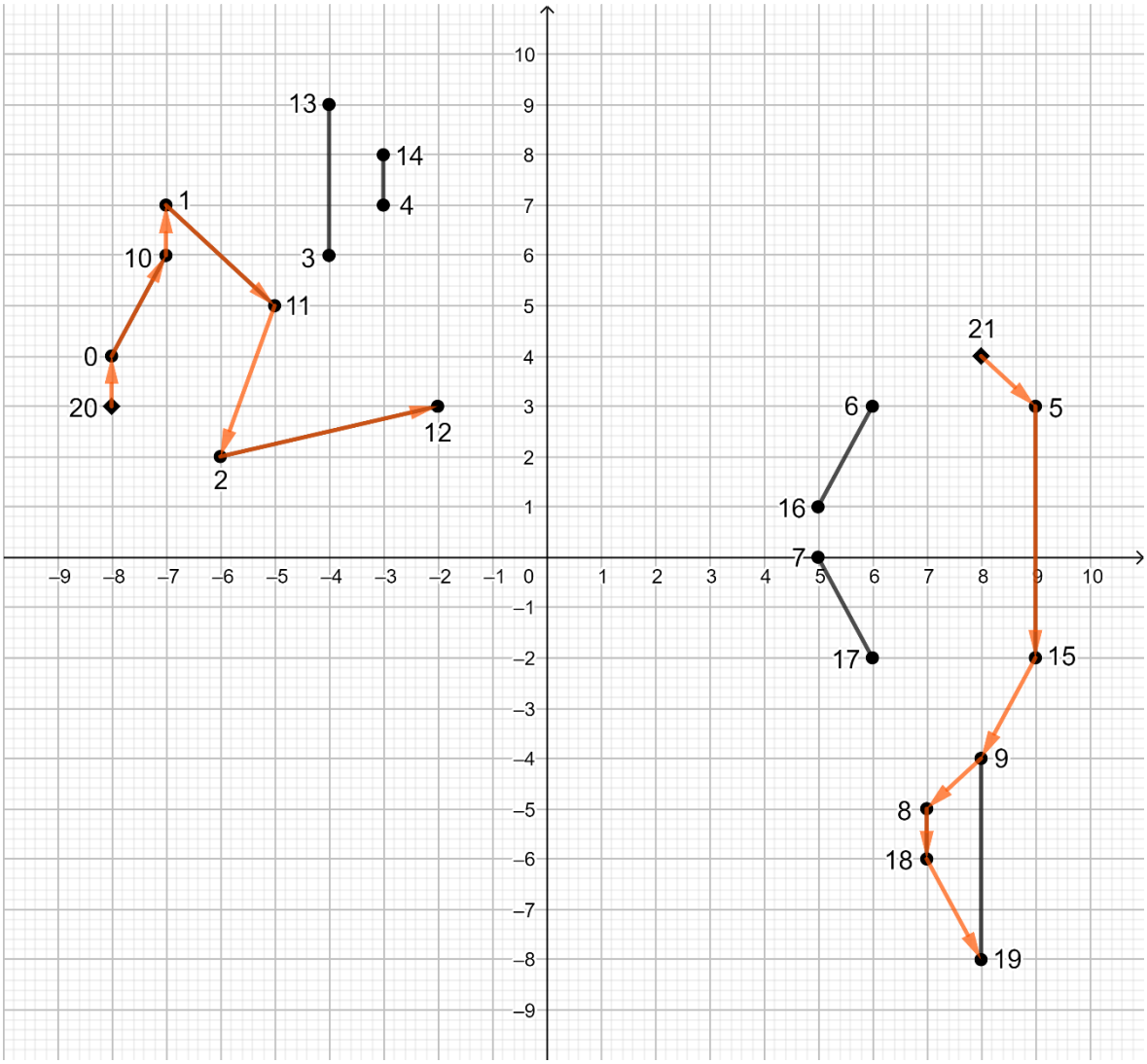

Abbildung 4: Ergebnis des Antizipatorischen Routings

Bei der Bedienung der Kunden 8 und 9 zeigt sich allerdings nun eine veränderte Reihenfolge, hier wird zuerst zum Kunden 9 gefahren, anschließend zum Kunden 8. Dass diese Reihenfolge aus Routing-Sicht besser ist, zeigt der Vergleich mit dem Zeitenplan des Nicht-Antizipatorischen Ansatzes. Obwohl beim Antizipatorischen Routing auf den Kunden 9 gewartet wird, dessen Anfrage 4 Zeiteinheiten nach der achten Anfrage eintrifft, beendet das

