h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

Seminararbeit

Vehicle Routing Problem

Name: Max Uhl

Matrikelnummer: 772773

E-Mail: max.uhl@stud.h-da.de

Abgabe: 20.06.2024

Seminar: Heuristische Optimierungsverfahren

Betreuer: Prof. Dr. Tobias Bedenk

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CVRP – Capacitated Vehicle Routing Problem

TSP – Traveling Salesman Problem

VRP – Vehicle Routing Problem

LNS – Large Neighbourhood Search

NLNS – Neural Large Neighbourhood Search

LKH3 – Lin-Kerhinghan-Helsaun TSP Solver Version 3

UHGS – Unified Hybrid Genetic Search

1 Einleitung

In der heutigen leistungsorientierten Welt spielt die effiziente Planung und Steuerung von Transport- und Logistikprozessen eine entscheidende Rolle. Unternehmen sind ständig auf der Suche nach Methoden, um ihre Lieferketten zu optimieren, Kosten zu senken und den Kundenservice zu verbessern. Ein zentraler Bestandteil dieser Bemühungen ist das Vehicle Routing Problem (VRP), ein komplexes Optimierungsproblem, das seit Jahrzehnten im Fokus der wissenschaftlichen Forschung und praktischen Anwendung steht.

Das VRP befasst sich mit der optimalen Planung von Routen für eine Flotte von Fahrzeugen, die eine bestimmte Anzahl von Kunden von einem oder mehreren Depots aus beliefern sollen. Ziel ist es, die Gesamtkosten zu minimieren, welche häufig durch Faktoren wie Fahrstrecke, Zeit und eingesetzte Ressourcen bestimmt werden. Die Vielfalt der Anwendungsfälle reicht von der Paketzustellung über die Abfallentsorgung bis hin zur Versorgung von Supermärkten und der Instandhaltung technischer Anlagen.

Um die Herausforderungen des VRP zu bewältigen, wurden zahlreiche heuristische und metaheuristische Verfahren entwickelt. Diese Methoden bieten praktikable Lösungsansätze, insbesondere für große und komplexe Problemstellungen, die mit exakten Algorithmen auf Grund ihrer Komplexität nicht innerhalb eines akzeptablen Zeitfensters gelöst werden können. Innerhalb dieser Seminararbeit werden sowohl konstruktive Heuristiken wie das Nearest Neighbour-Verfahren und verschiedene Einfügeverfahren als auch Intra- und Inter-Route Verbesserungsheuristiken und metaheuristische Ansätze wie der Large Neighbourhood Search (LNS) detailliert untersucht.

Darüber hinaus wird die momentan performanteste Kombination aus Deep Learning Modell und Metaheuristik vorgestellt, um das Potential der Kooperation beider Disziplinen hervorzuheben.

Ein Vergleich der verschiedenen konstruktiven Heuristiken anhand ausgewählter Benchmarkprobleme soll schließlich die Leistungsfähigkeit und Anwendbarkeit der einzelnen Methoden aufzeigen und eine fundierte Diskussion der Ergebnisse ermöglichen.

Diese Arbeit bietet somit eine umfassende Einführung in das VRP, eine detaillierte Analyse der gängigen heuristischen und metaheuristischen Methoden sowie einen Einblick in aktuelle Forschungstrends, praktische Anwendungen und zeigt dem Leser, welche konstruktiven Heuristiken sich in der Praxis bewähren können.

2 Einführung

2.1 Definition VRP

Das Vehicle Routing Problem (VRP) ist ein generischer Begriff für eine ganze Klasse von Problemen, welche sich mit der optimalen Gestaltung von Routen befassen, die von einer Flotte von Fahrzeugen verwendet werden sollen, um eine Gruppe von Kunden zu bedienen. Das VRP ist eine Verallgemeinerung des Traveling Salesman Problems (TSP), welches versucht innerhalb eines gegebenen Graphen die kürzeste Tour zu finden, welche alle n Knoten besucht. Das VRP wurde erstmals von Dantzig und Ramser unter dem Namen Truck Dispatching Problem eingeführt. Das Ziel bestand darin, Routen für eine Flotte von Tanklastwagen zwischen einem Großterminal und einer großen Anzahl von vom Terminal belieferten Tankstellen zu planen.

