UNIVERSITÄT
LEIPZIG

Institut für Informatik
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

Tabu Search Algorithmus für das Vehicle Routing Problem eines Logistikunternehmens

Bachelorarbeit

vorgelegt von:
Lea Christin Löffelmann

Matrikelnummer:
3709697

Betreuer:
Prof. Dr. Erhard Rahm

© 2022

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Zusammenfassung

Das Logistikunternehmen fox-Courier beliefert am Tag mit zwei bis vier Fahrzeugen etwa 25-30 Auftraggebende im Raum Leipzig und plant täglich manuell die entsprechenden Auslieferungsrouten. In dieser Arbeit wird eine Anwendung für die Optimierung der Routenplanung des Unternehmens vorgestellt. Für das vorliegende *Vehicle Routing Problem* mit Kapazitätsbeschränkungen, Zeitfenstern sowie weiteren Nebenbedingungen wird der implementierte Optimierungsalgorithmus präsentiert, welcher auf der Tabu-Suche basiert und um verschiedene Erweiterungen, wie einen reaktiven Mechanismus, einen Neustart-Mechanismus sowie multiple Nachbarschaften, ergänzt wurde. Zur entwickelten Anwendung gehört außerdem eine Benutzeroberfläche, welche die Eingabe der Auftragsdaten ermöglicht und in der die von der Tabu-Suche berechneten Routen dargestellt werden.

Die Evaluierung des implementierten Tabu-Suchalgorithmus anhand von Benchmark-Daten zeigt, dass der Algorithmus in der Lage ist, die optimale Lösungen für die meisten der Instanzen zu finden, diese allerdings nicht immer zuverlässig in jedem Durchlauf gefunden werden. Allerdings kann bei der Evaluierung mit Daten des Unternehmens festgestellt werden, dass der Algorithmus für diese Daten kürzere und schnellere Routen als die manuell geplanten Routen findet und dies in deutlich kürzerer Zeit als für die manuelle Planung benötigt wird, womit der potenzielle Nutzen der Anwendung für das Unternehmen gezeigt wird.

Lea Christin Löffelmann
3709697

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

In der heutigen globalisierten Welt ist der Austausch von Waren ein zentraler Bestandteil der Wirtschaft. Der Transport der Waren erfordert aufgrund des hohen damit verbundenen Planungsaufwands, dem Konkurrenzdruck zwischen Logistikdienstleistern und der Dringlichkeit, durch den Transport entstehende Emissionen zu vermindern, eine effiziente Planung der Lieferwege. Das Reduzieren der Länge von Transportwegen und Fahrzeiten sowie das Verringern der Anzahl der verwendeten Fahrzeuge kann einem Unternehmen erhebliche Einsparungen von Fahrt- sowie Personalkosten bringen, den geleisteten Service verbessern und dadurch die Wettbewerbsfähigkeit erhöhen. Zusätzlich können durch kürzere Lieferrouten schädliche Emissionen der Fahrzeuge reduziert werden.

Routingprobleme dieser Art werden als *Vehicle Routing Problems* (VRP) bezeichnet. Das VRP zählt zu den kombinatorischen Optimierungsproblemen und ist eine Generalisierung des bekannten *Travelling Salesman Problem* (TSP). Bei der Lösung des VRP geht es darum, ausgehend von einer Menge von Aufträgen und einer Flotte von Fahrzeugen, welche an einem Depot starten und enden, eine Menge von Routen (eine Route für jedes Fahrzeug) so zu bestimmen, dass die Gesamtentfernung bzw. die Fahrzeit minimiert werden.

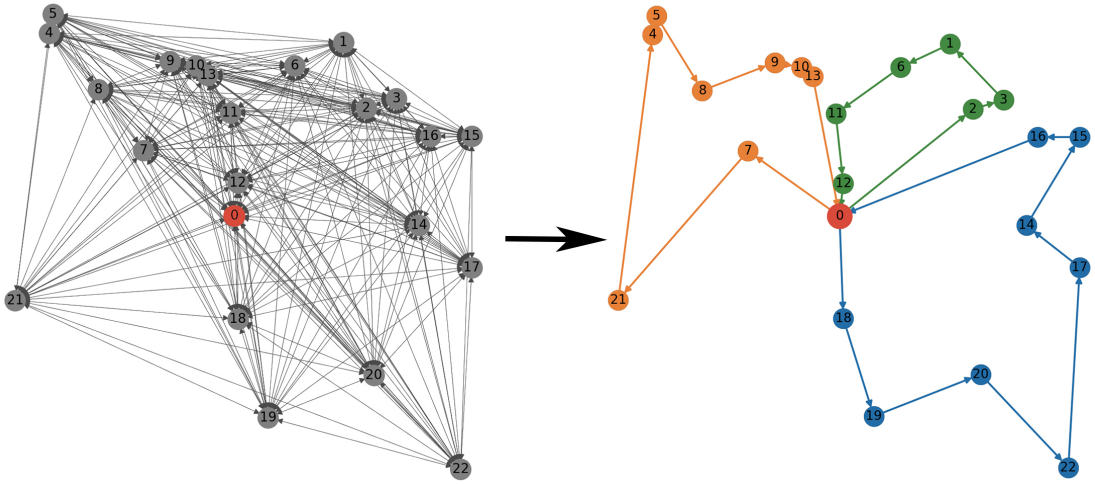


Abbildung 1.1.: Beispiel eines *Vehicle Routing Problems*

Das VRP sowie seine Variationen gehören zur Klasse der NP-schweren Probleme. Für diese Probleme gibt es somit zwar exakte Lösungsverfahren, diese benötigen aber bei größer werdenden Probleminstanzen sehr lange Rechenzeiten. In der Praxis werden daher meist metaheuristische Ansätze verwendet, die zwar das Finden der optimalen Lösung nicht garantieren, dafür aber mit weniger Rechenzeit sehr gute Lösungen finden können. Eine solche

Metaheuristik ist die Tabu-Suche, auf der zahlreiche erfolgreiche Ansätze zur Lösung des VRP basieren.

Da das *Vehicle Routing Problem* von hoher Relevanz für diverse Wirtschaftszweige ist, zählt es zu den am meisten untersuchten Problemen der kombinatorischen Optimierung. Aufgrund der Vielfalt der Bedingungen und Einflussfaktoren, die in verschiedenen Anwendungsgebieten vorkommen können, wurden diverse Varianten des VRP definiert und zahlreiche Ansätze für die Lösung des VRP entwickelt. Die tatsächliche praktische Anwendung eines Lösungsverfahrens erfordert allerdings je nach Komplexität und Umfang der gegebenen Zusatzbedingungen eine individuelle Anpassung an die Begebenheiten im Unternehmen.

Auch bei dem nahe Leipzig ansässigen Logistikunternehmen fox-Courier liegt eine Variation des VRP mit verschiedenen Einschränkungen vor. Das Unternehmen beliefert am Tag mit zwei bis vier Fahrzeugen ca. 25-30 Kundinnen und Kunden im Großraum Leipzig. Die Pakete müssen in bestimmten Zeitfenstern an die Auftraggebenden ausgeliefert werden, zudem müssen die Kapazitätsbeschränkungen der Fahrzeuge und weitere Bedingungen beachtet werden. Aktuell werden die Auslieferungsrouten im Unternehmen manuell geplant, was mit einem erheblichen Zeitaufwand verbunden ist. Die Entwicklung einer Softwarelösung, die die Routenplanung des Unternehmens übernehmen und optimieren kann, ist Gegenstand dieser Arbeit.

1.2. Ziele der Arbeit

Für das Unternehmen fox-Courier soll eine Anwendung entwickelt werden, mit der ihre Auslieferungstouren geplant werden können. Die Anwendung muss in einer Benutzeroberfläche die Eingabe der tagesaktuellen Auftragsdaten und der zur Verfügung stehenden Fahrzeuge ermöglichen. Für diese Daten gilt es, Routen mit möglichst kurzen Lieferwegen und Fahrzeiten zu bestimmen, bei denen alle Anforderungen und Einschränkungen des Unternehmens berücksichtigt werden. Zu diesem Zweck soll ein geeigneter Optimierungsalgorithmus implementiert werden. Anschließend ist die Qualität der vom Algorithmus geplanten Routen zu evaluieren und zu beurteilen, inwiefern die entwickelte Routenplanungssoftware von Vorteil für das Unternehmen ist.

1.3. Aufbau der Arbeit

Zunächst wird ein Überblick über die zugrundeliegende Theorie gegeben, wobei auf kombinatorische Optimierungsprobleme im Allgemeinen, die Problemdefinition des vorliegenden VRP und die Tabu-Suche, auf der der implementierte Algorithmus basiert, eingegangen wird. Anschließend werden einige Ansätze aus der Literatur vorgestellt, welche zur Lösung ähnlicher Probleme entwickelt wurden. Im Hauptteil wird detailliert auf den Algorithmus

und die implementierten Erweiterungen eingegangen sowie die entwickelte Benutzeroberfläche gezeigt. Im Evaluierungsteil wird veranschaulicht, welche Auswirkungen die verwendeten Mechanismen auf die Qualität der Lösungen haben und wie gut der Algorithmus insgesamt verschiedene Probleminstanzen löst. Dafür werden sowohl Benchmark-Daten als auch reale Daten des Unternehmens verwendet. Zudem werden die Ergebnisse eingeordnet und diskutiert. Im letzten Teil werden die Inhalte zusammengefasst und mögliche Themen für zukünftige Arbeiten vorgeschlagen.

2. Grundlagen

2.1. Kombinatorische Optimierung

Ziel der kombinatorischen Optimierung ist das Finden eines optimalen Objekts in einer endlichen oder abzählbar unendlichen Menge von Objekten. Für das optimale Objekt ist eine vorher definierte Zielfunktion je nach Problemstellung minimal oder maximal. Die Objekte können ganze Zahlen, Teilmengen oder auch Graphen sein [1].

Kombinatorische Optimierungsprobleme finden sich in vielzähligen Bereichen der Wissenschaft und Industrie, wie z.B. bei der Planung von Kommunikationsnetzwerken, der Personalplanung, der Flugplanung, in der Logistik, der Bioinformatik, der Graphentheorie oder der algorithmischen Geometrie. Bekannte Probleme sind beispielsweise das quadratische Zuordnungsproblem, das *Travelling-Salesman-Problem*, das Rucksackproblem oder auch das Finden eines *Minimum Spanning Tree* in einem Graphen.

Viele kombinatorische Optimierungsprobleme gehören zur Klasse der NP-schweren Probleme. Diese Probleme lassen sich nicht in Polynomialzeit lösen, wenn vorausgesetzt wird, dass $P \neq NP$. Im schlimmsten Fall wird dann für das Bestimmen optimaler Lösungen exponentielle Rechenzeit benötigt [1].

Lösungsverfahren

Aufgrund der vielen praktischen Anwendungsgebiete wurden zahlreiche Algorithmen zur Lösung kombinatorischer Optimierungsprobleme entwickelt, wobei zwischen exakten und (meta-) heuristischen Lösungsverfahren unterschieden wird. Verbreitete exakte Lösungsverfahren sind z.B. der *Branch and Bound* oder der *Branch and Cut* Algorithmus. Diese Algorithmen garantieren, dass für ein Problem eine optimale Lösung gefunden wird, was bei NP-schweren Problemen allerdings mit sehr hohen Rechenzeiten einhergehen kann.

Bei Heuristiken und Metaheuristiken wird auf die Garantie, optimale Lösungen zu finden, verzichtet, um ausreichend gute Lösungen in deutlich kürzerer Zeit zu erhalten. Heuristiken nutzen spezielle Eigenschaften der Problemstruktur und sind daher meist auf nur ein spezifisches Problem anwendbar. Metaheuristiken sind komplexere *high-level* Strategien, die untergeordnete Heuristiken auf intelligente Weise steuern und verschiedene Konzepte zur Erkundung des Suchraums kombinieren. Sie sind universell auf verschiedene Probleme übertragbar und können auf effiziente Weise nahezu optimale Lösungen finden [1, 2].

Da in den meisten praktischen Anwendungsgebieten eine kurze Rechenzeit essenziell ist und dennoch qualitativ hochwertige Lösungen erwünscht sind, ist die Entwicklung von Metaheuristiken Gegenstand zahlreicher Forschungen. Viele verbreitete metaheuristische Algorithmen

sind von der Natur inspiriert, wie die *Ant Colony Optimization* und die Partikelschwarmoptimierung, welche das Verhalten von Ameisenkolonien bzw. das Schwarmverhalten von Tieren simulieren, oder auch genetische sowie evolutionäre Algorithmen. Weiterhin basieren mehrere erfolgreiche Algorithmen, wie die Tabu-Suche oder das *Simulated Annealing*, auf dem *Local Search* Verfahren. Bei der *Local Search* wird von einer Ausgangslösung ausgehend iterativ nach besseren Lösungen in einer geeignet definierten Nachbarschaft der jeweils aktuellen Lösung gesucht.

2.2. Vehicle Routing Problem

Das *Vehicle Routing Problem* (VRP) zählt zu den kombinatorischen Optimierungsproblemen und wurde erstmals 1959 von Dantzig et al. [3] unter dem Titel *„The Truck Dispatching Problem“* definiert. In ihrer Arbeit stellen Dantzig und Ramser die mathematische Modellierung und einen algorithmischen Ansatz zur Lösung des Problems der Lieferung von Benzin an Tankstellen vor. Aufgrund der zahlreichen Anwendungsbereiche des *Vehicle Routing Problems* in Industrie und Logistik wurde eine Vielzahl an Lösungsverfahren entworfen, wobei sich aufgrund der Zugehörigkeit zur Klasse der NP-schweren Probleme ein Großteil der Forschungen mit metaheuristischen Verfahren beschäftigen [4].

In der Definition des klassischen VRP muss eine Menge von Fahrzeugen, die sich anfangs in einem Depot befinden, diskrete Mengen von Gütern an eine Menge von auftraggebenden Personen liefern. Die Bestimmung der optimalen Routen stellt ein *Vehicle Routing Problem* dar. Es gilt eine Menge von Routen so zu entwerfen, dass jede Route im Depot beginnt und endet, jede Kundin und jeder Kunde genau einmal von genau einem Fahrzeug angefahren wird und die Gesamtkosten für die Routen minimiert werden.