zweite Fahrzeug seine Tour bereits zum Zeitpunkt 26.65, während die zweite Tour bei keiner Antizipation erst zum Zeitpunkt 28.84 abgeschlossen wird. Somit zeigt dieser Ansatz deutlich, welche Verbesserungspotentiale in der Verwendung von Antizipation auf der Ebene des Routings liegen.

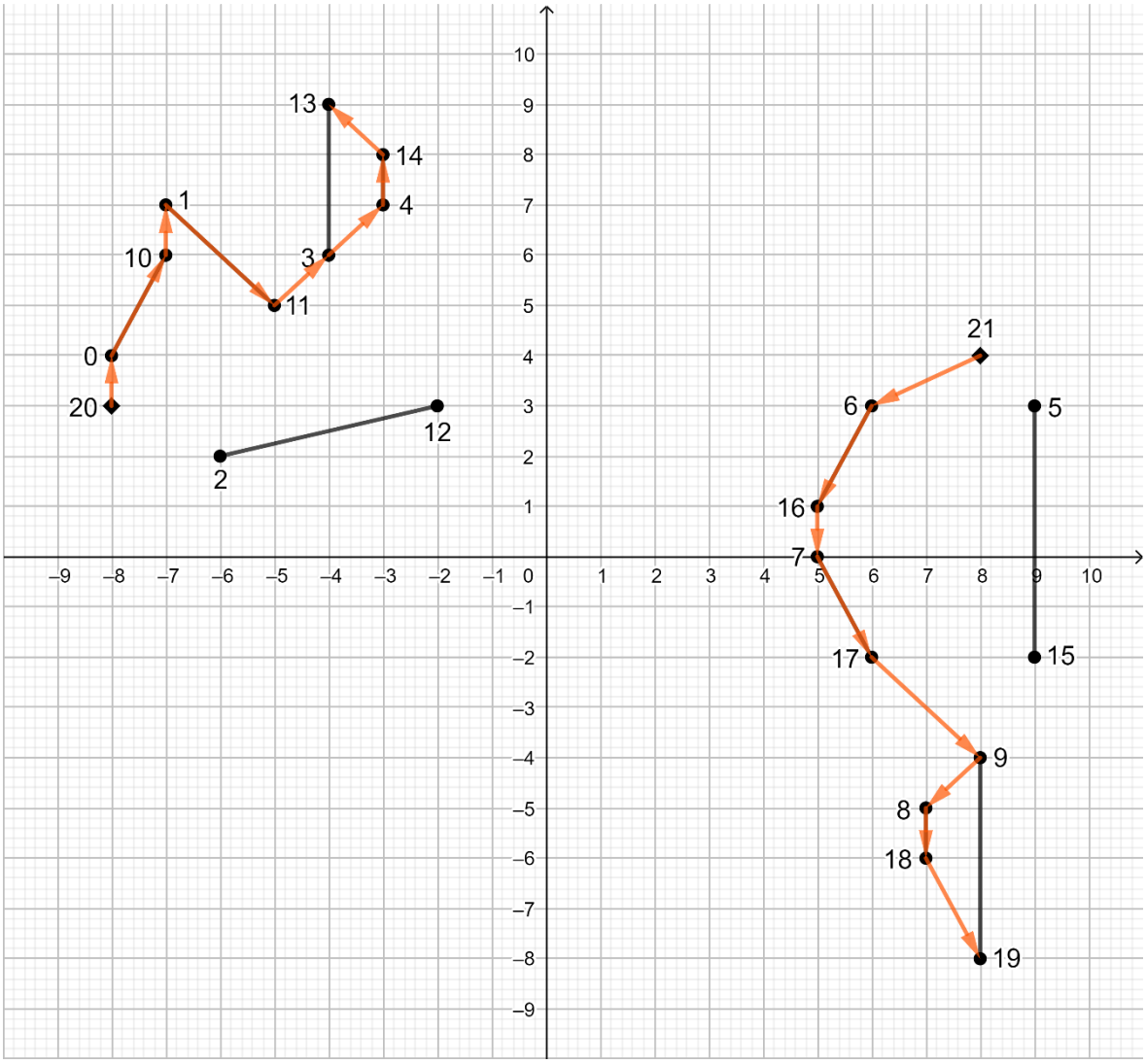

Abbildung 5: Ergebnis des Vollkommen-Antizipatorischen Ansatzes

Die Anwendung des Volllkommen-Antizipatorischen Ansatzes auf die Beispielinstanz führt dazu, dass das erste Fahrzeug die Kunden 0, 1, 3 und 4 bedient. Die Reihenfolge der Knoten im Tourenplan ist dieselbe wie beim Ansatz der Antizipatorischen Akzeptanz, beide Ansätze schätzen also dieselben Anfragen als wertvoll ein. Allerdings unterscheiden sich die Zeitenpläne beider Ansätze, hier macht sich beim Vollkommen-Antizipatorischen Ansatz die Verwendung von Antizipation auf beiden Ebenen, also der Ebene der Annahme- und der

Routingentscheidung, bemerkbar. Kunde 0 kann bei vollkommener Antizipation zum Zeitpunkt 2 und damit 1 Zeiteinheit früher bedient, als bei antizipatorischer Akzeptanz. Insgesamt betrachtet endet die erste Tour bereits zum Zeitpunkt 13.31 und damit 1 Zeiteinheit früher als bei antizipatorischer Akzeptanz, wo die erste Tour erst zum Zeitpunkt 14.31 beendet werden kann. Bei der zweiten Tour werden die Kunden 6, 7, 8 und 9 bedient, auch hier erachten beide Ansätze dieselben Anfragen als wertvoll. Auch hier können die Knoten der Kunden wieder zu früheren Zeitpunkten angefahren werden, als das bei der Antizipatorischen Akzeptanz der