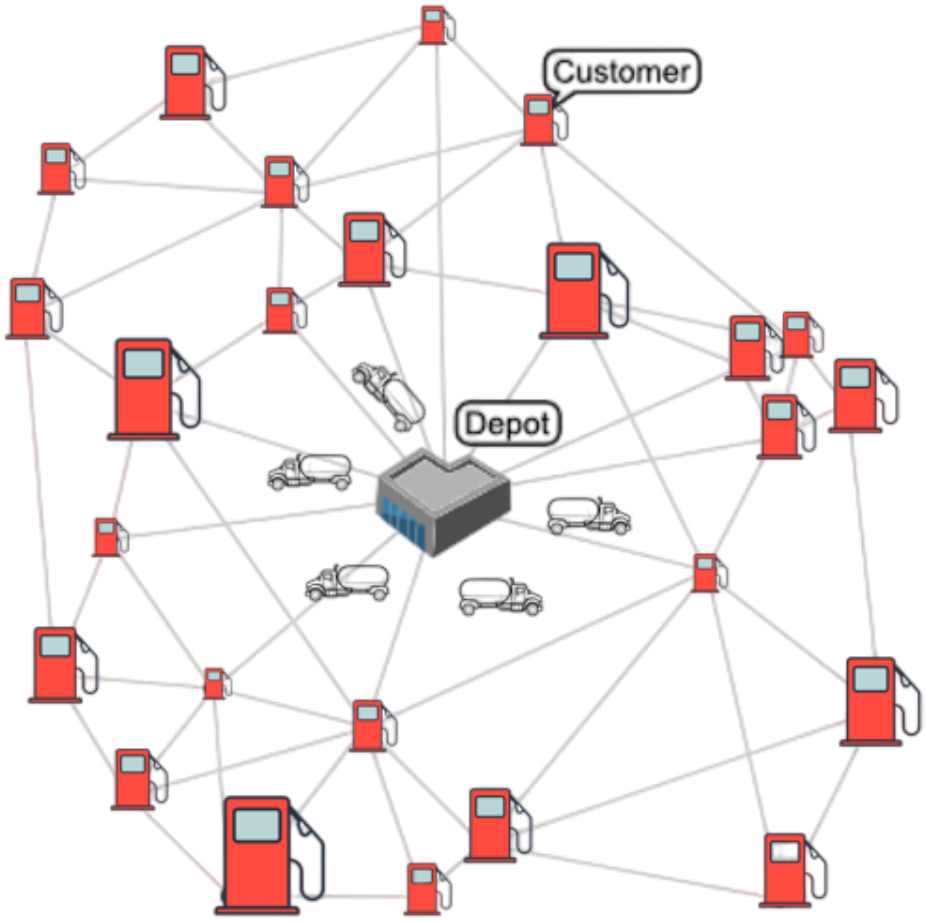

Abbildung 1 Veranschaulichung eines CVRP anhand von Tankstellen und LKWs [27][31][32]

Das Problem wurde erstmals wie folgt präsentiert: Wir haben eine homogene Flotte von Fahrzeugen, die sich zunächst am Terminal befinden und eine begrenzte Kapazität haben. Jede Tankstelle fordert eine bestimmte Menge an Benzin. Das Ziel ist es, LKWs den Routen zuzuweisen, so dass alle Anforderungen erfüllt sind und die Kapazität der Lastwagen nicht überschritten wird. Diese Problemkategorie wird als kapazitätsorientiertes Vehicle Routing Problem (CVRP) bezeichnet und ist die meistuntersuchte Variante des VRP [1]. Mathematisch kann man das CVRP folgend formulieren:

$$(1) \quad \min \sum_{k=1}^{m} \sum_{i \neq j} d_{ij} x_{ijk}$$

unter den Nebenbedingungen :

$$(2) \quad \sum_{i=1}^{n} q_i y_{ik} \leq C, \quad k = 1, \ldots, m$$

$$(3) \quad \sum_{k=1}^{m} y_{ik} = \begin{cases} m, & i = 0 \\ 1, & i = 1, \ldots, n \end{cases}$$

$$(4) \quad \sum_{j=0}^{n} x_{ijk} = y_{ik}, \quad i = 0, \ldots, n; \; k = 1, \ldots, m$$

$$(5) \quad \sum_{i=0}^{n} x_{ijk} = y_{jk}, \quad j = 0, \ldots, n; \; k = 1, \ldots, m$$

$$(6) \quad \sum_{i,j \in S} x_{ijk} \leq |S| - 1 \quad S \subset V; |S| \geq 1; \; k = 1, \ldots, m$$

$$(7) \quad x_{ijk} \in \{0,1\}, \quad i, j = 1, \ldots, n; \; k = 1, \ldots, m$$

$$(8) \quad y_{ik} \in \{0,1\}, \quad i = 1, \ldots, n; \; k = 1, \ldots, m$$

Abbildung 2 Standartproblem der Tourenplanung [4]

Hierbei ist n die Kundenanzahl, m die Fahrzeuganzahl, C die Kapazität eines Fahrzeugs, d_{ij} die Distanz zwischen den Standorten von Kunde i und Kunde j. Die Entscheidungsvariable x_{ijk} ist 1 wenn Fahrzeug k die Kante (i, j) befährt, sonst 0. y_{ik} = 1 wenn Fahrzeug k Knoten i bedient, sonst 0.