Die realen Probleme von Logistikunternehmen sind oft komplexer als das klassische VRP, weshalb dieses in in der Praxis meist durch verschiedene Nebenbedingungen erweitert wird. Durch die Berücksichtigung der Fahrzeugkapazitäten ergibt sich beispielsweise das *Capacitated Vehicle Routing Problem* (CVRP) und ein Problem, bei dem Zeitfenster, in denen die Kundinnen und Kunden bedient werden müssen, miteinbezogen werden, bezeichnet man als *Vehicle Routing Problem with Time Windows* (VRPTW). Weitere Variationen sind zum Beispiel das *Vehicle Routing Problem with Backhauls* (VRPB), bei dem Pakete sowohl ausgeliefert als auch abgeholt und zum Depot zurückgebracht werden müssen oder das *Multi Depot Vehicle Routing Problem* (MDVRP), bei dem die Fahrzeuge bei mehreren Depots starten und enden. In den letzten Jahren wird auch zunehmend das *Electric Vehicle Routing Problem* (EVRP) untersucht, bei dem die begrenzte Reichweite von Elektrofahrzeugen und deren Wiederaufladung miteinkalkuliert werden müssen [5].

Spezifizierung des vorliegenden Problems

Bei dem Problem des Logistikunternehmens fox-Courier sind mehrere Nebenbedingungen von Bedeutung. Es sollen die unterschiedlichen Kapazitäten der Fahrzeuge, vorgeschriebene Zeitfenster, in denen die Auftraggebenden beliefert werden müssen und eine maximale Fahrzeit der Fahrerinnen und Fahrer berücksichtigt werden. Außerdem können einzelne Fahrzeuge mehrere Touren hintereinander erledigen und abweichend vom Depot auch andere Zieladressen haben. Zudem ist zusätzlich zur Distanz auch die Fahrtdauer zu minimieren und sowohl die Distanz als auch die Fahrtdauer zwischen zwei Adressen können sich je nach Fahrtrichtung unterscheiden. Man könnte das Problem als *Asymmetric Capacitated Heterogenous Fleet Vehicle Routing Problem with Time Windows* (ACHVRPTW) bezeichnen, zur Vereinfachung wird in dieser Arbeit jedoch die Bezeichnung VRP verwendet.

Das vorliegende VRP lässt sich als gewichteter gerichteter Graph $G = (V, A)$ darstellen, wobei $V = \{v_0, v_1, v_2, ...v_n\}$ die Menge der Knoten darstellt und $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ die Menge der Kanten. Der Knoten v_0 repräsentiert das Depot und die anderen Knoten die Kundinnen und Kunden. Jeder Kante (v_i, v_j) ist ein Wert d_{ij} zugeordnet, welcher der Distanz zwischen v_i und v_j entspricht sowie ein Wert t_{ij}, welcher der Fahrtdauer entspricht. Jedem Knoten wird ein Nachfragewert (hier das jeweilige Paketgewicht) q_i; $q_0 = 0$ und ein Zeitfenster $z_i = [a_i, b_i]$ zugeordnet. Die Fahrzeuge sollen die Auftragsadressen vor der oberen Grenze b_i des Fensters erreichen. Sollte eine Adresse vor der unteren Grenze a_i erreicht werden, muss bis zur Weiterfahrt der Zeitpunkt a_i abgewartet werden. δ ist die Zustellzeit, die der Entladezeit der Waren entspricht. $K = \{k_1, k_2, ...k_m\}$ stellt die Menge der verfügbaren Fahrzeuge dar und $Q = \{q_1, q_2, ...q_m\}$ die jeweiligen Kapazitäten. Die Zielfunktion des VRP besteht in der Minimierung der Gesamtkosten, welche sich aus der Entfernung und der Fahrtzeit zusammensetzt, unter Berücksichtigung aller Beschränkungen.

2.3. Tabu-Suche

Die Tabu-Suche ist ein verbreitetes metaheuristisches Verfahren zur Lösung kombinatorischer Optimierungsprobleme wie das *Vehicle Routing Problem*, welches sich leicht an verschiedene Zusatzbeschränkungen anpassen lässt. Der Tabu-Suchalgorithmus wurde von Glover [6] entwickelt und 1986 erstmals erwähnt. Der Algorithmus basiert auf dem *Local Search* Verfahren, wobei Lösungen mögliche Touren sind und die Nachbarschaft einer Lösung eine Sammlung von Touren ist, die sich durch kleine Änderungen der aktuellen Tour erzeugen lassen. In jeder Iteration wird die beste Tour in der Nachbarschaft zur aktuellen Lösung erklärt und der Prozess erneut gestartet. Die Suche endet, wenn eine maximale Iterationszahl oder eine zeitliche Obergrenze erreicht ist.

Um das Steckenbleiben in lokalen Optima zu vermeiden, wird eine Tabu-Liste genutzt. Diese enthält bestimmte Schritte, z.B. das Tauschen zweier Knoten in einer Tour, durch die in

einer vergangenen Iteration eine neue Lösung erzeugt wurde. Für eine gewisse Anzahl an Iterationen werden dann Lösungen ignoriert, die mit diesen Tabu-Schritten erreicht werden können.

Für eine erfolgreiche Anwendung der Tabu-Suche gilt es, ein Gleichgewicht zwischen der Intensivierung und der Diversifizierung der Suche zu finden. Bei der Intensivierung handelt es sich um eine detaillierte Erkundung eines Bereichs des Lösungsraums, meist in der Nähe einer guten Lösung. Die Diversifizierung besteht darin, die Suche auf vielversprechende Regionen des Lösungsraums zu lenken, die noch nicht erforscht wurden. In der Literatur finden sich zahlreiche Variationen und Modifikationen des grundlegenden Tabu-Suchalgorithmus mit verschiedenen Intensivierungs- und Diversifizierungsmechanismen. Dabei wird an unterschiedlichen Stellen im Algorithmus angesetzt, beispielsweise bei der Beschaffenheit der Tabu-Liste oder der Nachbarschaftserzeugung, um eine Verbesserung der Lösungsqualität zu erreichen.

Initiale Lösungsgenerierung

Bevor die Tabu-Suche beginnen kann, muss eine initiale Lösung, also eine Aufteilung der Aufträge in verschiedene Touren entsprechend der verfügbaren Fahrzeuge mit festgelegter Reihenfolge der Auslieferungen, erzeugt werden. Ein mögliches Verfahren dafür ist die *Randomized Insertion*, bei der die Aufträge zufällig auf die Fahrzeuge aufgeteilt und so Touren gebildet werden. Da die Qualität der initialen Lösung einen großen Einfluss auf den Verlauf der Tabu-Suche haben kann, finden sich in der Literatur mehrere Ansätze, bei denen anstelle einer zufälligen Aufteilung komplexere Methoden für die Generierung einer möglichst guten initialen Lösung verwendet werden. Beim *Nearest Neighbor* Algorithmus werden Touren gebildet, indem ausgehend vom Depot immer der Knoten mit der geringsten Distanz zum zuletzt eingefügten Knoten als nächster Knoten in die Tour eingefügt wird. Beim *Clarke and Wright* Algorithmus werden zunächst vom Depot aus Routen zu jedem Knoten erzeugt. Auf Basis der größten Kosteneinsparung werden dann Routen miteinander vereint, bis keine Einsparungen mehr möglich sind. Weitere Ansätze, wie der *Sweep* Algorithmus oder *Cluster-First Route-Second* Algorithmen teilen die Aufträge zunächst mithilfe geometrischer Methoden in Cluster auf und erzeugen darauf basierend die Touren [7, 8].

Tabu-Liste

Zentraler Bestandteil des Tabu-Suchalgorithmus ist die Tabu-Liste. Um zu verhindern, dass in kurzer Zeit immer wieder die gleiche Lösung gefunden wird, werden Schritte, die zur Erzeugung einer Lösung führen, in der Tabu-Liste gespeichert. Lösungen, die durch Schritte aus der Tabu-Liste erzeugt werden können, werden während der Nachbarschaftsgenerierung ignoriert. Allerdings wird in meisten Ansätzen der Tabu-Status eines Schrittes ignoriert und die erzeugte Lösung trotzdem verwendet, wenn diese besser als die bisher beste Lösung ist. Der Verbleib eines Schrittes in der Tabu-Liste kann durch eine zu Beginn der Suche

definierte Tabu-Iterationszahl begrenzt werden. Dieser Wert ist bei vielen Ansätzen statisch und wird während der Suche beibehalten, bei einigen Ansätzen wird er jedoch während der Suche verändert, meist in Abhängigkeit zu den bereits ausgeführten Iterationen oder anderen Parametern des Algorithmus. Eine weitere Möglichkeit ist es, in jeder Iteration einen zufälligen Wert für die Tabu-Iterationszahl zu verwenden. Anstatt die Iterationszahl zu begrenzen, für die ein bestimmter Schritt in der Tabu-Liste verbleibt, kann auch die Listengröße beschränkt werden. Wenn dann die maximale Größe erreicht wird, werden die Schritte nach dem *first in, first out* Prinzip wieder aus der Liste entfernt. Die Listengröße kann statisch sein oder dynamisch im Suchverlauf verändert werden [7].

Nachbarschaftserzeugung

Um während der Tabu-Suche von einer Lösung zur nächsten zu gelangen, wird in jeder Iteration eine Nachbarschaft der aktuellen Lösung erzeugt. Dafür werden die Positionen einzelner Knoten innerhalb der Touren der aktuellen Lösung durch bestimmte Schritte verändert. Diese Schritte können zwei, drei oder mehr Knoten betreffen, welche zufällig oder nach bestimmten Kriterien ausgewählt werden. Die Art der Schritte, welche für die Nachbarschaftserzeugung verwendet werden, kann einen großen Einfluss auf die Lösungsqualität haben. In der Literatur finden sich verschiedene Arten von Schritten und bei einigen Ansätzen werden mehrere Schritte zur Erzeugung unterschiedlicher Nachbarschaften verwendet. Häufig verwendete Schritte sind unter anderem *Vertex Insertion* und *Vertex Exchange*, bei denen ein Knoten an einer Stelle aus einer Tour entfernt und an anderer Stelle eingefügt wird bzw. zwei Knoten vertauscht werden. Dies kann innerhalb einer Tour oder zwischen zwei Touren geschehen. Weiterhin werden *2-opt* bzw. *k-opt* Schritte vielfach eingesetzt. Dabei werden zwei bzw. k Kanten zwischen Knoten entfernt und die betroffenen Knoten in allen möglichen Kombinationen wieder miteinander verknüpft. Auch *Generalized Insertion* (GENI) wird in vielen Arbeiten genutzt. Bei diesem Schritt wird ein Knoten aus einer Tour entfernt und zwischen zwei nahe liegende Knoten eingefügt. Diese Knoten müssen nicht zwingend benachbart sein, wodurch sich auch ihre Positionen verändern können und an mehreren Stellen neue Kanten entstehen [7, 9].

3. Verwandte Arbeiten

In der Literatur werden zahlreiche metaheuristische Methoden zur Lösung verschiedener Variationen des *Vehicle Routing Problems* beschrieben. Ein verbreitetes Verfahren für Optimierungsprobleme ist die Partikelschwarmoptimierung (PSO), welche mehrfach für die Lösung von *Vehicle Routing Problems* eingesetzt wird. Das populationsbasierte Verfahren ist inspiriert vom Schwarmverhalten von Tieren. Mögliche Lösungen werden als Partikel dargestellt, welche sich innerhalb eines Schwarms in Richtung der eigenen bisher besten Position und der besten Position innerhalb des Schwarms bewegen. Ai et al. [10] verwenden die PSO für ein kapazitives VRP und vergleichen zwei Dekodierungsmethoden für die Lösungsdarstellungen. Marinakis et al. [11] schlagen einen PSO basierten Ansatz zur Lösung des VRP mit Zeitfenstern vor, wobei sie die PSO um verschiedene adaptive Suchmechanismen erweitern. Belmecheri et al. [12] entwickelten einen PSO basierten Ansatz verbunden mit lokaler Suche für die Lösung eines VRP mit Zeitfenstern und anderen Beschränkungen.

Ein hybrider Ansatz von Chen et al. [13] verbindet die Partikelschwarmoptimierung mit dem *Simulated Annealing* für die Lösung des kapazitiven VRP. *Simulated Annealing* (SA) ist ein metaheuristisches Verfahren, welches auf der lokalen Suche basiert. Durch zufällige Operationen werden Nachbarschaften erzeugt und bis zu einem gewissen Grad Verschlechterungen akzeptiert, um lokale Optima verlassen zu können, wobei sich die Akzeptanz von Verschlechterungen im Laufe der Suche verringert. Auch Lin et al. [14] und Wei et al. [15] schlagen SA basierte Ansätze für die Lösung verschiedener Versionen des kapazitiven VRP vor. Baños et al. [16] beschäftigen sich mit der Lösung des VRP mit Zeitfenstern und anderen Beschränkungen und vergleichen dafür verschiedene SA Variationen.

Weiterhin werden für *Vehicle Routing Problems* mehrfach genetische Algorithmen eingesetzt. Genetische Algorithmen (GA) lösen Optimierungsprobleme, indem sie den Prozess der natürlichen Selektion nachahmen. Das Suchverfahren beginnt mit einer zufällig erzeugten Population von Lösungen und erzeugt durch Rekombinationsmethoden neue Lösungen. Für mehr Diversität werden zusätzlich Mutationsmechanismen eingesetzt. Berger et al. [17] schlagen einen hybriden GA für die Lösung des kapazitiven VRP vor, wobei sie genetische Operatoren mit weiteren Suchstrategien verknüpfen. Nazif et al. [18] schlagen einen GA mit optimiertem Crossover Operator für die Lösung des kapazitiven VRP vor. Marinakis et al. [19] verknüpfen für ihren Ansatz den GA mit der Partikelschwarmoptimierung für eine effektivere Erkundung des Suchraums und wenden ihren Ansatz am klassischen VRP an. Liu et al. [20] kombinieren den GA mit dem *Large Neighborhood* Algorithmus für die Lösung eines kumulativen kapazitiven VRP mit Zeitfenstern.