Fall ist, da auf beiden Entscheidungsebenen antizipiert werden kann. Die Bedienung des Kunden 6 startet zum Zeitpunkt 13 und damit früher als bei Antizipatorischer Akzeptanz mit dem Zeitpunkt 15.24. Damit dauert die Bedienung der Kunden 6 und 7 insgesamt kürzer, obwohl bei beiden betrachteten Ansätzen genau dieselben Wegstrecken gefahren werden. Des Weiteren fällt beim Vergleich beider Ansätze auf, dass sich hier nun die Reihenfolge der Knoten im Tourenplan ändert, denn durch die vollkommene Antizipation können jetzt die Informationen über zukünftig eintreffende Anfragen auch in den Routing-Prozess einfließen. Dies führt dazu, dass zuerst zum Ursprung von Kunde 9 und anschließend zum Ursprung von Kunde 8 gefahren werden kann, bei der Antizipatorischen Akzeptanz ist die Reihenfolge andersherum. Dass die neu kreierte Reihenfolge aus Routing-Sicht betrachtet sehr sinnvoll ist, zeigt ein Blick auf die Zeitenpläne der beiden Ansätze. Hier vergrößert sich durch das veränderte Routing der Kunden 8 und 9 die zeitliche Lücke zwischen beiden Ansätzen, in Knoten 17 beträgt der Unterschied in den Ankunftszeiten 2.24 Zeiteinheiten. Am Ende der Tour des zweiten Fahrzeugs, und somit nach der Bedienung der Kunden 8 und 9, vergrößert sich der zeitliche Abstand beider Tourenpläne deutlich, dieser Abstand beträgt nun 4.46 Zeiteinheiten. Wie man sehen kann, werden beim Vollkommen-Antizipatorischen Ansatz genau die gleichen Anfragen wie beim Ansatz der Antizipatorischen Akzeptanz angenommen, ersterer absolviert die Tour jedoch in deutlich kürzerer Zeit.