Die Zielfunktion (1) ermittelt die minimale Gesamtdistanz zwischen allen von den m Fahrzeugen befahrenen Kanten. Die Nebenbedingung (2) stellt sicher, dass die Kapazitätsgrenze jedes Fahrzeuges eingehalten wird. Nebenbedingung (3) sorgt dafür, dass jeder Kundenknoten nur einmal besucht wird und dass jedes Fahrzeug das Depot besucht. Nebenbedingungen (4) und (5) sorgen dafür, dass Kanten den Fahrzeugen zugeordnet werden, welche auch ihre inzidenten Knoten besuchen. Die Subtour Elemination (6) stellt sicher, dass x keine Subtouren enthält, was einfach gesagt bedeutet, dass die Tour jedes Fahrzeuges zusammenhängend sein muss. Nebenbedingungen (7) und (8) sichern die Ganzzahligkeit von x und y.

Es existieren noch einige weitere spezialisiertere Varianten des VRP wie z.B. das VRP mit Zeitfenstern, bei dem Kunden nur zu bestimmten Zeitfenstern anzutreffen sind, dass Multi-Depot VRP, bei dem es mehrere Ausgangsdepots gibt oder das heterogene VRP, bei dem die Lieferflotte aus verschiedenen Fahrzeugen mit entsprechend verschiedenen Einschränkungen besteht. Diese Arbeit beschränkt sich ausschließlich auf das kapazitative Vehicle Routing Problem (CVRP).

2.2 Anwendungsfälle

Das Vehicle Routing Problem (VRP) spielt in zahlreichen Anwendungsbereichen eine zentrale Rolle. Ein prominentes Beispiel ist die Lieferlogistik, bei der Unternehmen Routen für Lieferfahrzeuge planen, um Produkte effizient an Kunden zu liefern und dabei Zeit und Kosten zu minimieren. Ebenso wird das VRP in der Abfallentsorgung verwendet, um Müllsammelfahrzeuge optimal zu routen. Im Personentransport, sei es mit Bussen oder Shuttles, hilft das VRP, optimale Routen und Fahrpläne zu erstellen. Für Paketdienste wie Amazon ist das VRP unerlässlich, um die effizientesten Routen für die Zustellung von Paketen zu planen.

Darüber hinaus nutzen Unternehmen VRP-Ansätze für das Routing von Servicetechnikern, Schulbussen, Flottenmanagement sowie für Notfall- und Katastrophenreaktionen. Historisch gesehen wurden VRP-Modelle auch in der Milch- und Postauslieferung sowie in der Energieversorgung eingesetzt, um effiziente Routen für die Verteilung von Gütern oder die Wartung von Infrastruktureinrichtungen zu planen.

Außerdem lässt sich die hohe Relevanz des VRP an der Anzahl der diesbezüglichen wissenschaftlichen Veröffentlichungen (Abb. 3) erkennen.

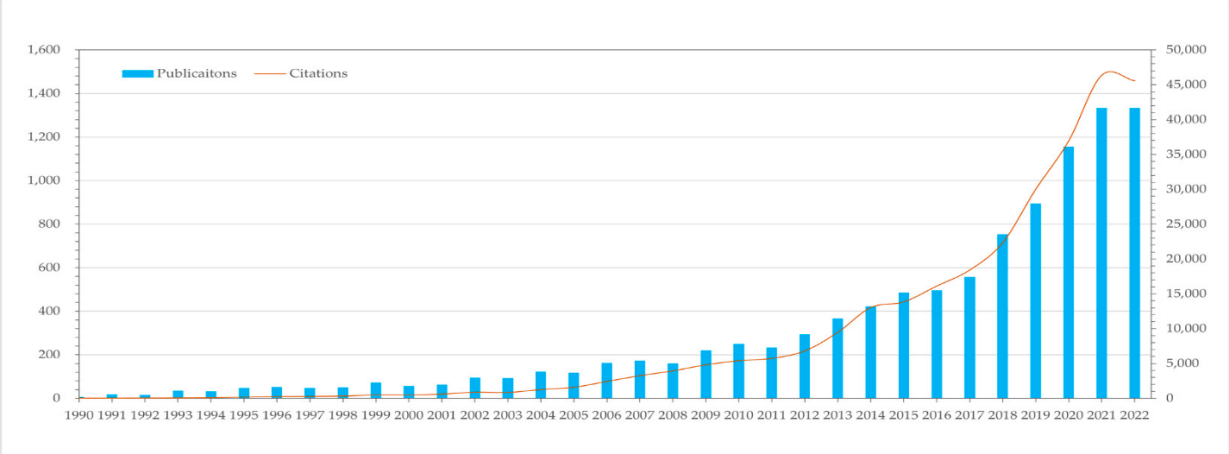

Abbildung 3 Anzahl Veröffentlichungen (Blau), Anzahl Zitierungen(gelb) jeweils pro Jahr; [15]

3 Heuristiken

3.1 Konstruktive Heuristiken

Bei konstruktiven Heuristiken handelt es sich um Pläne oder Abläufe, die durch schrittweise Ein- oder Anfügung von Operationen erzeugt werden.