Ein weiterer verbreiteter Ansatz ist der *Ant Colony Optimization* Algorithmus (ACO). Dieser Optimierungsalgorithmus ist durch das Verhalten von Ameisenkolonien inspiriert und simuliert künstliche Ameisen, die bei der Erkundung verschiedener Lösungen Pheromone

abgeben, wodurch in folgenden Iterationen mehr Ameisen bessere Lösungen finden. Rizzoli et al. [21] zeigen die erfolgreiche praktische Anwendung der ACO für verschiedene reale *Vehicle Routing Problem* Variationen. Yu et al. [22] präsentieren einen ACO Ansatz mit einer neuen Pheromonaktualisierungsstrategie und einer Mutationsoperation für die Lösung des VRP. Ding et al. [23] schlagen einen hybriden ACO für das VRP mit Zeitfenstern vor. Den ACO ergänzen sie um mehrere Anpassungen, um dem Steckenbleiben in lokalen Optima entgegenzuwirken und die Konvergenzgeschwindigkeit zu verbessern. Lee et al. [24] kombinieren die ACO mit dem *Simulated Annealing* Verfahren für das kapazitive VRP.

In aktuellen Ansätzen werden immer häufiger auch neuronale Netze zur Lösung von Optimierungsproblemen wie das VRP eingesetzt. Nazari et al. [25] kombinieren ein *Recurrent Neural Network* mit einem Attention-Mechanismus für die Lösung des VRP. Kool et al. [26] verwenden ein Graph-Attention-Netzwerk welches mittels *Reinforcement Learning* trainiert wird, um Lösungen für verschiedene Routingprobleme zu generieren, darunter das *Travelling Salesman Problem* und das kapazitive VRP. Hottung et al. [27] haben einen Ansatz entwickelt, den sie *Neural Large Neighborhood Search* nennen. Dieser basiert auf der *Large Neighborhood Search* und nutzt ein *Deep Neural Network* mit Attention-Mechanismus.

Einer der erfolgreichsten Ansätze für das *Vehicle Routing Problem* ist die Tabu-Suche, auf welcher auch der Algorithmus in dieser Arbeit basiert. In der Literatur finden sich zahlreiche Implementierungen und Variationen der Tabu-Suche für verschiedene Arten von *Vehicle Routing Problems*.

Eine populäre Variation ist die Erweiterung der Tabu-Suche um einen *Adaptive Memory* Mechanismus, welcher von Rochat et al. [28] entwickelt wurde. Es wird dabei eine Menge von guten Lösungen von der Tabu-Suche erzeugt und diese fortlaufend aktualisiert, wenn neue gute Lösungen gefunden werden. Durch Rekombinationen von Lösungen aus dieser Menge werden neue Lösungen erzeugt. Dafür werden qualitativ hochwertige Teiltouren aus den Lösungen extrahiert und kombiniert welche zusammen mit den restlichen ungerouteten Punkten initiale Lösungen bilden, mit denen erneut eine Tabu-Suche gestartet wird. Neben anderen schlagen Tarantilis [29], Euchi et al. [30] und Li et al. [31] Tabu-Suchalgorithmen mit *Adaptive Memory* für verschiedene Variationen des VRP vor.

Battiti et al. [32] beschreiben erstmals eine reaktive Tabu-Suche, welche sie zur Lösung des Rucksackproblems und des quadratischen Zuordnungsproblems verwenden. Sie ergänzen dabei die Tabu-Suche um einen Mechanismus, der Suchparameter an den Verlauf der Suche anpasst. Dabei wird die Tabu-Liste vergrößert, wenn sich Lösungen wiederholen, um weitere Wiederholungen zu vermeiden und dann nach einer gewissen Anzahl an Iterationen wieder langsam verkleinert. Unter anderem beschreiben Chiang et al. [33] und Bräysy [34] auf der Tabu-Suche basierende Ansätze mit reaktiven Mechanismen für das VRP mit Zeitfenstern.

Eine andere Variation ist die granulare Tabu-Suche, welche von Toth et al. [35] entwickelt und für das *Vehicle Routing Problem* angepasst wurde. Bei der granularen Tabu-Suche werden die Nachbarschaften begrenzt, um die Rechenzeit zu reduzieren, ohne jedoch die Qualität

der gefundenen Lösungen wesentlich zu beeinträchtigen. Dafür wird in jeder Iteration nur ein Teilgraph des originalen Graphen mit besonders vielversprechenden Pfaden für die Nachbarschaftsgenerierung berücksichtigt. Bernal et al. [36] nutzen eine abgewandelte granulare Tabu-Suche für ein reales VRP mit heterogenem Fuhrpark und Zeitfenstern. Weiterhin schlagen Jin et al. [37] und Escobar et al. [38] auf granularer Tabu-Suche basierende Ansätze für verschiedene VRP vor.

Ein weiterer vielfach untersuchter Ansatz ist die parallele Tabu-Suche, wobei verschiedene Parallelisierungsstrategien entwickelt wurden, die der Reduzierung der Rechenzeit oder auch der breiteren Suche im Lösungsraum dienen können. Cordeau et al. [39] zeigen die erfolgreiche Nutzung einer parallelen Tabu-Suche für mehrere VRP Variationen. Badeau et al. [40] schlagen eine parallele Tabu-Suche für das VRP mit Zeitfenstern vor und Jin et al. [37] für das kapazitive VRP.

In mehreren Arbeiten werden zusätzlich zu anderen Mechanismen multiple Nachbarschaften (auch variable Nachbarschaften genannt) genutzt. Anstatt einen Schritt zur Nachbarschaftserzeugung zu verwenden, wie bei der klassischen Tabu-Suche, werden verschiedene Arten von Schritten genutzt. Von diesen wird in jeder Iteration zufällig oder strategisch ein Schritt ausgewählt oder auch mehrere parallel verwendet. Beispielsweise werden in den bereits erwähnten Arbeiten von Toth et al. [35] und Jin et al. [37] multiple Nachbarschaften verwendet oder auch bei Kytöjoki et al. [41] und Xia et al. [42].

Außerdem wird mehrfach die Verwendung eines Neustart-Mechanismus erwähnt, auch in den bereits genannten Arbeiten von Rochat et al. [28] und Bernal et al. [36]. Dieser Mechanismus dient der Intensivierung der Suche, indem nach einer bestimmten Anzahl an Iterationen der Suchprozess mit der bisher besten gefunden Lösung erneut gestartet wird. Auch Brandão [43] schlägt einen Tabu-Suchalgorithmus mit Neustart-Mechanismus für eine Variation des VRP vor.

4. Implementierung

4.1. Vorverarbeitung der Daten

Liefer Straße	Liefer HausNr	Liefer PLZ	Liefer Ort
Musterstraße	3	04109	Leipzig
Modellplatz	10	04105	Leipzig
Beispielweg	25	04177	Leipzig

Gewicht	Liefer-Von	Liefer-Bis	...
15,2	08:00	10:00	...
3,1		12:00	...
7,0	09:00	11:00	...

Tabelle 4.1.: Aufbau der Auftragsdaten

Die vom Unternehmen bereitgestellten Auftragsdaten für die Routenplanung sind in Tabelle 4.1 beispielhaft dargestellt. Ein Datensatz enthält jeweils Angaben für die Aufträge an einem Tag. Relevant sind davon die Lieferadressen, Paketgewichte sowie die Zeitfenster, in denen die Pakete bei den Auftraggebenden geliefert werden sollen.

Die Spalte *Gewicht* bezieht das Volumengewicht, eine Größe aus dem Logistikbereich, mit ein. Dieses Maß setzt sich aus dem Volumen eines Pakets und einem Divisor zusammen. Das Volumengewicht wird mit dem tatsächlichen Gewicht des Pakets verglichen und der größere der beiden Werte findet sich in den Daten. So lassen sich auch sperrige, aber vergleichsweise leichte Pakete ins Verhältnis setzen, womit die Spalte *Gewicht* für das Kapazitätskriterium des Problems genutzt werden kann.

Für die weitere Verarbeitung werden die Spalten *Liefer Straße*, *Liefer HausNr*, *Liefer PLZ* und *Liefer Ort* zu einer Spalte zusammengefügt und die Werte in *Liefer-Von* und *Liefer-Bis* in Minuten umgerechnet.

Bestimmen der Koordinaten

Um die zugehörigen Koordinaten zu den Adressen zu bestimmen, wird der *Nominatim geocoder*[1] genutzt, welcher auf *OpenStreetMap*[2] Daten basiert. Da sich mit diesem Geocoder manche Adressen nicht finden lassen (z.B. bei Rechtschreibfehlern) wird für die nicht gefundenen Adressen der *Google Maps geocoder*[3] verwendet, welcher zuverlässiger bei fehlerhaften

[1] `https://geopy.readthedocs.io/en/stable/#nominatim` (abgerufen am 15.12.2021)
[2] `https://www.openstreetmap.de/` (abgerufen am 15.12.2021)
[3] `https://developers.google.com/maps/documentation/geocoding/overview` (abgerufen am 15.12.2021)

Adressen funktioniert. Da dieser aber ab einer bestimmten Anzahl an Aufrufen kostenpflichtig ist, soll so die Anzahl der Anfragen möglichst gering gehalten werden. Zusätzlich werden die bereits bestimmten Koordinaten-Adress-Paare gespeichert und bei zukünftigen Programmaufrufen genutzt. Da einige Kundinnen und Kunden regelmäßig vom Unternehmen beliefert werden, können dadurch API-Anfragen eingespart werden.

Ermitteln der Entfernungen und Fahrzeiten

Mit Hilfe der *openrouteservice*[4] API können zu einer Liste von Koordinaten eine Distanz- und eine Fahrtdauer-Matrix bestimmt werden. Die jeweiligen Werte werden basierend auf *OpenStreetMap* Daten entsprechend dem Transportmittel Auto berechnet. So werden zu jeder zu beliefernden Adresse, basierend auf den Koordinaten, die Distanzen und die Fahrtdauer zu jeder anderen Adresse ermittelt.

Adresse	Gewicht	Liefer-Von	Liefer-Bis
Musterstraße 3, 04109 Leipzig	15,2	480	600
Modellplatz 10, 04105 Leipzig	3,1		720
Beispielweg 25, 04177 Leipzig	7,0	540	660

Breitengrad	Längengrad	Distanzen	Fahrtzeiten
51.344198	12.396224	[0, 5948, 6718]	[0, 17, 20]
51.309969	12.373963	[5431, 0, 6587]	[16, 0, 19]
51.331827	12.318616	[6539, 7124, 0]	[19, 18, 0]

Tabelle 4.2.: Daten nach der Vorverarbeitung

Tabelle 4.2 zeigt den veränderten Beispieldatensatz nach der Vorverarbeitung. Die zur Adresse gehörenden Spalten wurden zu einer Spalte zusammengefasst und die Werte in *Liefer-Von* und *Liefer-Bis* in Minuten umgerechnet. Die Spalten *Breitengrad* und *Längengrad*, welche die ermittelten Koordinaten zu den Adressen enthalten, wurden hinzugefügt sowie die berechneten Distanzen und Fahrtzeiten ergänzt.

4.2. Tabu-Suche

4.2.1. Initiale Lösungsgenerierung

Bevor die eigentliche Tabu-Suche beginnen kann, muss eine initiale Lösung als Ausgangspunkt der Suche erzeugt werden. Dafür werden zunächst 1000 Lösungen erstellt, indem mehrmals zufällige Permutationen aller Adressen erzeugt werden. Diese werden jeweils sukzessive auf

[4] `https://openrouteservice.org/dev/#/api-docs/directions` (abgerufen am 15.12.2021)

die verfügbaren Fahrzeuge aufgeteilt, bis jede Adresse einem Fahrzeug zugeordnet wurde. Für die den Fahrzeugen zugeteilten Adressen wird jeweils eine *Nearest Neighbor* Suche durchgeführt. Dafür wird vom Depot ausgehend zu jedem Knoten der am nächsten liegende Knoten als nächster Knoten in der Tour gewählt. Dies nimmt nur wenig Rechenzeit in Anspruch, erzeugt jedoch eine erste Vorsortierung der Adressen. Von den auf diese Weise generierten Lösungen wird die mit den geringsten Kosten als initiale Lösung für den Start der Tabu-Suche ausgewählt.

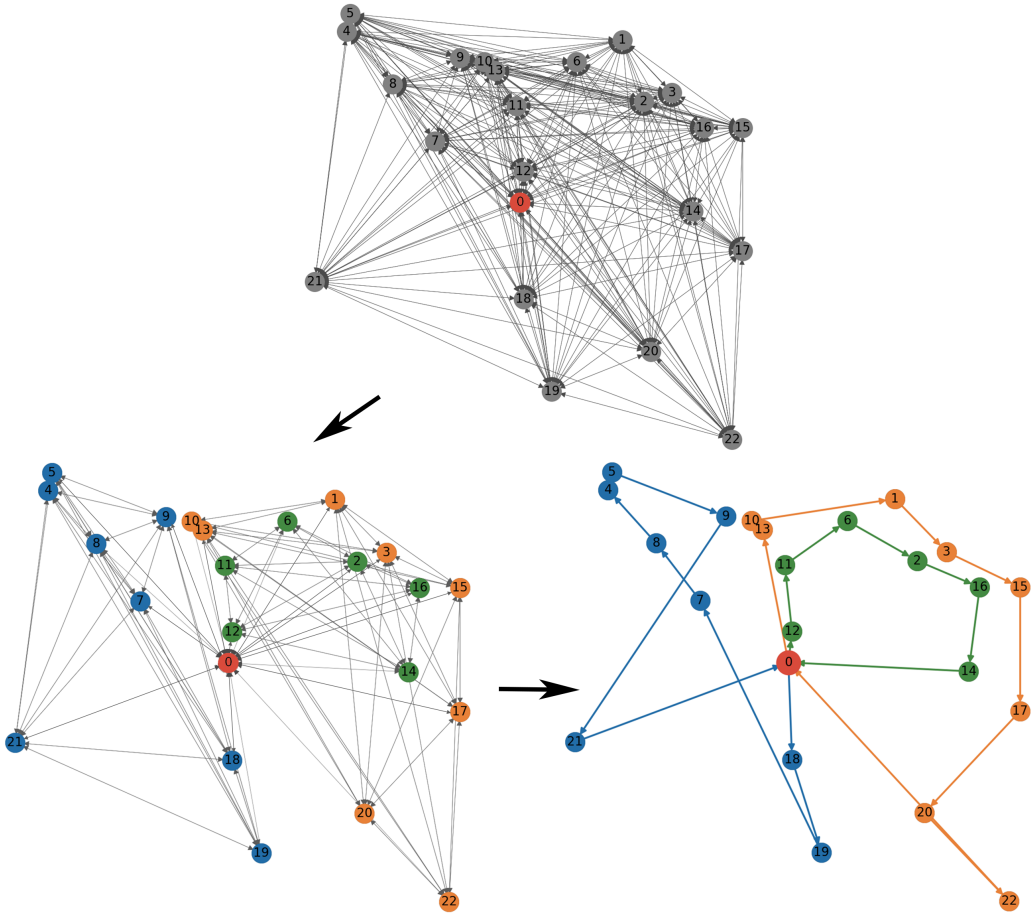


Abbildung 4.1.: Erzeugung einer initialen Lösung

In Abbildung 4.1 sind die beiden Schritte zur Erzeugung der initialen Lösung an einem Beispielproblem dargestellt. Ausgehend vom Ausgangsgraphen werden im ersten Schritt die Aufträge zufällig auf die in diesem Fall drei verfügbaren Fahrzeuge aufgeteilt und die Reihenfolgen der Aufträge der einzelnen Touren im zweiten Schritt mittels der *Nearest Neighbor* Suche bestimmt. Die hier dargestellte initiale Lösung ist die mit den geringsten Kosten der 1000 für das Problem erzeugten initialen Lösungen. Sie bildet den Ausgangspunkt für die Tabu-Suche.