4.3 Vorteile antizipativer Lösungsverfahren

Im Folgenden sollen nun die Vorteile erläutert werden, die aus einer Verwendung von Antizipation in der Entscheidungsfindung resultieren.

4.3.1 Maximierung der Gesamtzahl angenommener Anfragen

Wie in den Beispielimplementierungen gezeigt werden konnte, führt das Antizipieren von Anfragen dazu, dass die Gesamtzahl aller angenommenen Anfragen über den gesamten Planungshorizont hinaus maximiert werden kann. Das wird allein schon bei der Betrachtung

der Ergebnisse aller Ansätze deutlich. Während bei keiner Antizipation insgesamt 6 Fragen angenommen werden, liegt die Anzahl angenommener Anfragen bei der Antizipatorischen Akzeptanz und der Vollkommenen Akzeptanz bei 8 und damit deutlich höher. Eine Ausnahme bietet hierbei die Antizipatorische Akzeptanz mit 6 Anfragen, diese wird in dieser Beispielimplementierung allerdings etwas vereinfacht umgesetzt. Ein deutlich komplexerer realitätsnäherer Programmieransatz würde vermutlich ebenfalls zu einer erhöhten Anzahl an angenommenen Anfragen führen. Auch bei der Entscheidung zwischen einzelnen Anfragen wird die erhöhte Anzahl angenommener Anfragen deutlich. Hierfür müssen die Anfragen 5, 6 und 7 betrachtet werden. Bei keiner Antizipation wird die Anfrage 5 angenommen, die Anfragen 6 und 7 müssen abgelehnt werden, da zu wenig Zeit verbleibt, um diese beiden Anfragen bedienen zu können. Bei Antizipatorischer Akzeptanz und Vollkommener Antizipation werden beide Anfragen jedoch angenommen, dafür muss Anfrage Nummer 5 aktiv abgelehnt werden. Hier ist ersichtlich, dass bei der Verwendung von Antizipation im Zweifelsfall so entschieden wird, dass über den gesamten Planungshorizont gesehen eine maximale Zahl an Anfragen akzeptiert werden kann.

4.3.2 Realisierung von Zeitgewinnen

Zusätzlich ist ein möglicher Zeitgewinn zu beachten, der durch die Verwendung von Antizipation in der Entscheidungsfindung realisiert werden kann. Dieser Aspekt ist vor allem bei antizipativen Routing-Strategien zu sehen, in den Beispielimplementierungen konnte nämlich gezeigt werden, dass Antizipation im Routing zu einer kürzeren Fahrzeit führt. Diese verkürzte Fahrtzeit resultiert zum einen aus dem aktiven früheren Anfahren der jeweiligen Knoten, durch die ein Knoten früher angefahren und wieder verlassen werden kann. Zum anderen führt das Antizipieren von Anfragen zu Veränderungen innerhalb des Tourenplans, da nun hinsichtlich zukünftig eintreffender Anfragen sinnvollere Strecken gefahren werden können. Aber auch auf Seiten der Annahmeentscheidung lässt sich durch das Verwenden von Antizipation Zeit gewinnen, wie im Ansatz der Antizipatorischen Akzeptanz zu sehen ist. Hier wird die Anfrage des zweiten Kunden aktiv abgelehnt, damit die Kunden 3 und 4 bedient werden können. Im Nicht-Antizipatorischen Ansatz ist zu sehen, dass eine Annahme des Kunden 2 zu einer Ablehnung der Kunden 3 und 4 führt, da zu wenig Zeit für deren Bedienung zur Verfügung steht. Durch das aktive Ablehnen der unvorteilhaften Anfrage konnte somit Zeit für die Bedienung weiterer Anfragen gewonnen werden. Insgesamt betrachtet ist das Gewinnen von Zeit natürlich sehr vorteilhaft, da somit mehr Zeit für die Annahme weiterer Anfragen zur Verfügung steht. Gleichzeitig bedeutet das schnellere

Absolvieren von Touren auch, dass die Kunden schneller an ihr Ziel gebracht werden können und somit weniger Zeit im Fahrzeug verbringen müssen.