3.1.1 Nearest Neighbour

Der Nearest Neighbour Algorithmus ist im allgemeinen Fall, dass man ausgehend von dem derzeitigen Knoten die minimalgewichtete Kante zu dem nächsten unbesuchten Knoten wählt und dies wiederholt bis alle Knoten einmal besucht wurden. Danach bildet man einen hamiltonischen Kreis indem man den Startknoten mit dem Endknoten verbindet. Da es allerdings im Allgemeinen nicht möglich ist einen hamiltonischen Kreis zu bilden ohne die Kapazitätsgrenze zu überschreiten kehrt das Fahrzeug zum Depot zurück sobald der nächste Schritt die Kapazitätsgrenze überschreiten würde [2]. Der Algorithmus kann wie folgend beschrieben werden:

1. Initialisiere Subtour S, die nur das Depot (v_0) enthält
2. Initialisiere Kapazitätsvariable c = 0
3. Finde den Knoten v_n mit minimalem Abstand zu dem hintersten Knoten aus der Subtour S
4. Prüfe durch c + $c_n \leq c_{max}$ ob Kapazitätsgrenze eingehalten wird, wenn die Kapazitätsgrenze eingehalten wird gehe zu Schritt 5, wenn die Kapazitätsgrenze überschritten wird füge S den Depotknoten v_0 an letzter Stelle hinzu und gehe danach zu Schritt 1
5. Füge der Subtour S den Knoten v_n an letzter Stelle hinzu, gehe zu Schritt 3

Da der Nearest Neighbour Algorithmus in jedem Schritt die derzeit beste Wahl trifft ist er ein sogenannter Greedy Algorithmus [3] und bietet demnach selten die optimale Lösung. Der Hauptvorteil des Nearest Neighbour Algorithmus lieg darin, dass er sehr simpel und somit Lauzeitsparend ist. Seine Laufzeit beträgt O(n^2). Außerdem wird er oft verwendet um die Startlösung für Metaheuristiken zu finden.

3.1.2 Einfügeverfahren

Die Grundidee der Einfügeverfahren besteht darin, dass man ausgehend von einer geschlossenen Subtour, welche aus einem Teil der Knoten des Graphens besteht (klassischerweise dem Startpunkt und dem Endpunkt) iterativ ausgehend von einem verfahrensabhängigen Kriterium dem Kreis einen unbesuchten Knoten hinzufügt. Dabei wird der neue Knoten immer so eingefügt, dass die durch seine Einfügung entstandene Mehrdistanz ($d_{ik} + d_{kj} - d_{ij}$) minimiert wird. Hierbei sind i, j die beiden Knoten, zwischen welchen der neue Knoten k eingefügt wird und d_{ab} bezeichnet die Gewichtung, welche in konkreten Fällen meistens die Distanz der Kante zwischen Knoten a und Knoten b darstellt. Hierbei unterscheidet man laut [4] zwischen der Cheapest-, Nearest-, Farthest-, und Random Insertion, welche folgend vorgestellt werden.

3.1.2.1 Cheapest Insertion

Bei der Cheapest Insertion wählt man den Knoten, der die betragliche Länge der Subtour am geringsten ansteigen lässt. Anschließend bestimmt man den Knoten mit der günstigsten Einfügungskante und fügt ihn an der optimalen Stelle in die Tour ein. Die hierbei erzeugte Tour bezeichnet man auch als INSERTC. Der Cheapest Insertion Algorithmus kann folgend beschrieben werden:

1. Initialisiere Tour, die nur das Depot v_0 enthält
2. Finde den unbesuchten Knoten v_k mit minimalem Abstand e_k= min(d_{0k}) zum Startdepot. Bilde anschließend die Subtour S = (v_0, v_k, v_0).
3. Initialisiere Kapazitätsvariable c = c_k
4. Finde für die Subtour S den unbesuchten Knoten v_n, dessen Einfügungskosten minimal sind.

5. Prüfe durch c + $c_n \leq c_{max}$ ob Kapazitätsgrenze eingehalten wird, wenn die Kapazitätsgrenze eingehalten wird gehe zu Schritt 6, wenn die Kapazitätsgrenze überschritten wird gehe zu Schritt 1
6. Finde die Kante für die e_n= min($d_{in} + d_{nj} - d_{ij}$) gilt und füge sie zwischen Knoten i und Knoten j in die Subtour S ein.
7. Setze c = c + c_n, gehe zu Schritt 4

Der laufzeitärmste Cheapest Insertion Algorithmus hat die Ordnung O($n^2 log_2(n)$) und ist im suboptimalsten Fall doppelt so lang wie die optimale Tour.