4.2.2. Kostenfunktion

Um die Qualität einer Lösung zu bestimmen und sie mit anderen Lösungen zu vergleichen, wird eine Kostenfunktion aufgestellt, welche alle relevanten Werte miteinbezieht. Sie setzt sich wie folgt zusammen:

$$cost(s) = k_1 \cdot a + k_2 \cdot b + k_3 \cdot c + k_4 \cdot d + k_5 \cdot e \tag{4.1}$$

mit

a: Summe der Überlast aller Fahrzeuge,

b: Summe der Fahrzeitüberschreitungen,

c: Summe der Verspätungen,

d: Gesamtdistanz,

e: Gesamtdauer,

wobei $k_1 > k_2 > k_3 > k_4 \approx k_5 = 1$

Während der Tabu-Suche soll diese Kostenfunktion minimiert, also eine Lösung s gefunden werden, deren Kosten möglichst gering sind. Dabei bezieht die Kostenfunktion die Kapazitätsbeschränkung mit ein, indem die Summe der Überlast der Fahrzeuge, also die Überschreitung der Fahrzeugkapazität durch die Paketgewichte, mit einem besonders hohen Faktor k_1 multipliziert wird. So sucht der Algorithmus primär nach neuen Lösungen, bei denen die Überlast geringer bzw. möglichst null ist.

Am zweithöchsten gewichtet wird die Summe der Fahrzeitüberschreitungen, die auftritt, wenn die Fahrtzeit einer Tour länger als die vorgegebene maximale Fahrzeit ist. Dieser Wert sollte möglichst null sein, um die Arbeitszeitbestimmungen der Fahrerinnen und Fahrer nicht zu verletzen. Bei der Berechnung der Fahrtzeiten wird pro Auftrag eine Zustellzeit von 5 Minuten mit eingerechnet.

Die Priorität von Verspätungen steht an dritter Stelle. Hierfür wird nach Lösungen gesucht, bei denen die ermittelten Ankunftszeiten bei allen Adressen den oberen Wert der zugehörigen Zeitfenster möglichst nicht überschreiten. Liegt die Ankunftszeit eines Fahrzeugs bei einer Adresse unter dem unteren Wert des Zeitfensters, so wird dieser Wert abgewartet, bis die Tour fortgesetzt werden kann.

Damit der Algorithmus zunächst Lösungen findet, die die genannten Beschränkungen nicht verletzen, werden die Distanz und Fahrtdauer mit deutlich geringeren Faktoren gewichtet. Diese beiden Werte korrelieren bei vielen Tourabschnitten, jedoch muss die Fahrtdauer sich nicht zwingend bei einer größeren Entfernung erhöhen. Sollte ein Tourabschnitt einer neuen Tour beispielsweise über eine Autobahn führen, kann das eine höhere Distanz, aber eine niedrigere Fahrtdauer zur Auswirkung haben. Allerdings ist für das Unternehmen die Minimierung beider Werte von Vorteil, weshalb diese ähnlich hoch gewichtet werden.

4.2.3. Suchalgorithmus

Für die Lösung des vorliegenden *Vehicle Routing Problems* wurde ein reaktiver Tabu-Suchalgorithmus mit multiplen Nachbarschaften und einem Neustart-Mechanismus implementiert. Er ist in Algorithmus 1 vereinfacht als Pseudocode dargestellt.

Algorithmus 1 Tabu-Suche

```
1:  max_iterations ← 500 + 20 · num_addresses
2:  best_solution ← current_solution ← initial_solution
3:  tenure_max ← 18
4:  tenure_min ← tabu_tenure ← 6
5:  tabu_list ← visited_tours ← [ ]
6:  counter ← 0
7:  while counter <= max_iterations do
8:      neighbors ← generate_neighbors(current_solution, visited_tours)
9:      current_solution ← best(neighbors)
10:     tabu_list ← update_tabu(tabu_list, current_solution, tabu_tenure)
11:     visited_tours ← update_visited_tours(current_solution)
12:     if current_solution.cost >= best_solution.cost then
13:         counter ← counter + 1
14:         if tabu_tenure < tenure_max & counter % 20 == 0 then
15:             tabu_tenure ← tabu_tenure + 1
16:         end if
17:     else
18:         best_solution ← improve_individual_tours(current_solution)
19:         counter ← 0
20:         tabu_tenure ← tenure_min
21:     end if
22:     if counter % round(0.2 · max_iterations) == 0 then
23:         current_solution ← best_solution
24:         tabu_tenure ← tenure_min
25:         visited_tours ← [ ]
26:     end if
27: end while
```

Zunächst werden die benötigten Variablen und Konstanten definiert. Die Zahl der maximalen Iterationen ist hierbei abhängig von der Anzahl der zu beliefernden Adressen (Zeile 1). Solange die maximale Iterationszahl nicht erreicht ist, wird folgender Ablauf wiederholt: Es werden zunächst drei zufällige Arten von Schritten ausgewählt und damit jeweils eine Nachbarschaft der aktuellen Lösung bis zu einer festgelegten Größe generiert (Zeile 8). In verschiedenen Stadien der Suche kann mit unterschiedlichen Arten von Schritten eine Verbesserung erzielt werden, daher kann so eher ein aktuell passender Schritt gefunden werden als mit nur einem einzelnen zufälligen Schritt. Wird währenddessen eine bessere Lösung als die aktuelle Lösung gefunden, wird der Nachbarschaftsgenerierungsprozess beendet und diese bessere Lösung wird zur neuen aktuellen Lösung. Ansonsten wird die beste aller erzeugten Nachbarschaftslösungen als neue aktuelle Lösung verwendet.

Bei der Nachbarschaftserzeugung werden nur Lösungen berücksichtigt, bei deren Bildung keine als tabu gelisteten Knotenpaare beteiligt sind. Eine Ausnahmeregelung davon tritt ein, wenn eine durch einen Tabu-Schritt erzeugte Lösung besser als die bisher beste Lösung ist. In dem Fall wird diese Lösung als aktuelle Lösung gesetzt. So wird gewährleistet, dass keine neuen besten Touren übersehen werden.

Nachdem eine neue aktuelle Lösung gefunden wurde, wird die Tabu-Liste aktualisiert (Zeile 10). Dafür wird das Knotenpaar, welches an der Erzeugung der neuen Lösung beteiligt war, zusammen mit dem aktuellen Tabu-Iterationswert in die Tabu-Liste eingetragen. Die Iterationswerte der anderen Einträge in der Tabu-Liste werden dekrementiert und anschließend die Knotenpaare, deren Tabu-Wert 0 beträgt, aus der Liste entfernt.

Daraufhin werden die Kosten der neuen Lösung mit denen der bisher besten Lösung verglichen (Zeile 12). Nur wenn die neue Lösung nicht besser ist, wird der Iterationszähler erhöht. Somit beschränkt die maximale Iterationszahl allein die Iterationen ohne Verbesserungen. Daher wird die Rechenzeit für einen Durchlauf der Tabu-Suche zusätzlich auf maximal zehn Sekunden beschränkt, um so die Rechenzeit auf eine annehmbare Dauer zu begrenzen. Wenn eine neue beste Lösung gefunden wurde, so wird der Zähler wieder auf 0 gesetzt. Eine neu gefundene beste Lösung wird zusätzlich optimiert, indem die Einzeltouren verbessert werden (Zeile 18). Dafür wird für jede einzelne Tour der Lösung eine verkürzte Tabu-Suche mit wenigen Iterationen durchgeführt.

Reaktiver Mechanismus

Zeile 15 und 20 gehören zum reaktiven Mechanismus, der in gewissem Maße auf den aktuellen Stand der Suche reagiert. Zum einen wird der Tabu-Iterationswert entsprechend angepasst. Wenn keine bessere Lösung gefunden wurde, wird der Wert langsam (alle 20 Iterationen) erhöht, um eine diversere Suche nach weiter entfernten Nachbarn zu ermöglichen. Sollte eine neue beste Lösung gefunden werden, wird der Tabu-Iterationswert wieder auf den minimalen Wert zurückgesetzt, was zu einer Intensivierung der Suche führt, da so das Durchlaufen kleinerer Perioden ermöglicht wird. Zum anderen wird darauf reagiert, wie oft einzelne Lösungen besucht werden. Dafür werden die besuchten Lösungen zusammen mit der Anzahl, wie oft sie bisher vorkamen, gespeichert und diese Angabe in jeder Iteration aktualisiert (Zeile 11). Wenn die aktuelle Lösung bisher bereits mindestens fünf Mal besucht wurde, werden für die Nachbarschaftserzeugung gezielt Schritte verwendet, die größere Veränderungen der Lösung vornehmen. So wird die Suche in entferntere Nachbarschaften geführt, um aus einem möglichen lokalen Optimum zu entkommen.

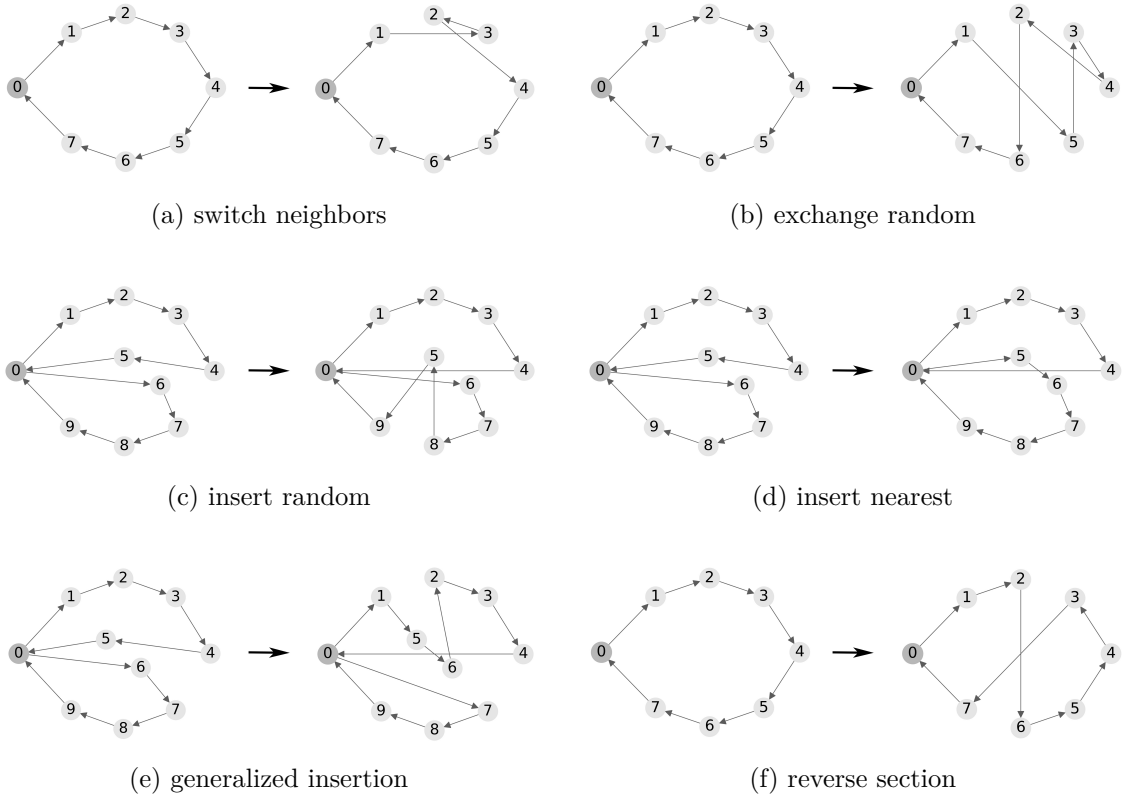
Neustart-Mechanismus

Immer nach 20% der maximalen Iterationen wird ein Neustart durchgeführt, indem die aktuelle Lösung verworfen und die Suche mit der bisher besten Lösung weitergeführt wird

(Zeile 23). Zudem wird der Tabu-Iterationswert auf den Startwert zurückgesetzt und die Liste der zuvor besuchten Lösungen geleert. So kann ein wenig aussichtsreicher Teil des Suchraums verlassen und in der Nähe der besten Lösung weitergesucht werden. Sollte drei Mal hintereinander beim Zeitpunkt des Neustarts die gleiche Lösung die aktuell beste sein, wird die Suche beendet.

Multiple Nachbarschaften

Für eine möglichst diverse Suche werden verschiedene Arten von Schritten genutzt und in jeder Iteration zufällige Schritte für die Nachbarschaftsgenerierung ausgewählt.



(a) switch neighbors

(b) exchange random

(c) insert random

(d) insert nearest

(e) generalized insertion

(f) reverse section

Abbildung 4.2.: Schritte zur Nachbarschaftserzeugung

Beim Schritt *switch neighbors* werden die Positionen zweier benachbarter Knoten in einer Tour miteinander vertauscht. Dies wird für jedes benachbarte Knotenpaar wiederholt und so die Nachbarschaft erzeugt. In Abbildung 4.2a wird beispielhaft das Vertauschen der Nachbarknoten 2 und 3 dargestellt.

Der Schritt *exchange random* generiert eine Nachbarschaft, indem jeweils zwei zufällige Knoten ausgewählt und miteinander vertauscht werden. Abbildung 4.2b zeigt die veränderte Tour, nachdem die Positionen der Knoten 2 und 5 ausgetauscht wurden.