4.3.3 Anwendbarkeit auf unterschiedlichen Entscheidungsebenen

Ein weiterer vorteilhafter Aspekt ist, dass das Verwenden von Antizipation nicht nur auf eine Ebene beschränkt ist. Wie die Beispielimplementierungen zeigen, kann Antizipation auf unterschiedlichen Entscheidungsebenen erfolgen, sowohl in der Annahmeentscheidung, in der Routingentscheidung, als auch in beiden gleichzeitig. Wie Haferkamp und Ehmke (2020) betonen, können die vorgestellten Ansätze für unterschiedlichste Szenarios in der Praxis angewendet werden. So eignet sich der Antizipatorische Akzeptanz-Ansatz beispielsweise für eine Anwendung im Zentrum einer Innenstadt, in dem eher kurze Strecken gefahren werden, aber viele Anfragen eintreffen werden. Das Antizipatorische Routing würde sich zum Beispiel in Szenarien eignen, in denen eine größtenteils fixe Kundenmenge bedient werden muss (vgl. Haferkamp und Ehmke (2020, S. 21f)). Wie man sieht, sind antizipative Strategien flexibel und können auf unterschiedliche Weisen in die Entscheidungsprozesse eines DDARP integriert werden.

4.4 Nachteile antizipativer Strategien

Neben den unbestreitbaren Vorzügen antizipativer Strategien müssen allerdings auch die Nachteile der Anwendung derartiger Strategien beachtet werden. Diese sollen im Folgenden erläutert werden.

4.4.1 Diskriminierungsproblem

Das aktive Ablehnen von Anfragen, was mit der Antizipation zukünftig eintreffender Anfragen einhergeht, führt zu einem Diskriminierungsproblem. Es kann nämlich beobachtet werden, dass Kunden aufgrund der spezifischen Eigenschaften ihrer Anfragen benachteiligt werden. Das ist auch in den Ergebnissen der Beispielinstanzen ersichtlich. Anfrage Nummer 2 wird beim Nicht-Antizipatorischen Ansatz angenommen, da die Bedienung zulässig ist. Die Verwendung von Antizipation auf der Ebene der Annahmeentscheidung führt allerdings dazu, dass Anfrage Nummer 2 als nicht wertvoll erachtet wird, da sie die Annahme der Anfragen 3 und 4 verhindert. Kunde 2 hat seine Anfrage also eigentlich zur richtigen Zeit gestellt, um befördert werden zu können, wird aber als minderwertig erachtet, da zu viel Zeit durch dessen Bedienung in Anspruch genommen wird. Kunde Nummer 2 wird folglich

aufgrund anfragenspezifischer Eigenschaften diskriminiert. Diesen Umstand haben auch Haferkamp und Ehmke untersucht, beide konnten zeigen, dass durch das Anwenden von Antizipation definitiv ein Diskriminierungsproblem entsteht (vgl. Haferkamp und Ehmke (2020, S. 19f)). Das wiederholte Ablehnen solcher Anfragen kann zum einen dazu führen, dass solche Anfragen in der Zukunft nicht mehr gestellt werden. Zum anderen könnte das aktive Ablehnen von Anfragen dazu führen, dass Kunden verärgert und den Fahrdienst nicht mehr nutzen werden (vgl. Haferkamp und Ehmke (2020, S. 7)).