3.1.2.2 Nearest Insertion

Bei der Nearest Insertion wählt man den Knoten der die geringste summierte Distanz zu allen Teilknoten der Subtour hat und fügt ihn in die Tour ein. Die hierbei entstehende Tour wird als INSERTN bezeichnet. Der Nearest Insertion Algorithmus kann folgend für ein CVRP beschrieben werden:

1. Initialisiere Tour, die nur das Depot v_0 enthält
2. Finde den unbesuchten Knoten v_k mit minimalem Abstand e_k= <u>min</u>(d_{0k}) zum Startdepot. Bilde anschließend die Subtour S = (v_0, v_k, v_0).
3. Initialisiere Kapazitätsvariable c = c_k
4. Bilde die Distanzen zwischen allen unbesuchten Knoten und allen Knoten aus S und wähle den Knoten, welcher die <u>minimale</u> Distanz zu einem beliebigen Knoten aus S hat.
5. Prüfe durch c + $c_n \leq c_{max}$ ob Kapazitätsgrenze eingehalten wird, wenn die Kapazitätsgrenze eingehalten wird gehe zu Schritt 6, wenn die Kapazitätsgrenze überschritten wird gehe zu Schritt 1
6. Finde die Kante für die e_n= min($d_{in} + d_{nj} - d_{ij}$) gilt und füge sie zwischen Knoten i und Knoten j in die Subtour S ein.
7. Setze c = c + c_n, gehe zu Schritt 4

Der Nearest Insertion Algorithmus hat eine Laufzeit von O(n^2) und ist im suboptimalstem Fall doppelt so lang wie die optimale Tour [21].

3.1.2.3 Farthest Insertion

Bei der Farthest Insertion wählt man genau den Knoten, der die größte Summierte Distanz zu allen Teilknoten der Subtour hat. Die hierbei entstehende Tour wird auch als INSERTF bezeichnet. Der Farthest Insertion Algorithmus kann folgend beschrieben werden:

1. Initialisiere Tour, die nur das Depot v_0 enthält
2. Finde den unbesuchten Knoten v_k mit maximalem Abstand e_k= <u>max</u>(d_{0k}) zum Startdepot. Bilde anschließend die Subtour S = (v_0, v_k, v_0).
3. Initialisiere Kapazitätsvariable c = c_k
4. Bilde die Distanzen zwischen allen unbesuchten Knoten und allen Knoten aus S und wähle den Knoten, welcher die <u>maximale</u> Distanz zu einem beliebigen Knoten aus S hat.
5. Prüfe durch c + $c_n \leq c_{max}$ ob Kapazitätsgrenze eingehalten wird, wenn die Kapazitätsgrenze eingehalten wird gehe zu Schritt 6, wenn die Kapazitätsgrenze überschritten wird gehe zu Schritt 1
6. Finde die Kante für die e_n= min($d_{in} + d_{nj} - d_{ij}$) gilt und füge sie zwischen Knoten i und Knoten j in die Subtour S ein.
7. Setze c = c + c_n, gehe zu Schritt 4

Wie man sehen kann ist die Farthest Insertion sehr ähnlich zu der Nearest Insertion, die beiden Algorithmen unterscheiden sich ausschließlich an den <u>unterstrichenen</u> Stellen. Der Farthest Insertion

Algorithmus hat eine Laufzeit von O(n^2) und besitzt im suboptimalsten Fall die $2\ ln(n)$ + 0.16 fache Länge der optimalen Lösung [21].

3.1.2.4 Random Insertion

Wie der Name bereits suggeriert wird bei der Random Insertion ein zufälliger unbesuchter Knoten gewählt, welcher dann der Subtour zugefügt wird. Die hierbei entstandene Tour bezeichnet man auch als INSERTR. Der Random Insertion Algorithmus kann folgend beschrieben werden:

1. Initialisiere Tour, die nur das Depot v_0 enthält
2. Wähle einen zufälligen unbesuchten Knoten v_k
3. Initialisiere Kapazitätsvariable c = c_k
4. Bestimme einen zufällig bestimmten unbesuchten Knoten v_n
5. Prüfe durch c + $c_n \leq\ c_{max}$ ob Kapazitätsgrenze eingehalten wird, wenn die Kapazitätsgrenze eingehalten wird gehe zu Schritt 6, wenn die Kapazitätsgrenze überschritten wird gehe zu Schritt 1
6. Finde die Kante für die e_n= min($d_{in} + d_{nj} - d_{ij}$) gilt und füge sie zwischen Knoten i und Knoten j in die Subtour S ein.
7. Setze c = c + c_n, gehe zu Schritt 4

Der Random Insertion Algorithmus hat analog zu dem Farthest Insertion Algorithmus eine Laufzeit von O(n^2) und besitzt im suboptimalsten Fall die $2\ ln(n)$ + 0.16 fache Länge der optimalen Lösung [21].