Bei *insert random* wird ein zufälliger Knoten ausgewählt, welcher neben einem zweiten zufälligen Knoten eingefügt wird. In Abbildung 4.2c ist veranschaulichend eine Einfügeoperation dargestellt, bei der Knoten 5 hinter Knoten 8 eingesetzt wird.

Beim Schritt *insert nearest* wird ein zufälliger Knoten ausgewählt und n Knoten bestimmt, welche die geringste Entfernung von diesem Knoten aus oder zu diesem Knoten hin haben. Der ausgewählte Knoten wird entsprechend vor oder hinter die nahe liegenden Knoten eingefügt. Beispielhaft dargestellt wird dies in Abbildung 4.2d. Hier wurde Knoten 5 ausgewählt und vor Knoten 6 eingesetzt, da die Distanz zu diesem Knoten am geringsten ist.

Bei der *generalized insertion* werden zu einem zufälligen Knoten die n Knoten mit geringster Entfernung zu und n Knoten mit geringster Entfernung von diesem Knoten aus ausgewählt. Der zufällige Knoten wird hinter einen Knoten des ersten Sets eingefügt und ein Knoten des zweiten Sets wird zusätzlich dahinter eingefügt. So liegt der zufällig ausgewählte Knoten nach den Einfügeoperationen zwischen zwei Knoten mit geringer Entfernung. Abbildung 4.2e zeigt die veränderten Touren, nachdem Knoten 5 hinter Knoten 1 und Knoten 6 anschließend hinter Knoten 5 eingefügt wurde. Knoten 5 ist hierbei der zufällig ausgewählte Knoten und Knoten 1 der mit der geringsten Distanz hin zu Knoten 5 und Knoten 6 der mit der geringsten Distanz von Knoten 5 ausgehend.

Der Schritt *reverse section* erzeugt eine Nachbarschaft, indem zwei zufällige Knoten einer Tour ausgewählt werden und die Reihenfolge der Knoten im Abschnitt zwischen diesen Knoten umgekehrt wird. In Abbildung 4.2f wurden die Knoten 2 und 7 ausgewählt und die Reihenfolge des Tourabschnitts zwischen ihnen umgekehrt.

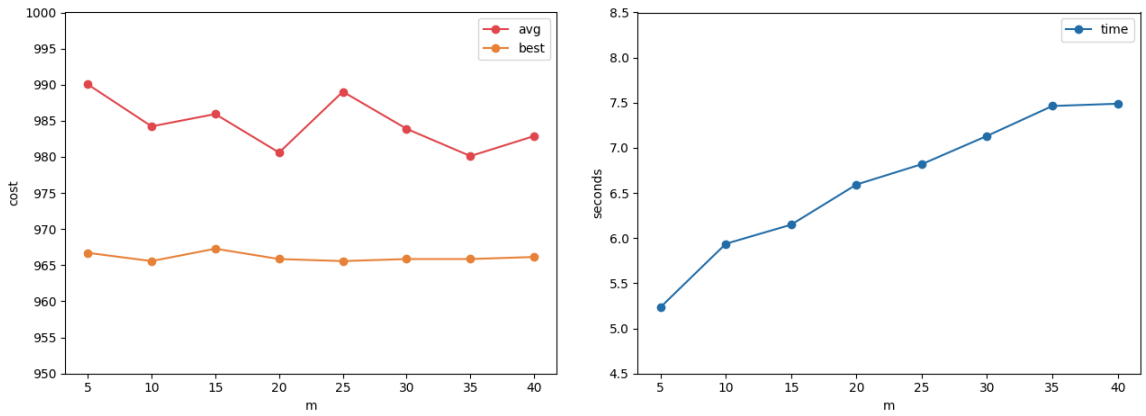
Die Operationen der Schritte *switch neighbors* und *reverse section* betreffen ausschließlich Knoten innerhalb einer Tour, bei den anderen Schritten können die Knoten auch aus verschiedenen Touren stammen. Bei zusätzlich verwendeten Abwandlungen der Schritte *exchange random*, *insert random* und *insert nearest* werden hinzukommend zu den Knoten, die ausgetauscht oder eingefügt werden, die Nachbarn der Knoten mit getauscht oder eingesetzt. Die Operationen eines Schrittes werden so lange mit neuen zufälligen Knoten durchgeführt, bis die Größe der erzeugten Nachbarschaft einen maximalen Wert erreicht.

4.2.4. Parameterkonfiguration

Einige Parameter des Tabu-Suchalgorithmus werden vor Beginn der Tabu-Suche festgelegt und haben Einfluss auf die Lösungsqualität sowie die Dauer eines Suchdurchlaufs. Um die Werte zu bestimmen, die ein gutes Gleichgewicht zwischen Qualität und Rechenzeit bilden, werden mehrere Suchen für Auftragsdaten des Unternehmens fox-Courier mehrerer Tage mit verschiedenen Werten für die Parameter durchgeführt. Es wird jeweils der Durchschnitt der jeweils besten und durchschnittlichen Kosten von je zehn Durchläufen pro Instanz ermittelt sowie die durchschnittliche benötigte Rechenzeit.

Maximale Iterationszahl

Damit sich die Tabu-Suche an unterschiedliche Problemgrößen anpasst, wird die maximale Iterationszahl u abhängig von der Anzahl der Aufträge n festgelegt: $u = 500 + m \cdot n$

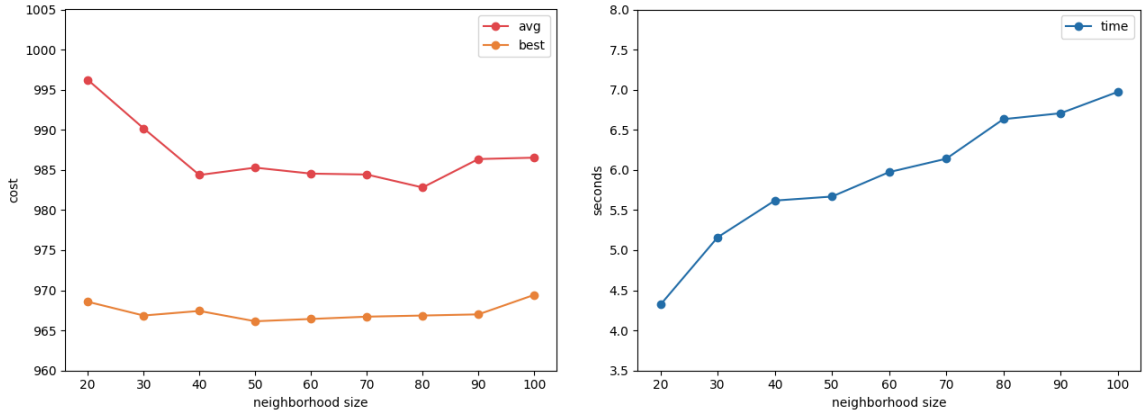


(a) Kosten (b) Rechenzeit

Abbildung 4.3.: Durchschnittliche Kosten, durchschnittlich beste Kosten und durchschnittliche Rechenzeit in Abhängigkeit von m

Abbildung 4.3 zeigt, dass die durchschnittliche Dauer der Durchläufe mit größer werdendem m ansteigt. Die durchschnittlichen Kosten sind bei $m = 20$ und $m = 35$ am geringsten, die Kosten für die besten Lösungen bleiben ab $m = 20$ gleich. Es eignet sich somit der Wert 20 für m, da hier die Rechenzeit deutlich geringer ist als bei $m = 35$.

Nachbarschaftsgröße



(a) Kosten (b) Rechenzeit

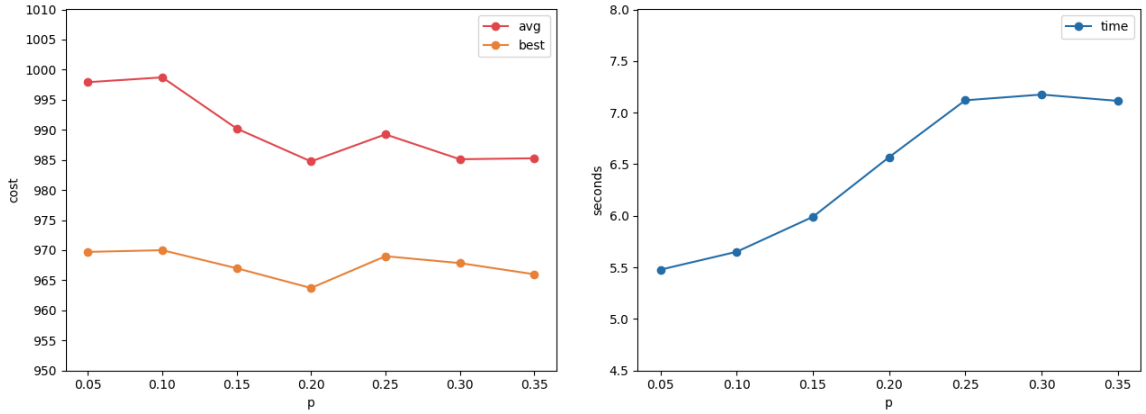
Abbildung 4.4.: Durchschnittliche Kosten, durchschnittlich beste Kosten und durchschnittliche Rechenzeit in Abhängigkeit von der Nachbarschaftsgröße

In Abbildung 4.4 ist zu sehen, dass die Kosten mit zunehmender Nachbarschaftsgröße zunächst sinken, allerdings ab einer Nachbarschaftsgröße von 80 nicht mehr weiter fallen bzw.

wieder leicht ansteigen. Die durchschnittliche Dauer eines Durchlaufs steigt mit zunehmender Nachbarschaftsgröße annähernd linear an. Es eignet sich somit der Wert 80 für die Begrenzung der Nachbarschaftsgröße, da für diesen Wert die Kosten am geringsten sind.

Häufigkeit des Neustarts

Der Neustart mit der bisher besten Lösung wird alle r Iterationen ohne Verbesserung durchgeführt, wobei r abhängig von der maximalen Iterationszahl ist, mit $r = p \cdot max\_iterations$.



(a) Kosten (b) Rechenzeit

Abbildung 4.5.: Durchschnittliche Kosten, durchschnittlich beste Kosten und durchschnittliche Rechenzeit in Abhängigkeit von p

Abbildung 4.5 zeigt, dass die Dauer der Durchläufe mit größer werdendem p, also mit sinkender Häufigkeit des Neustarts, ansteigt. Die durchschnittlichen Kosten fallen zunächst und sind für die Werte $0,2$, $0,3$ und $0,35$ am geringsten. Auch die Kosten der besten Lösungen sind für einen Wert von $p = 0,2$ am geringsten, womit sich dieser Wert am besten für p eignet.

4.3. Benutzeroberfläche

Die Benutzeroberfläche wurde als Webanwendung mit Hilfe von Flask[5], einem Framework für Python Webanwendungen, implementiert. In Abbildung 4.6 ist die Startseite mit bereits eingetragenen Beispieldaten dargestellt. Die Fahrzeugdaten, im Einzelnen die Fahrzeugkapazität, die frühestmögliche Abfahrtszeit am Depot sowie die Zieladresse, lassen sich manuell eingeben. Die Auftragsdaten, bestehend aus den Empfangsadressen, Paketgewichten sowie der oberen und unteren Begrenzung der Zeitfenster der Aufträge, können sowohl manuell eingegeben als auch aus einer Datei ausgelesen werden. So können die bei fox-Courier vorliegenden Dateien mit den Auftragsdaten direkt hochgeladen und weiterverarbeitet werden.

[5] `https://flask.palletsprojects.com/en/2.0.x/` (abgerufen am 10.01.2022)

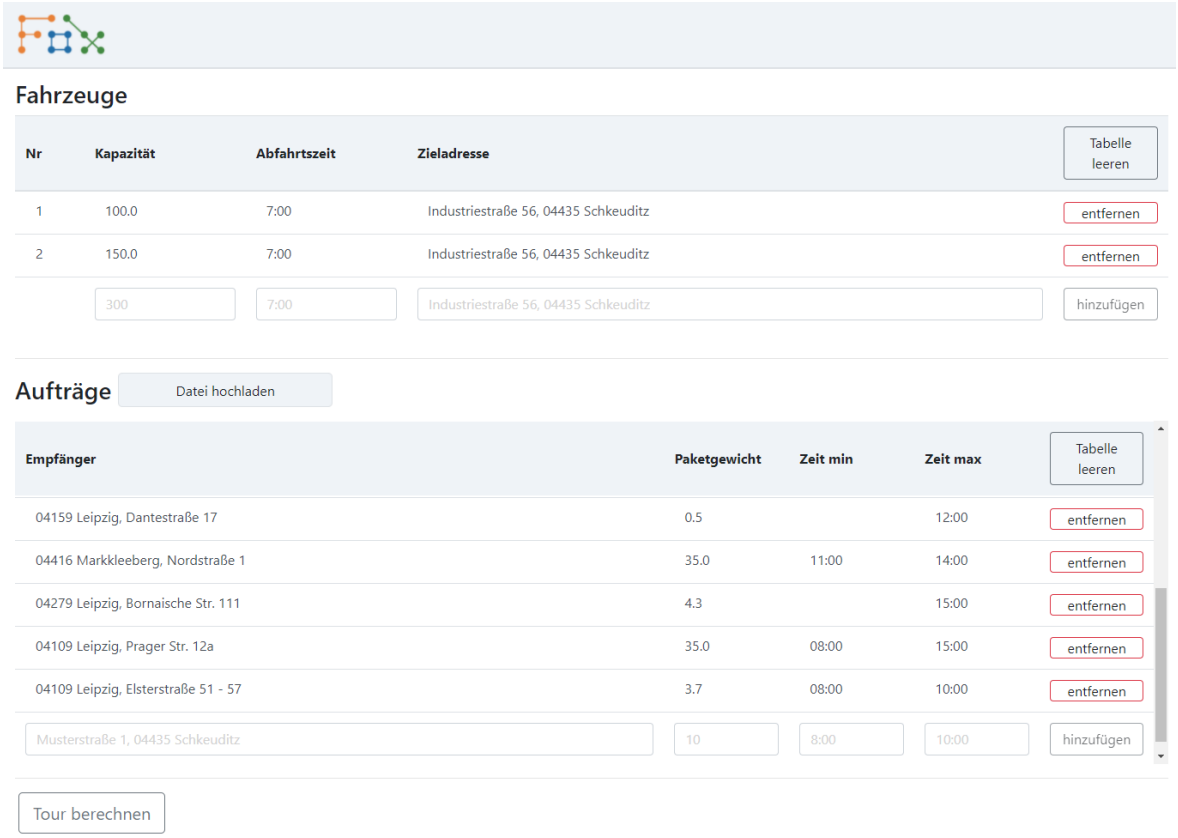


Abbildung 4.6.: Benutzeroberfläche: Dateneingabe

Mit einem Klick auf den Button „Tour berechnen“ werden die Eingabedaten zunächst vorverarbeitet, wobei fehlerhafte Eingaben sowie nicht existierende Adressen abgefangen werden. Anschließend wird die Tabu-Suche durchgeführt. Die beste gefundene Lösung wird auf einer neuen Seite sowohl in Listenform als auch grafisch dargestellt (Abbildung 4.7). Für die Erzeugung der Graphstrukturen wird das Python Package *NetworkX*[6] verwendet. Die Tabellen zu den jeweiligen Einzeltouren enthalten die Aufträge in der Reihenfolge, in der sie beliefert werden sollen mit den vorgegebenen Zeitfenstern und der berechneten Ankunftszeit.