4.4.2 Unsicherheit der Nachfrage

Ein weiterer Aspekt, der in der hier vorgestellten vereinfachten Modellwelt nicht abgebildet werden kann, ist die Unsicherheit der Nachfrage. In den Beispielmodellen wird jede Anfrage, die antizipiert werden kann, perfekt vorhergesehen. Das heißt, dass der genaue Ursprungs- und Zielort und der genaue Eintreffzeitpunkt der Anfrage antizipiert werden kann. Es steht zusätzlich auch noch fest, dass diese Anfrage sicher eintreffen wird. Doch in der Praxis wäre ein Szenario, in dem Anfragen perfekt vorhergesehen werden können, nicht realistisch. Hier besteht unter anderen das Risiko, dass eine vorhergesehene und eingeplante Anfrage doch nicht eintreffen wird. Gleichzeitig könnte eine eintreffende Anfrage hinsichtlich ihrer Position oder Eintreffzeit variieren, eine Vorhersage ist somit nicht so einfach möglich, wie es hier in der vereinfachten Modellwelt der Fall ist. Die zentrale Frage ist hierbei, wie mit solchen Situationen umgegangen wird. Denkbar wäre beispielsweise auch, dass die zukünftige Anfrage unterschätzt wird. Das heißt, dass in der Gegenwart zu viele unvorteilhafte Anfragen angenommen werden, und zukünftig ausreichende Anfragen nicht ausreichend bedient werden können, da keine Kapazität mehr vorhanden ist. Andersrum könnte die zukünftige Nachfrage überschätzt werden, dies würde bedeuten, dass in der Gegenwart zu viele Anfragen abgelehnt werden, um Kapazitäten für zukünftige Perioden freizuhalten. Falls in der Zukunft allerdings nicht so viele Anfragen eintreffen wie geplant, mündet dies in einer schlechteren Gesamtperformance des Systems (vgl. Haferkamp und Ehmke (2020, S. 7)).

4.4.3 Zunehmende Komplexität der Modelle

Ein letzter negativer Aspekt zeigt sich in der zunehmenden Komplexität, die mit dem Erweitern des Grundmodells um antizipatorische Aspekte einhergeht. Während die Antizipation auf der Ebene der Akzeptanzentscheidung vergleichsweise geringe Probleme

verursacht, sind antizipative Routing-Strategien schon deutlich komplexer. Das Problem besteht darin, dass das Routing im DDARP für große Modellinstanzen bereits herausfordernd genug ist. Das zusätzliche Antizipieren von Anfragen erhöht die Schwierigkeit in der Umsetzung solcher Modelle erheblich. Beim Vollkommen-Antizipatorische Ansatz wird das Antizipieren auf der Annahme- und der Routing-Entscheidungsebene gleichzeitig verwendet, dieser Ansatz stellt somit die technisch gesehen größte Herausforderung dar (vgl. Haferkamp und Ehmke (2020, S. 7)).

5 Schlussfolgerungen und Ausblick

Im Rahmen dieser Masterarbeit wurde ein beispielhafter Einblick gegeben, wie das Verwenden von Antizipation in der Entscheidungsfindung im Rahmen einer Problemstellung aus dem Bereich der Shared Mobility Systeme erfolgen kann. Hierbei stand insbesondere die Herangehensweise von Haferkamp und Ehmke (2020) an dieses Thema im Fokus. Es wurde ausführlich auf die von den Autoren vorgeschlagenen unterschiedlichen Antizipationsansätze eingegangen. Neben der Erläuterung der theoretischen Grundlagen wurden die verschiedenen Ansätze anhand eines eigens erstellten Beispiels didaktisch aufbereitet. So konnten die Ergebnisse der Ansätze anschaulich aufgezeigt und miteinander verglichen werden. Hierbei wurde auf die Vorzüge, aber auch auf die Probleme der Verwendung antizipativer Strategien im Entscheidungsprozess eingegangen.

In Kapitel 2 wurde ein kleiner Literaturüberblick gegeben. Hier fällt auf, dass in den letzten Jahren immer mehr interessante Beiträge zur antizipatorischen Entscheidungsfindung erschienen sind. Das Thema ist somit hochaktuell und in Zukunft können aus diesem Bereich noch weitere innovative antizipatorische Lösungsansätze erwartet werden.