Abgesehen von der Cheapest Insertion haben alle Einfügeverfahren die Laufzeit O(n^2), wobei man allerdings beachten sollte, dass die random insertion in der Praxis trotzdem ungefähr 2 mal schneller ist als die restlichen Insert-Verfahren, da keine Knotensuche stattfindet. Laut den Testergebnissen von [21], welche TSP-Probleme mit jeweils 50 euklidischen Knoten untersuchten stellte sich heraus, dass das Nearest Insertion Verfahren im Vergleich am schlechtesten abschnitt [4]. Die von ihm konstruierten Touren waren zwischen 7 und 22% Länger als die Pfade des performantesten Farthest Insertion Verfahrens. Allerdings sank in den Testergebnissen die Differenz der Tourlänge der beiden Verfahren bei einer steigenden Knotenanzahl, wobei bei 2000 Knoten ein Equilibrium erreicht wurde. Außerdem sollte man beachten, dass bei der Farthest Insertion bis zu $\frac{n^2}{4}$ Operationen benötigt um den nächsten Knoten zu finden. Auf Grund der Verfahrensindifferenz bei großen Problemen wählt man bei diesen präferiert das Random Insertion Verfahren, da es Operationen quadratischer Ordnung ersparen kann [4]. Allerdings sollte man hierbei beachten, dass das in [4] behandelte TSP sich nur mit einer einzigen zusammenhängenden Route befasst, was bei dem CRVP im Allgemeinen nicht der Fall ist. In Abschnitt 5.3 wird genauer auf diese Thematik eingegangen.

3.1.3 Savings

Das 1964 von Clarke und Wright entwickelte Savings-Verfahren ist bis heute das wohl bekannteste und am meisten eingesetzte Verfahren für knotenorientierte Tourenprobleme ([4] Seite 283). Allerdings funktioniert das Savings-Verfahren nur wenn eine Symmetrische Entfernungsmatrix vorliegt. Das Verfahren beginnt mit einer Anfangslösung, bei der jeder Kunde von einem einzelnen Fahrzeug bedient wird. Hierbei ergibt sich als Gesamtentfernung Z für die Startlösung Z = $2\sum_{i=0}^{n} d_{0i}$, wobei d_{0i} die Entfernung vom Knoten i von Depot 0 und n die Gesamtanzahl der Kunden sei. Nun wird diese meist ungünstige Startlösung sukzessive unter Beachtung der Kapazitätsrestriktionen verbessert. Dies geschieht, indem man 2 Touren so verknüpft, dass die neue Tour jeweils einen Endkunden aus jeden der vorherigen Touren als Endkunde besitzt, wobei ein Endkunde entweder der erste oder letzte angefahrene Kunde einer Tour ist. Ein Beispiel für einen Schritt des Savings-

Algorithmus findet man in der folgenden Abbildung.

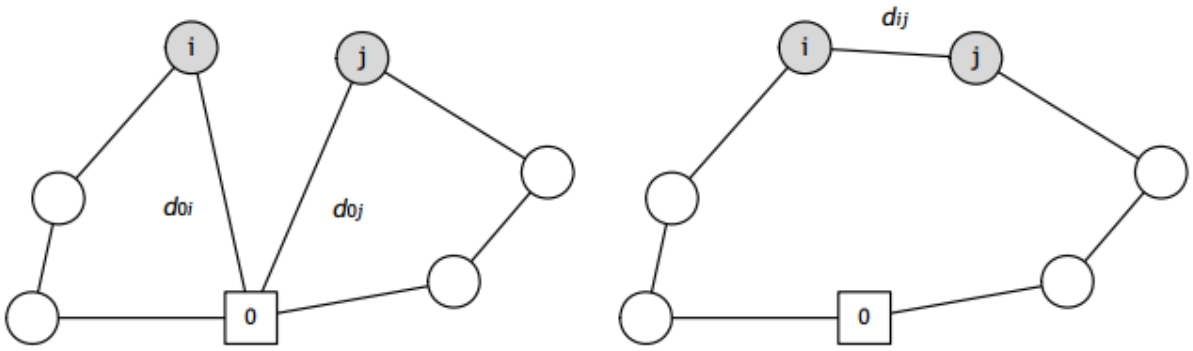

Abbildung 4 Tour vor Anwendung von Savings-Schritt(links), Tour Nach Anwendung von Savings-Schritt (rechts); [4] Seite 288

Durch dieses Vorgehen spart man eine Strecke von $s_{ij} = d_{0i} + d_{0j} - d_{ij}$. Falls die Entfernungen d_{ij} mit der Koordinatenmethode gewonnen wurden gilt die Dreiecksungleichung $d_{ik} \leq d_{ij} + d_{ik}$ für alle Knoten i, j und k ([4] Seite 284). Der Savingswert ist größer, je näher Kunden i und j nebeneinander liegen und je weiter sie von dem Depot entfernt sind.