Zu jeder Einzeltour wird außerdem das Gesamtgewicht der Pakete sowie die entsprechende Kapazität des verwendeten Fahrzeugs, die Gesamtlänge und -dauer der Tour und die Verspätung angegeben. Die Einzeltouren können zusätzlich per Buttonklick in Google Maps geöffnet werden. Zudem können die Aufträge der Touren einzeln bearbeitet werden für den Fall, dass bei einem Fahrzeug, das noch nicht vom Depot losgefahren ist, nachträglich Pakete hinzugefügt oder entfernt werden sollen. Für die geänderten Daten wird dann eine neue Tour berechnet.

[6] `https://networkx.org/` (abgerufen am 17.02.2022)

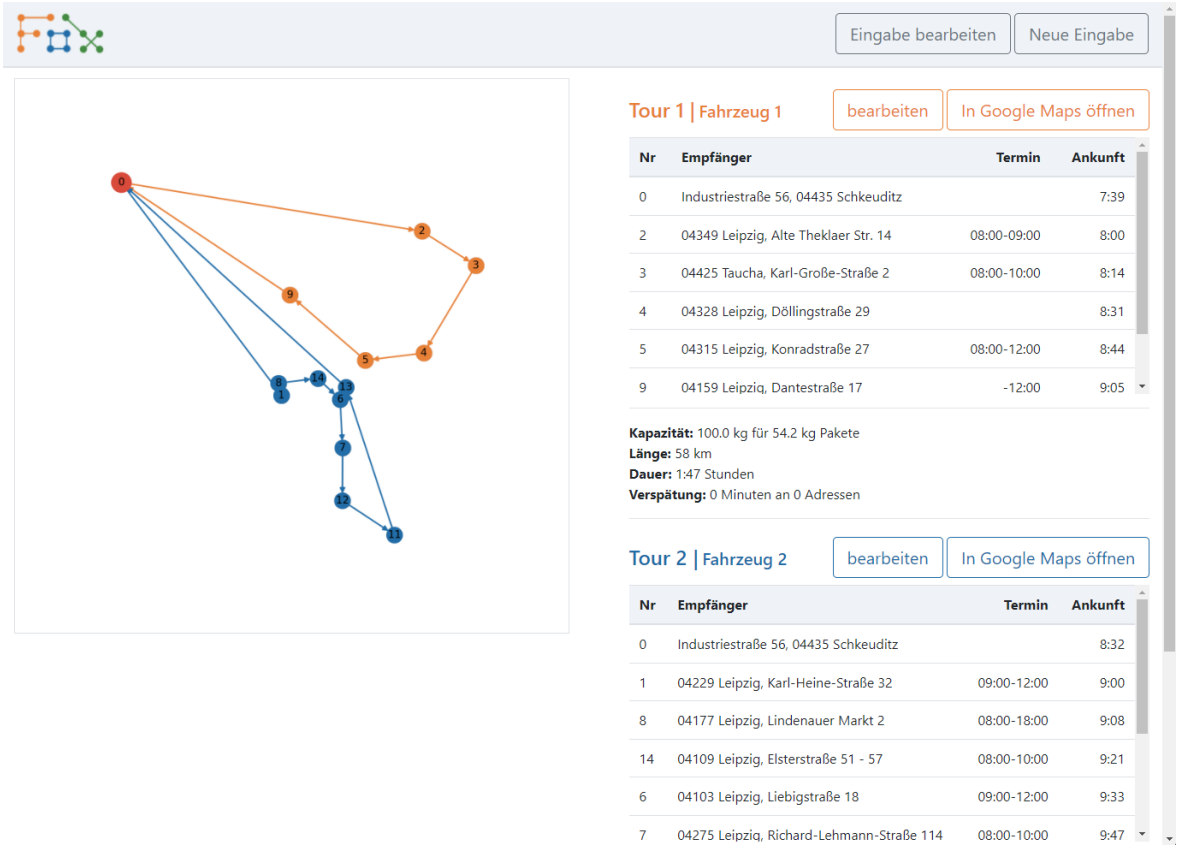


Abbildung 4.7.: Benutzeroberfläche: Berechnete Touren

5. Evaluierung

Für die Evaluierung des implementierten Tabu-Suchalgorithmus werden sowohl Benchmark-Daten als auch Daten des Unternehmens verwendet. Die Benchmark-Daten stammen aus der *Capacitated Vehicle Routing Problem Library*[7], in der VRP Instanzen sowie deren durch exakte Verfahren bestimmten optimalen Lösungen aus verschiedenen Arbeiten zusammengetragen und zur Verfügung gestellt werden. Es handelt sich dabei nicht um reale Adressdaten, sondern um Punkte in einem kartesischen Koordinatensystem, die entweder zufällig oder als Cluster generiert wurden. Für jede Instanz sind die Koordinaten der Punkte, die Nachfragewerte sowie die verfügbaren Fahrzeuge und deren Kapazität gegeben. Fahrtzeiten und Zeitfenster werden nicht berücksichtigt. Die Distanzen lassen sich durch die Berechnung der euklidischen Abstände zwischen den Punkten bestimmen. Da die angegebenen Werte der optimalen Lösungen aus gerundeten Distanzwerten berechnet werden, werden auch hier für die Vergleichbarkeit die berechneten Abstandswerte auf die nächste ganze Zahl gerundet [44].

Der von fox-Courier zur Verfügung gestellte Datensatz umfasst die Auftragsdaten von sieben Tagen mit den zu beliefernden Adressen, den Zeitfenstern und den Paketgewichten sowie die Information, wie die Aufträge auf die Fahrzeuge aufgeteilt und in welcher Reihenfolge sie jeweils ausgeliefert wurden. Es sind keine Angaben zu den Kapazitäten der Fahrzeuge vorhanden, weshalb für die Tabu-Suche die Kapazität auf die höchste auftretende Gewichtssumme einer Einzeltour der von fox-Courier geplanten Touren beschränkt wird.

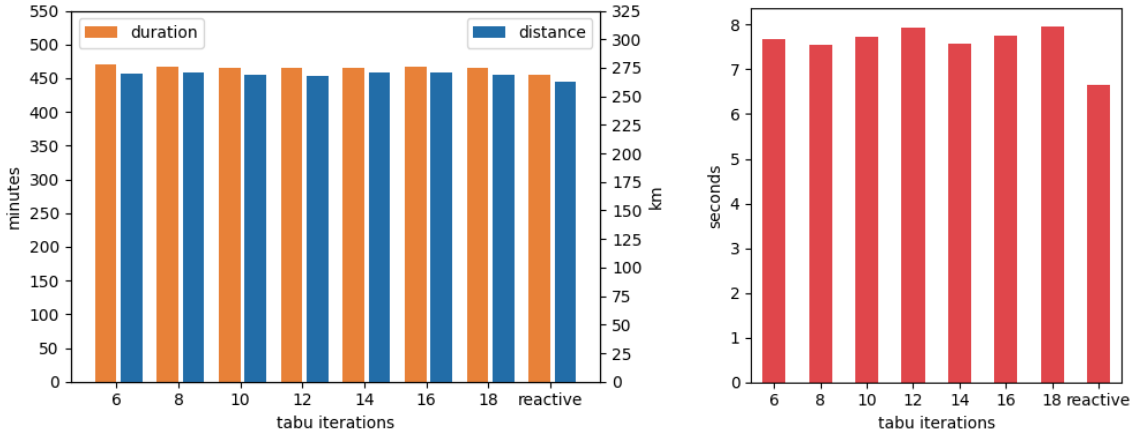
Zunächst wird anhand der Daten des Unternehmens geprüft, inwieweit die implementierten Mechanismen zu Verbesserungen führen. Anhand der Benchmark-Daten wird evaluiert, ob der Algorithmus die bestmögliche Lösung für verschiedene Instanzen findet. Weiterhin werden die Daten des Unternehmens genutzt, um die vom Algorithmus gefundenen Touren mit den manuell geplanten Touren von fox-Courier zu vergleichen.

5.1. Effekte der implementierten Mechanismen

Reaktiver Mechanismus

Abbildung 5.1 zeigt die durchschnittliche Distanz in Kilometern, die Fahrtdauer in Minuten sowie die durchschnittliche Rechenzeit in Sekunden bei verschiedenen statischen Tabu-Iterationswerten ohne reaktive Anpassung der Nachbarschaften im Vergleich zur Nutzung des reaktiven Mechanismus mit reaktiver Anpassung des Tabu-Iterationswerts und der Nachbarschaften. Es ist erkennbar, dass die Distanz und Dauer der Routen bei der Verwendung des reaktiven Mechanismus am geringsten sind. Die Distanz und die Fahrtdauer sind jeweils um

[7] `http://vrp.galgos.inf.puc-rio.br/index.php/en/` (abgerufen am 27.01.2022)



(a) Distanz und Fahrtdauer (b) Rechenzeit

Abbildung 5.1.: Vergleich der durchschnittlichen Distanz, Fahrtdauer und Rechenzeit von je zehn Durchläufen ohne reaktivem Mechanismus mit statischen Tabu-Iterationswerten und mit reaktivem Mechanismus

ca. 2,5% geringer als der Durchschnitt der Lösungen ohne reaktiven Mechanismus. Außerdem ist die Rechenzeit bei der Nutzung des reaktiven Mechanismus um etwa 13,9% geringer als der Durchschnitt der Rechenzeiten ohne die Nutzung des reaktiven Mechanismus.

Multiple Nachbarschaften

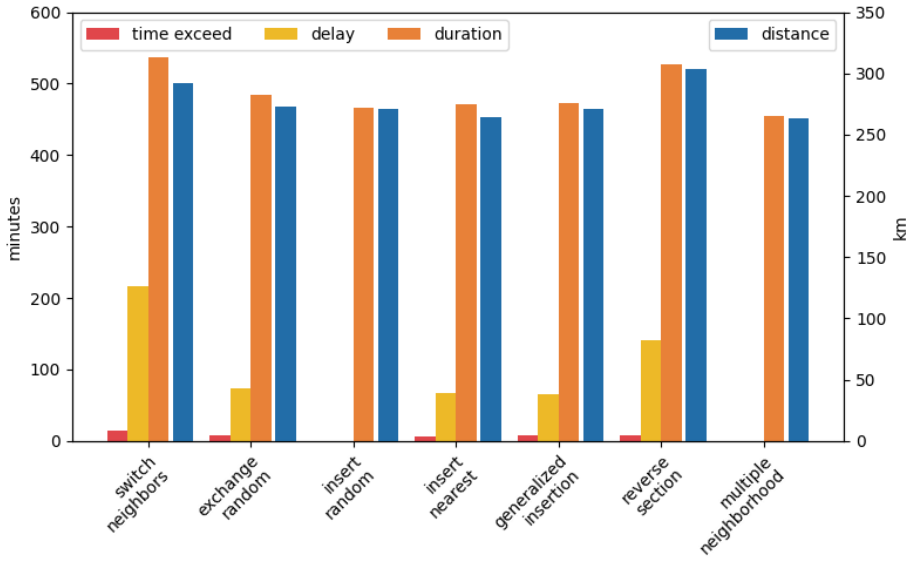
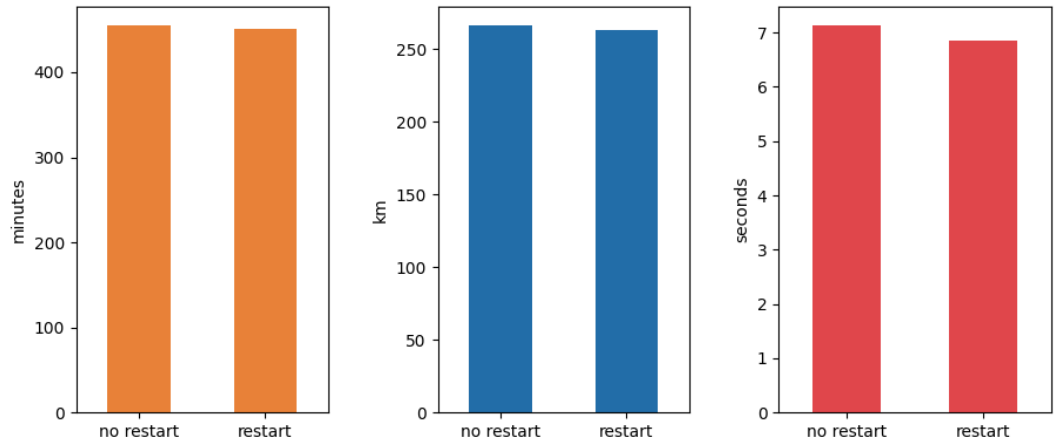


Abbildung 5.2.: Vergleich der durchschnittlichen Arbeitszeitüberschreitung, Verspätung, Fahrtdauer und Distanz von je zehn Durchläufen bei der Verwendung von einzelnen Schritten und der Verwendung von multiplen Nachbarschaften

In Abbildung 5.2 sind die durchschnittliche Arbeitszeitüberschreitung, Verspätung, Fahrtdauer in Minuten und Distanz in Kilometern bei der Verwendung von einzelnen Schritten zur Nachbarschaftserzeugung der Verwendung von multiplen Nachbarschaften, wobei die Schritte

kombiniert verwendet werden, gegenübergestellt. Es ist zu sehen, dass bei Durchläufen mit den einzelnen Schrittarten nicht zuverlässig Touren ohne Arbeitszeitüberschreitung und Verspätung gefunden werden können, außer bei der Verwendung des Schritts *insert random*. Bei der Nutzung von multiplen Nachbarschaften können die Zeitüberschreitungen und Verspätungen vollständig vermieden werden und zusätzlich ist die durchschnittliche Fahrtzeit um etwa 2,5% und die Distanz um ca. 3% geringer als bei *insert random*. Die Tabu-Suche findet somit durch die Kombination der Schritte bessere Touren als bei der Verwendung von einzelnen Schritten.