Das dritte Kapitel widmete sich der Vorstellung eines dynamisch-stochastischen Optimierungsproblems. Hier stand der von Haferkamp und Ehmke (2020) vorgeschlagene Modellierungsansatz im Mittelpunkt. Wichtig war es hierbei, die theoretischen Grundlagen dieses komplexen Modells einzuführen und verständlich zu erklären. Hierzu wurde neben dem DDARP der MDP geschildert, dieser stellt das grundlegende Schema dar, nach dem der Modellierungsansatz abläuft. Anschließend wurde erklärt, wie das Verwenden von Antizipation in diesem Modell konkret erfolgen kann. Zuletzt wurde eine LNS eingeführt, mit der die unterschiedlichen Antizipationsansätze einheitlich gelöst und somit miteinander verglichen werden können. Neben dem grundlegenden Ablauf einer LNS wurde hierbei für einen jeden Antizipationsansatz aufgezeigt, wie dieser mit einer LNS gelöst werden kann.

In Kapitel 4 stand eine beispielhafte Implementierung der unterschiedlichen Antizipationsstrategien in der Programmiersprache Python im Fokus. Hierfür wurden die vorgestellten Ansätze angewendet, um eine eigens erstellte Beispielinstanz zu lösen. Anschließend konnte auf die unterschiedlichen Ergebnisse der jeweiligen Ansätze eingegangen und diese miteinander verglichen werden. Im Rahmen dieser Ergebnisse konnte auch diskutiert werden, welche Vor- und Nachteile sich aus der Verwendung antizipativer Strategien ergeben. Zu den Implementierungen ist jedoch zu sagen, dass diese an einigen Stellen deutlich vereinfacht werden mussten, da sich dort das Implementieren als zu komplex herausstellte. Nichtsdestotrotz gelang es, für einen jeden Ansatz die Unterschiedlichkeiten in den Ergebnissen herauszustellen. Jedoch muss beachtet werden, dass der vorgestellte Modellierungsansatz eine Vereinfachung der Realität ist. Zukünftige Arbeiten auf diesem Gebiet könnten sich beispielsweise darauf fokussieren, ein deutlich realitätsnäheres und somit praxistaugliches Modell zu erschaffen.

Generell lässt sich beobachten, dass das Thema der antizipatorischen Entscheidungsfindung in Shared Mobility Systemen sehr zukunftsrelevant ist. Neben dem bereits geschilderten steigerten Interesse in der Wissenschaft ist auch in der Praxis zu erwarten, dass solche neuen Mobilitätsformen zunehmend relevanter werden. Hier spielt zum einen der technische Fortschritt eine Rolle, autonom fahrende Autos könnten beispielsweise für einen Rideselling-Anbieter zu einer gesteigerten Kosteneffizienz führen. Die Kosten wurden im Rahmen des in dieser Arbeit vorgestellten Modells überhaupt nicht betrachtet, sind aber für Umsetzungen in der Praxis von großer Bedeutung. Gleichzeitig könnte auch angesichts des sich verschärfenden Klimawandels der Anpassungsdruck von Seiten der Politik auf die Mobilitätsbranche erhöht werden. So könnten neue Mobilitätsformen wie das Rideselling möglicherweise von staatlichen Subventionierungen profitieren.

Letztendlich ist zu sagen, dass die Herangehensweise von Haferkamp und Ehmke (2020) sehr schön darstellt, wie Antizipation auf unterschiedlichen Entscheidungsebenen angewendet werden kann. Das Gebiet der Antizipation wird in Zukunft zunehmend wichtiger werden. Es ist deshalb zu erwarten, dass zukünftig weitere Nachforschungen in diesem Themengebiet stattfinden werden und viele Modelle um den Aspekt der Antizipation erweitert oder gänzlich neue Modellierungsansätze vorgestellt werden.