Algorithmus ([4] Seite 278):

Initialisierung

1. Initialisiere eine Tourenliste mit den n Pendeltouren.

Verarbeitung

2. Berechne für je zwei Endkunden i, j die $n^2 - n$ Savingswerte s_{ij} und lege sie in einem Array ab i, j = 1...n.
3. Sortiere die berechneten Savingswerte im Array absteigend.
4. Betrachte die positiven Savingswerte s_{ij} beginnend mit dem größten Savingswert in absteigender Reihenfolge und überprüfe, ob unter Beachtung der Restriktionen jeweils die zu den Endkunden i und j gehörenden Touren verknüpft werden können.
5. Können in 4 zwei Touren verknüpft werden ohne das Kapazitätslimit zu überschreiten, führe dies aus.
6. Streiche in der Tourenliste die beiden Touren und nehme die neue Tour in den Tourenplan auf.
7. Überprüfe in 4 den nächst kleineren Savingswert.

Terminierung

8. Das Verfahren bricht ab, wenn alle positiven Savingswerte überprüft sind

Der Savings-Algorithmus hat auf Grund der Sortierung in Schritt 3 einen Rechenaufwand von $O(n^3 log_2(n))$ [22]. Allerdings kann man den Rechenaufwand reduzieren, indem man nur eine Teilmenge der Knoten berücksichtigt, hierbei sind die Knoten die weiter vom Depot entfernt sind interessanter, da sie wie vorherig festgestellt größere Savingswerte versprechen.

3.1.4 Sweep

Der 1974 von Gillet und Miller entwickelte Sweep-Algorithmus ist ein sogenanntes Koordinatenorientiertes Verfahren, was bedeutet, dass die Knoten in Polar- oder kartesischen

Koordinaten vorliegen. Hierbei wird das Koordinatensystem vor dem Beginn des Algorithmus so Transformiert, dass das Depot im Ursprung liegt. Die Entfernung eines Knoten zum Depot wird hierbei Euklidisch (also per Luftlinie) ermittelt.

Algorithmus ([4] Seite 278):

Initialisierung

1. Sortierung und Nummerierung der Kunden nach aufsteigendem Polarwinkel.

Verarbeitung

2. FOR i := 1 To n DO
    a. Erstelle bei Kunde i beginnenden Tourenplan der die Kundenliste wie folgend abarbeitet:
    b. REPEAT
        i. Eröffne neue Tour
        ii. Füge der Tour solange Kunden hinzu bis die Kapazitätsgrenze/Zeitbeschränkung o.ä. erreicht ist
        iii. Bestimme für diese Kundengruppe die TSP-Tour und deren Länge
    c. UNTIL Alle Kunden sind einer Tour zugeordnet
    d. Ermittle Gesamtlänge des Tourplans P_i
3. ENDFOR;
4. Ermittle den kürzesten Tourplan P = $min_i \{P_i\}$

Einfach gesagt bildet der Sweep-Algorithmus Gruppen aus Kunden die in sich in einer ähnlichen Richtung befinden und findet für jede dieser Kundengruppen mit Hilfe einer TSP-Heuristik wie Nearest Neighbour oder Insert eine praktische Route, die ein Flottenfahrzeug bewältigen kann. Ein Beispiel für die von einem Sweep Algorithmus gebildeten Tourenpläne findet man in Abbildung 5.

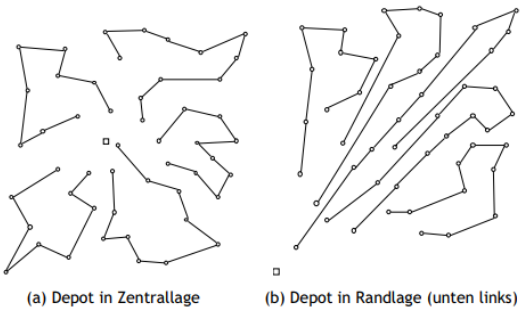

Abbildung 5 Sweep Algorithmus mit Depot in Zentrallage (links); Sweep Algorithmus mit Depot in Randlage (rechts); [5]

Wie man bereits anhand von Abb. 5 a) erahnen kann liefert der Sweep Algorithmus gute Ergebnisse, wenn das Depot recht zentral innerhalb der Kundenstruktur liegt und relativ viele Kunden pro Tour angefahren werden können. Ist dies allerdings wie in Abb. 5 b) nicht der Fall kann man bereits anhand der Abbildung erahnen, dass es sehr wahrscheinlich bessere Heuristiken für die Tourenplanung gibt.

3.2 Verbesserungsheuristiken

Verbesserungsheuristiken nehmen eine existierende Lösung für ein VRP und versuchen für diese eine naheliegende und objektive Verbesserung zu finden. Da Heuristiken dieser Klasse schnell gegen lokale Optima konvergieren werden sie häufig benutzt, um umfangreiche Probleme zu lösen.