Neustart-Mechanismus



(a) Fahrtdauer (b) Distanz (c) Rechenzeit

Abbildung 5.3.: Vergleich der durchschnittlichen Distanz, Fahrtdauer und Rechenzeit aus jeweils zehn Durchläufen mit und ohne Neustart

Abbildung 5.3 zeigt die durchschnittliche Distanz in Kilometern, die Fahrtdauer in Minuten sowie die Rechenzeit in Sekunden mit und ohne die Verwendung des Neustart-Mechanismus. Die durchschnittliche Distanz und Dauer verringern sich minimal beim Einsatz des Neustart-Mechanismus um ca. 0,14% bzw. 0,23%. Die Rechenzeit kann durch die Nutzung des Neustart-Mechanismus um etwa 7% verringert werden.

5.2. Evaluierung mit Benchmark-Daten

Die Benchmark-Daten decken ein weites Spektrum an verschiedenen Probleminstanzen ab, von Problemen mit 15 bis 30000 Aufträgen und mit zwei bis über 200 verfügbaren Fahrzeugen. Für die Evaluierung werden jedoch nur Instanzen verwendet, die einen ähnlichen Umfang wie die Probleme des Unternehmens haben, also mit höchstens 35 Aufträgen und bis zu fünf Fahrzeugen.

Instanz	n	k	Tabu-Suche		Optimal	Abweichung
			best	avg		
A-n32-k5	31	5	**784**	792,6	784	1,10%
A-n33-k5	32	4	**661**	668,0	661	1,06%
A-n34-k5	33	5	**778**	784,0	778	0,77%
A-n36-k5	35	5	815	823,9	799	3,12%
B-n31-k5	30	5	673	679,7	672	1,15%
B-n34-k5	33	5	**788**	791,1	788	0,39%
B-n35-k5	34	5	958	975,3	955	2,13%
E-n22-k4	21	4	**375**	**375,0**	375	**0,00%**
E-n23-k3	22	3	**569**	**569,0**	569	**0,00%**
E-n30-k3	29	3	**534**	**534,0**	534	**0,00%**
E-n33-k4	32	4	**835**	845,3	835	1,23%
P-n19-k2	18	2	**212**	213,6	212	0,75%
P-n20-k2	19	2	**216**	**216,0**	216	**0,00%**
P-n21-k2	20	2	**211**	**211,0**	211	**0,00%**
P-n22-k2	21	2	**216**	217,1	216	0,51%

Tabelle 5.1.: Vergleich der besten und durchschnittlichen Distanzen aus je zehn Durchläufen der Tabu-Suche mit den Distanzen der optimalen Lösungen der Benchmark-Daten

In Tabelle 5.1 sind für die verschiedenen Instanzen die Anzahl der Aufträge n und die Anzahl der Fahrzeuge k dargestellt sowie die Distanz der jeweils besten Tour aus zehn Durchläufen und die durchschnittliche Distanz. Dem gegenübergestellt ist die Distanz der optimalen Lösung für die jeweilige Instanz. Außerdem ist die Abweichung der durchschnittlichen Distanz der Lösung der Tabu-Suche von der optimalen Lösung angegeben.

Für fast alle Instanzen findet die Tabu-Suche in mindestens einem der zehn Durchläufe die optimale Lösung. Für fünf der Instanzen wird die optimale Lösung zuverlässig in jedem der zehn Durchläufe gefunden und für die anderen Instanzen weichen die durchschnittlichen Lösungen nur wenig vom Optimum ab. Die höchsten Abweichungen treten bei Instanzen mit 30 oder mehr Aufträgen und/oder fünf verwendeten Fahrzeugen auf. Durchschnittlich liegt die Abweichung für alle Instanzen bei ca. 0,81%.

5.3. Vergleich mit Touren des Unternehmens

In Tabelle 5.2 sind für die jeweiligen Instanzen der Unternehmensdaten die Anzahl der Aufträge n angegeben und die Anzahl der verwendeten Fahrzeuge k, die Gesamtdistanz, die Gesamtdauer und die Gesamtverspätung der von fox-Courier geplanten Touren den Werten

n	fox-Courier				Tabu-Suche			
	k	Distanz	Dauer	Verspätung	k	Distanz	Dauer	Verspätung
26	2	285	604	162	2,0	225,0	469,0	0
27	4	273	853	0	2,6	179,1	553,8	0
23	2	233	435	0	2,0	163,6	384,2	0
13	2	190	282	15	1,0	95,2	217,0	0
20	3	603	871	0	2,0	543,2	567,0	0
27	2	512	691	351	2,0	435,7	623,8	0
30	3	348	713	0	2,0	205,2	462,4	0
Ø	2,6	349,1	635,6	75,4	1,9	263,9	468,2	0

Tabelle 5.2.: Vergleich der durchschnittlichen Distanz, Fahrtdauer, Verspätung und Anzahl der Fahrzeuge aus zehn Durchläufen der Tabu-Suche mit den manuell geplanten Touren von fox-Courier

der von der Tabu-Suche gefundenen Lösungen gegenübergestellt. Die Distanz ist in Kilometern angegeben und die Dauer und Verspätung in Minuten. Die Distanzen, Fahrtzeiten und Verspätungen der von fox-Courier geplanten Touren wurden entsprechend der angegebenen Reihenfolge und Aufteilung der Aufträge anhand der in der Vorverarbeitung ermittelten Distanzen und Fahrtzeiten bestimmt.

Der Vergleich der Werte zeigt, dass die Tabu-Suche für alle Instanzen kürzere und schnellere Touren als die von fox-Courier geplanten Touren findet. Durchschnittlich wird die Distanz um etwa 85 Kilometer reduziert und die Fahrtzeit um ca. 167 Minuten. Zusätzlich verringert sich die durchschnittliche Anzahl an verwendeten Fahrzeugen um 0,7. Verspätete Auslieferungen werden bei allen Touren der Tabu-Suche verhindert.

5.4. Diskussion

Die Evaluierung der verwendeten Mechanismen anhand der Daten des Unternehmens zeigt, dass durch alle Mechanismen eine Reduzierung der Rechenzeit und/oder eine Verbesserung der Lösungsqualität erreicht werden kann, auch wenn diese teilweise nur gering ist. Außerdem werden durch die Verwendung von multiplen Nachbarschaften Arbeitszeitüberschreitungen und Verspätungen verhindert, welche bei der Verwendung von einzelnen Schritten nicht immer zuverlässig vermiedenen werden können. Trotz des geringen Umfangs des Datensatzes sind die gezeigten Ergebnisse ein Indikator dafür, dass die implementierten Mechanismen allgemein zu einer Verbesserung des Algorithmus beitragen.

Die Evaluierung mit den Benchmark-Daten zeigt, dass der Tabu-Suchalgorithmus für alle Instanzen qualitativ hochwertige Lösungen findet. Der Algorithmus ist in der Lage, für die meisten Instanzen die optimale Lösung zu finden, auch wenn diese nicht für alle Instanzen zuverlässig in jedem Durchlauf gefunden wird. Die größten Abweichungen vom Optimum

treten bei Instanzen mit fünf Fahrzeugen bzw. 30 oder mehr Aufträgen auf. Bei Instanzen mit zwei bis vier Fahrzeugen und bis zu 30 Aufträgen, welche eher dem Umfang der bei fox-Courier vorkommenden Probleminstanzen entsprechen, werden zuverlässiger die optimalen Lösungen gefunden.

Es ist anzumerken, dass die Ergebnisse der Evaluierung mit den Benchmark-Daten nur eingeschränkt aussagekräftig für die Bewertung der Fähigkeit des implementierten Algorithmus für die Lösung des vorliegenden Problems des Unternehmens sind, da sich die Benchmark-Daten von den Auftragsdaten des Unternehmens unterscheiden. Bei den Benchmark-Daten ist allein die Distanz unter Berücksichtigung der Fahrzeugkapazitäten zu minimieren, wohingegen bei den realen Auftragsdaten die Fahrtdauer, die Zeitfenster und die Arbeitszeitbeschränkungen mit berücksichtigt werden müssen. Da der Algorithmus jedoch gute Ergebnisse für das grundlegende kapazitive VRP liefert, lässt sich annehmen, dass der Algorithmus auch für das vorliegende VRP des Unternehmens Lösungen mit geringen Distanzen unter Berücksichtigung der Fahrzeugkapazitäten finden kann.

Der gezeigte Vergleich mit den Daten des Unternehmens gibt keinen Aufschluss darüber, ob der Algorithmus für diese Daten die optimalen Lösungen findet, jedoch wird gezeigt, dass für alle Instanzen bessere Lösungen als die manuell geplanten Touren gefunden werden. Alle von der Tabu-Suche gefundenen Touren haben sowohl eine kürzere Distanz und Fahrtdauer als die manuell geplanten Touren als auch durchschnittlich eine geringere Anzahl an benötigten Fahrzeugen. Es ist anzumerken, dass die Evaluierung nicht gänzlich vergleichbar mit der realen Routenplanung im Unternehmen ist, da die Angabe zu den Kapazitäten der verwendeten Fahrzeuge fehlt. Jedoch wurde ein realistischer Wert für die Kapazitätsbeschränkung in der Evaluierung verwendet, weshalb angenommen werden kann, dass die Routenplanung mit der Tabu-Suche das Potenzial hat, auch beim Einsatz im Unternehmen bessere als die manuell geplanten Touren zu finden.

6. Zusammenfassung & Ausblick

Ziel dieser Arbeit war die Entwicklung einer Anwendung für die Planung der Auslieferungsrouten des Logistikunternehmens fox-Courier. Für das vorliegende *Vehicle Routing Problem* mit Kapazitätsbeschränkungen, Zeitfenstern sowie weiteren Nebenbedingungen wurde ein auf der Tabu-Suche basierender Optimierungsalgorithmus implementiert, welcher um verschiedene Erweiterungen, wie einen reaktiven Mechanismus, einen Neustart-Mechanismus sowie multiple Nachbarschaften, ergänzt wurde. Bei der Evaluierung mit Auftragsdaten des Unternehmens konnte gezeigt werden, dass alle Mechanismen zu einer Verbesserung des Algorithmus beitragen.

Die Evaluierung des Tabu-Suchalgorithmus mit den Benchmark-Daten hat gezeigt, dass der Algorithmus in der Lage ist, für die meisten der Instanzen die optimale Lösung zu finden, diese jedoch bei einigen Instanzen nicht zuverlässig in jedem Durchlauf gefunden werden. Auffällig war auch, dass die höchsten Abweichungen vom Optimum bei Instanzen mit vergleichsweise höheren Auftragsmengen bzw. mehr verwendeten Fahrzeugen auftraten. Interessant wäre daher zu untersuchen, wie sich die Lösungsqualität für solche Instanzen verbessern lassen kann und wie die Reliabilität des Algorithmus erhöht werden kann.

Für die Erhöhung der Zuverlässigkeit oder der Verbesserung der Lösungsqualität im Allgemeinen kommen verschiedene Ansätze in Frage. Vielversprechend wäre beispielsweise die Beschleunigung der Suche etwa durch eine Parallelisierung, wodurch in gleicher Zeit größere Teile des Suchraums betrachtet werden können. Interessant wäre auch zu untersuchen, ob durch das Ergänzen des Algorithmus um ein *Adaptive Memory* oder weitere erfolgversprechende Mechanismen für Optimierungsalgorithmen Verbesserungen erzielt werden können. Da in der Literatur mehrfach auch hybride Ansätze für kombinatorische Optimierungsprobleme vorgeschlagen werden, könnte die Kombination des Tabu-Suchalgorithmus mit z.B. einem genetischen Algorithmus oder dem *Simulated Annealing* für die Lösung des vorliegenden VRP untersucht werden.

Als initiale Lösung für den Start der Tabu-Suche wurde auf 1000 zufällig erzeugte Lösungen der *Nearest Neighbor* Algorithmus angewandt und daraus die beste Lösung ausgewählt. Gegenstand zukünftiger Arbeiten könnte sein, zu untersuchen, ob durch andere Methoden zur Erzeugung initialer Lösungen, wie z.B. den *Clarke and Wright* Algorithmus oder den *Sweep* Algorithmus, die Qualität der von der Tabu-Suche gefundenen Lösungen verbessert werden kann.

Zusätzlich zur Evaluierung mit den Benchmark-Daten wurden die Lösungen des Tabu-Suchalgorithmus mit manuell geplanten Touren des Unternehmens verglichen. Hierfür standen die Auftragsdaten sowie in welcher Reihenfolge und mit wie vielen Fahrzeugen diese vom Unternehmen ausgeliefert wurden, von insgesamt sieben Werktagen zur Verfügung. Beim Vergleich zeigte sich, dass die Tabu-Suche für all diese Probleminstanzen sowohl kürzere

als auch schnellere Routen finden konnte. Auch wenn der Umfang des zur Verfügung stehenden Datensatzes recht gering ist und Angaben zu den Fahrzeugkapazitäten fehlen, lassen die Ergebnisse die Vermutung zu, dass durch den Einsatz der entwickelten Anwendung Einsparungen durch reduzierte Fahrtkosten und Arbeitszeit für das Unternehmen erzielt werden können. Außerdem ist ein Durchlauf der Tabu-Suche mit höchstens zehn Sekunden deutlich schneller als das manuelle Planen der Routen, wodurch auch an dieser Stelle Arbeitszeit eingespart werden kann, selbst falls mehrere Durchläufe aufgrund einer geänderten Auftragslage notwendig sein sollten. Jedoch muss sich der tatsächliche Nutzen der Anwendung in einer Testphase im Unternehmen zeigen, die in naher Zukunft erfolgen soll.