Literaturverzeichnis

Alonso-Mora, J.; A. Wallar und D. Rus (2017): Predictive routing for autonomous mobility-on-demand systems with ride-sharing. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, S. 3583–3590.

Chao, I.-M.; B. L. Golden und E. A. Wasil (1996): The team orienteering problem. European Journal of Operational Research, Volume 88, S. 464-474.

Ferrucci, F.; S. Bock und M. Gendreau (2013): A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. European Journal of Operational Research, Volume 225, Issue 1, S. 130-141.

Guo, Y.; Y. Zhang, J. Yu und X. Shen (2020): A Spatiotemporal Thermo Guidance Based Real-Time Online Ride-Hailing Dispatch Framework. IEEE Access, Vol. 8, S. 115063-115077.

Haferkamp, J. und J. F. Ehmke (2020): Evaluation of Anticipatory Decision-Making in Ride-Sharing Services. Working Paper Series, 2020, Working Paper No. 4.

Horn, M.E.T. (2002): Fleet scheduling and dispatching for demand-responsive passenger services. Transportation Research Part C 10, 2002, S. 35–63.

Hosni H.; J. Naoum-Sawaya und H. Artail (2014): The shared-taxi problem: Formulation and solution methods. Transportation Research Part B: Methodological, Volume 70, S. 303–318.

Ichoua, S.; M. Gendreau und J.Y. Potvin (2006): Exploiting Knowledge About Future Demands for Real-Time Vehicle Dispatching. Transportation Science, Volume 40, No. 2, S. 211–225.

Lois, A.; A. Ziliaskopoulos und S. Tsalapatas (2018): Evaluation of Probabilistic Demands Usage for the Online Dial-a-Ride Problem. In: Nathanail, E. und I. Karakikes (Hrsg.): Data Analytics: Paving the Way to Sustainable Urban Mobility. CSUM 2018, Advances in Intelligent Systems and Computing, Vol. 879, S. 437- 444.

Potvin, J.-Y. und J.-M. Rousseau (1993): A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research, Volume 66, Issue 3, S. 331-340.

Pureza, V. und G. Laporte (2008): Waiting and Buffering Strategies for the Dynamic Pickup and Delivery Problem with Time Windows. INFOR: Information Systems and Operational Research, Volume 46, Issue 3, S. 165-175.

Ropke, S. und D. Pisinger (2006): An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science, Vol. 40, No. 4, S. 455-472.

Schilde, M.; K.F. Doerner und R.F. Hartl (2011): Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. Computers & operations research, Volume 38, Issue 12, S. 1719–1730.

Shaw, P. (1998): Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: Maher, M. und JF. Puget (Hrsg.): Principles and Practice of Constraint Programming — CP98. Vol. 1520, Springer, Berlin, Heidelberg, S. 417-431.

Wei, C.; Y. Wang, X. Yan und C. Shao (2018): Look-Ahead Insertion Policy for a Shared-Taxi System Based on Reinforcement Learning. IEEE Access, Vol. 6, S. 5716-5726.

Xiang, Z.; C. Chu und H. Chen (2008): The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments. European Journal of Operational Research, Volume 185, Issue 2, S. 534–551.

Zwiers, J.; L. Büttner, S. Behrendt, I. Kollosche, C. Scherf, S. Mader und W. Schade (2021): Wandel des öffentlichen Verkehrs in Deutschland: Veränderung der Wertschöpfungsstrukturen durch neue Mobilitätsdienstleistungen. Eine Transformationsanalyse aus der Multi-Level-Perspektive. Studie der Hans-Böckler-Stiftung, No. 451, Hans-Böckler-Stiftung, Düsseldorf.

Schriftliche Versicherung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbständig und ohne fremde Hilfe verfasst wurde. Alle Stellen, die ich wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Quellen übernommen habe, habe ich als solche gekennzeichnet. Es wurden alle Quellen, auch Internetquellen, ordnungsgemäß angegeben.

Ort, Datum

Verfasser