Für eine aussagekräftigere Evaluierung mit den Unternehmensdaten sowie eine effektivere Parameteroptimierung wäre es von Vorteil, die optimalen Lösungen der Instanzen zu kennen. In zukünftigen Arbeiten könnte ein exakter Algorithmus, wie beispielsweise der *Branch and Bound* Algorithmus, implementiert werden, um die optimalen Touren für die Probleminstanzen zu ermitteln und um anschließend zu untersuchen, ob die Tabu-Suche in der Lage ist, diese optimalen Lösungen zu finden.

Einen weiteren Teil der Anwendung stellt die entwickelte Benutzeroberfläche dar. Diese ermöglicht die Eingabe der Auftragsdaten und zeigt die berechneten Touren grafisch und in Listenform mit zusätzlichen Informationen zu den berechneten Ankunftszeiten, Verspätungen und der Distanz und Fahrzeit. Die Teilrouten lassen sich per Buttonklick in Google Maps öffnen. Gegenstand zukünftiger Arbeiten könnte sein, die grafische Darstellung der Routen durch eine Karte zu ersetzen, auf der die Routen direkt angezeigt werden. Vorstellbar wäre auch, interaktive Änderungen der Routen auf der Karte zu ermöglichen.

Weiterhin wäre die Erweiterung der Anwendung um die Berechnung der Routen auf Basis von Echtzeitverkehrsdaten denkbar, wodurch Fahrtzeitverzögerungen und Umleitungen durch Staus, Baustellen und ähnliches bei der Bestimmung der Distanzen und Fahrtzeiten mit einbezogen werden könnten. Für die Bestimmung von Distanzen und Fahrzeiten auf Basis von Echtzeitverkehrsdaten existieren allerdings nach aktuellem Kenntnisstand keine kostenfreien Dienste. Geeignet wäre beispielsweise die kostenpflichtige erweiterte Distanzmatrix API von Google Maps[8]. Allerdings müsste abgewogen werden, ob die Auswirkungen des Verkehrs auf die Routen so groß ist, dass sich die Verwendung eines kostenpflichtigen Dienstes rentiert.

[8] `https://developers.google.com/maps/documentation/distance-matrix/overview#distance-matrix-advanced` (abgerufen am 24.02.2022)

Literatur

[1] Christian Blum und Andrea Roli. „Metaheuristics in combinatorial optimization: Overview and conceptual comparison“. In: *ACM Computing Surveys* 35.3 (1. Sep. 2003), S. 268–308. DOI: 10.1145/937503.937505.

[2] Ibrahim H. Osman und Gilbert Laporte. „Metaheuristics: A bibliography“. In: *Annals of Operations Research* 63.5 (1. Okt. 1996), S. 511–623. DOI: 10.1007/BF02125421.

[3] G. B. Dantzig und J. H. Ramser. „The Truck Dispatching Problem“. In: *Management Science* 6.1 (1. Okt. 1959), S. 80–91. DOI: 10.1287/mnsc.6.1.80.

[4] J-F Cordeau, M Gendreau, G Laporte, J-Y Potvin und F Semet. „A guide to vehicle routing heuristics“. In: *Journal of the Operational Research Society* 53.5 (Mai 2002), S. 512–522. DOI: 10.1057/palgrave.jors.2601319.

[5] Tonci Caric und Hrvoje Gold. *Vehicle Routing Problem*. IntechOpen, 2008. DOI: 10.5772/64.

[6] Fred Glover. „Future paths for integer programming and links to artificial intelligence“. In: *Computers & Operations Research*. Applications of Integer Programming 13.5 (1. Jan. 1986), S. 533–549. DOI: 10.1016/0305-0548(86)90048-1.

[7] Sumanta Basu. „Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey“. In: *American Journal of Operations Research* 02 (Jan. 2012). DOI: 10.4236/ajor.2012.22019.

[8] G. Laporte, M. Gendreau, J-Y. Potvin und F. Semet. „Classical and modern heuristics for the vehicle routing problem“. In: *International Transactions in Operational Research* 7.4 (2000), S. 285–300. DOI: 10.1111/j.1475-3995.2000.tb00200.x.

[9] Michel Gendreau, Alain Hertz und Gilbert Laporte. „New Insertion and Postoptimization Procedures for the Traveling Salesman Problem“. In: *Operations Research* 40 (1. Dez. 1992), S. 1086–1094. DOI: 10.1287/opre.40.6.1086.

[10] The Jin Ai und Voratas Kachitvichyanukul. „Particle swarm optimization and two solution representations for solving the capacitated vehicle routing problem“. In: *Computers & Industrial Engineering* 56.1 (1. Feb. 2009), S. 380–387. DOI: 10.1016/j.cie.2008.06.012.

[11] Yannis Marinakis, Magdalene Marinaki und Athanasios Migdalas. „A multi-adaptive particle swarm optimization for the vehicle routing problem with time windows“. In: *Information Sciences* 481 (Mai 2019), S. 311–329. DOI: 10.1016/j.ins.2018.12.086.

[12] Farah Belmecheri, Christian Prins, Farouk Yalaoui und Lionel Amodeo. „Particle swarm optimization algorithm for a vehicle routing problem with heterogeneous fleet, mixed backhauls, and time windows“. In: *Journal of Intelligent Manufacturing* 24.4 (1. Aug. 2013), S. 775–789. DOI: 10.1007/s10845-012-0627-8.

[13] Ai-ling Chen, Gen-ke Yang und Zhi-ming Wu. „Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem“. In: *Journal of Zhejiang University-SCIENCE A* 7.4 (1. Apr. 2006), S. 607–614. DOI: 10.1631/jzus.2006.A0607.

[14] S.-W. Lin, K.-C. Ying, Z.-J. Lee und F.-H. Hsi. „Applying Simulated Annealing Approach for Capacitated Vehicle Routing Problems“. In: *2006 IEEE International Conference on Systems, Man and Cybernetics*. Bd. 1. Okt. 2006, S. 639–644. DOI: 10.1109/ICSMC.2006.384457.

[15] Lijun Wei, Zhenzhen Zhang, Defu Zhang und Stephen C. H. Leung. „A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints“. In: *European Journal of Operational Research* 265.3 (16. März 2018), S. 843–859. DOI: 10.1016/j.ejor.2017.08.035.

[16] Raúl Baños, Julio Ortega, Consolación Gil, Antonio Fernández und Francisco de Toro. „A Simulated Annealing-based parallel multi-objective approach to vehicle routing problems with time windows“. In: *Expert Systems with Applications* 40.5 (1. Apr. 2013), S. 1696–1707. DOI: 10.1016/j.eswa.2012.09.012.

[17] Jean Berger und Mohamed Barkaoui. „A Hybrid Genetic Algorithm for the Capacitated Vehicle Routing Problem“. In: *Genetic and Evolutionary Computation — GECCO 2003*. Hrsg. von Erick Cantú-Paz et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, S. 646–656. DOI: 10.1007/3-540-45105-6_80.

[18] Habibeh Nazif und Lai Soon Lee. „Optimised crossover genetic algorithm for capacitated vehicle routing problem“. In: *Applied Mathematical Modelling* 36.5 (1. Mai 2012), S. 2110–2117. DOI: 10.1016/j.apm.2011.08.010.

[19] Yannis Marinakis und Magdalene Marinaki. „A hybrid genetic – Particle Swarm Optimization Algorithm for the vehicle routing problem“. In: *Expert Systems with Applications* 37.2 (1. März 2010), S. 1446–1455. DOI: 10.1016/j.eswa.2009.06.085.

[20] Ran Liu und Zhibin Jiang. „A hybrid large-neighborhood search algorithm for the cumulative capacitated vehicle routing problem with time-window constraints“. In: *Applied Soft Computing* 80 (1. Juli 2019), S. 18–30. DOI: 10.1016/j.asoc.2019.03.008.

[21] A. E. Rizzoli, R. Montemanni, E. Lucibello und L. M. Gambardella. „Ant colony optimization for real-world vehicle routing problems“. In: *Swarm Intelligence* 1.2 (1. Dez. 2007), S. 135–151. DOI: 10.1007/s11721-007-0005-x.

[22] Bin Yu, Zhong-Zhen Yang und Baozhen Yao. „An improved ant colony optimization for vehicle routing problem“. In: *European Journal of Operational Research* 196.1 (1. Juli 2009), S. 171–176. DOI: 10.1016/j.ejor.2008.02.028.

[23] Qiulei Ding, Xiangpei Hu, Lijun Sun und Yunzeng Wang. „An improved ant colony optimization and its application to vehicle routing problem with time windows“. In: *Neurocomputing*. Bio-inspired computing and applications 98 (3. Dez. 2012), S. 101–107. DOI: 10.1016/j.neucom.2011.09.040.

[24] Chou-Yuan Lee, Zne-Jung Lee, Shih-Wei Lin und Kuo-Ching Ying. „An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem“. In: *Appl. Intell.* 32 (1. Feb. 2010), S. 88–95. DOI: 10.1007/s10489-008-0136-9.

[25] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V. Snyder und Martin Takáč. „Reinforcement Learning for Solving the Vehicle Routing Problem“. In: *CoRR* abs/1802.04240 (2018). arXiv: 1802.04240.

[26] Wouter Kool, Herke Van Hoof und Max Welling. *Attention, Learn to Solve Routing Problems!* 2018. arXiv: 1803.08475.

[27] André Hottung und Kevin Tierney. „Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem“. In: *ECAI 2020* (2020). Publisher: IOS Press, S. 443–450. DOI: 10.3233/FAIA200124.

[28] Yves Rochat und Éric D. Taillard. „Probabilistic diversification and intensification in local search for vehicle routing“. In: *Journal of Heuristics* 1.1 (Sep. 1995), S. 147–167. DOI: 10.1007/BF02430370.

[29] C. D. Tarantilis. „Solving the vehicle routing problem with adaptive memory programming methodology“. In: *Computers & Operations Research* 32.9 (1. Sep. 2005), S. 2309–2327. DOI: 10.1016/j.cor.2004.03.005.

[30] Jalel Euchi und Habib Chabchoub. „A hybrid tabu search to solve the heterogeneous fixed fleet vehicle routing problem“. In: *Logistics Research* 2.1 (Juni 2010), S. 3–11. DOI: 10.1007/s12159-010-0028-3.

[31] Xiangyong Li, Stephen C. H. Leung und Peng Tian. „A multistart adaptive memory-based tabu search algorithm for the heterogeneous fixed fleet open vehicle routing problem“. In: *Expert Systems with Applications* 39.1 (1. Jan. 2012), S. 365–374. DOI: 10.1016/j.eswa.2011.07.025.

[32] Roberto Battiti und Giampietro Tecchiolli. „The Reactive Tabu Search“. In: *ORSA journal on computing* 6.2 (1994), S. 126–140.

[33] Wen-Chyuan Chiang und Robert A. Russell. „A Reactive Tabu Search Metaheuristic for the Vehicle Routing Problem with Time Windows“. In: *INFORMS Journal on Computing* 9.4 (1997). Publisher: INFORMS: Institute for Operations Research, S. 417. DOI: 10.1287/ijoc.9.4.417.

[34] Olli Bräysy. „A Reactive Variable Neighborhood Search for the Vehicle-Routing Problem with Time Windows“. In: *INFORMS Journal on Computing* 15.4 (2003). Publisher: INFORMS: Institute for Operations Research, S. 347–368. DOI: 10.1287/ijoc.15.4.347.24896.

[35] Paolo Toth und Daniele Vigo. „The Granular Tabu Search and Its Application to the Vehicle-Routing Problem“. In: *INFORMS Journal on Computing* 15.4 (2003). Publisher: INFORMS: Institute for Operations Research, S. 333–346. DOI: 10.1287/ijoc.15.4.333.24890.

[36] Jose Bernal, John Willmer Escobar und Rodrigo Linfati. „A granular tabu search algorithm for a real case study of a vehicle routing problem with a heterogeneous fleet and time windows“. In: *Journal of Industrial Engineering and Management* 10.4 (26. Okt. 2017), S. 646. DOI: 10.3926/jiem.2159.

[37] Jianyong Jin, Teodor Gabriel Crainic und Arne Løkketangen. „A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems“. In: *European Journal of Operational Research* 222.3 (Nov. 2012), S. 441–451. DOI: 10.1016/j.ejor.2012.05.025.

[38] John Willmer Escobar, Rodrigo Linfati, Paolo Toth und Maria G. Baldoquin. „A hybrid Granular Tabu Search algorithm for the Multi-Depot Vehicle Routing Problem“. In: *Journal of Heuristics* 20.5 (1. Okt. 2014), S. 483–509. DOI: 10.1007/s10732-014-9247-0.

[39] Jean-François Cordeau und Mirko Maischberger. „A parallel iterated tabu search heuristic for vehicle routing problems“. In: *Computers & Operations Research* 39.9 (1. Sep. 2012), S. 2033–2050. DOI: 10.1016/j.cor.2011.09.021.

[40] Philippe Badeau, François Guertin, Michel Gendreau, Jean-Yves Potvin und Eric Taillard. „A parallel tabu search heuristic for the vehicle routing problem with time windows“. In: *Transportation Research Part C: Emerging Technologies* 5.2 (1997), S. 109–122. DOI: https://doi.org/10.1016/S0968-090X(97)00005-3.

[41] Jari Kytöjoki, Teemu Nuortio, Olli Bräysy und Michel Gendreau. „An efficient variable neighborhood search heuristic for very large scale vehicle routing problems“. In: *Computers & Operations Research* 34.9 (Sep. 2007), S. 2743–2757. DOI: 10.1016/j.cor.2005.10.010.

[42] Yangkun Xia und Zhuo Fu. „Improved tabu search algorithm for the open vehicle routing problem with soft time windows and satisfaction rate“. In: *Cluster Computing* 22.4 (1. Juli 2019), S. 8725–8733. DOI: 10.1007/s10586-018-1957-x.

[43] José Brandão. „A tabu search algorithm for the open vehicle routing problem“. In: *European Journal of Operational Research* 157.3 (Sep. 2004), S. 552–564. DOI: 10.1016/S0377-2217(03)00238-8.

[44] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal und Anand Subramanian. „New benchmark instances for the Capacitated Vehicle Routing Problem“. In: *European Journal of Operational Research* 257.3 (März 2017), S. 845–858. DOI: 10.1016/j.ejor.2016.08.012.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit mit dem Thema:

„Tabu Search Algorithmus für das Vehicle Routing Problem eines Logistikunternehmens“

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, den 21.03.2022

Lea Christin Löffelmann