3.2.1 Intra-Route

Die Intra-Route Heuristik untersucht die Nachbarschaft innerhalb einer einzigen Tour. Hierbei gibt es einige unterschiedliche Verfahren, wobei ich nachfolgend die laut [6] 5 relevantesten vorstellen werde.

I) Das ***Relocate*** Verfahren nimmt einen Knoten aus einer Route und ändert, wenn möglich seine Position in der Anfahrreihenfolge so, dass die Gesamtdistanz der Route maximal reduziert wird. Die Komplexität dieser Operation beträgt O(n^2) [6].

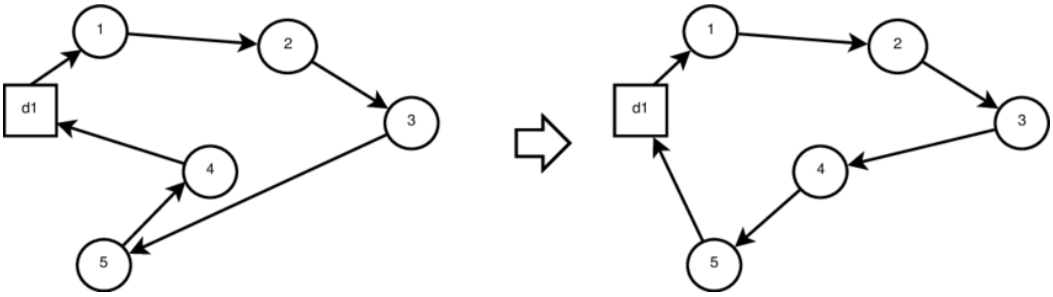

Abbildung 6 Route vor Relocate (links); Route nach Relocate(rechts) [28]

II) Das ***Exchange*** Verfahren nimmt eine Route und tauscht die Positionen von 2 in ihr enthaltenen Knoten. Hierbei wird die beste Lösung ebenfalls innerhalb einer Laufzeit von O(n^2) gefunden [6].

Abbildung 7. Route vor Exchange (links); Route nach Exchange (rechts) [29]

III) Das ***λ-opt*** Verfahren nimmt eine ganzzahlige Anzahl λ von Kanten und ändert deren Anknüpfungspunkte. Hierbei wird eine Laufzeit von O(n^{λ}) benötigt, um die optimale Lösung zu finden [6].

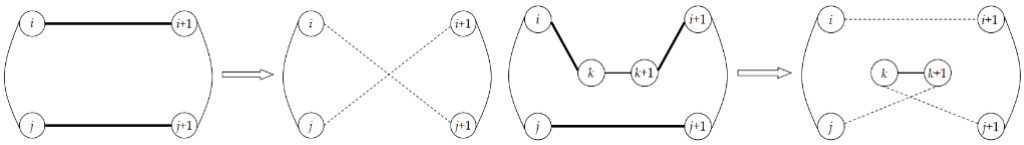

Abbildung 8 2-Opt (links); 3-Opt (rechts), dick gezeichnete Kanten werde bei dem jeweiligen Verfahren durch die gestrichelten Kanten ersetzt. [23]

IV) Das von Or im Jahr 1976 entwickelte ***Or-Opt*** Verfahren selektiert eine Knotenfolge mit vorgegebener Länge und ändert ihre Position innerhalb der Tour. Obwohl das **Or-Opt** ähnliche Ergebnisse wie das 3-Opt Verfahren liefert beträgt die Komplexität von **Or-Opt** nur O(n^2) [6].

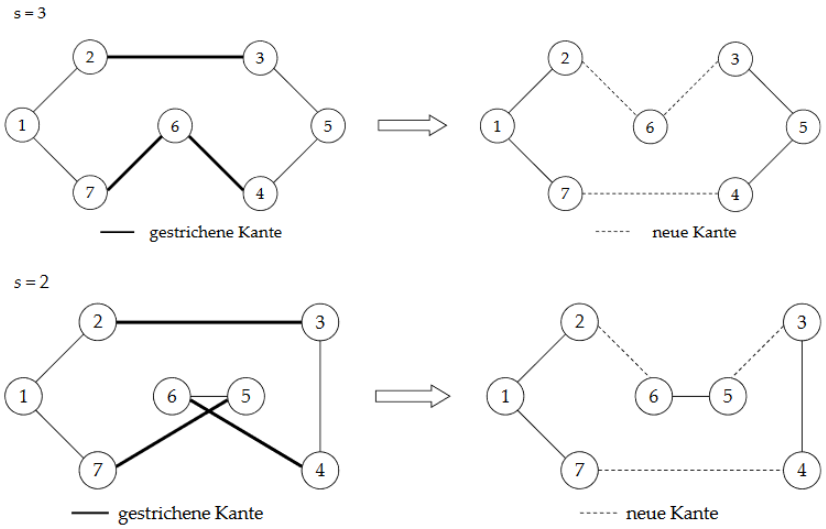

Abbildung 9 Or-Opt Verfahren, die dick gezeichnete Kanten werden bei dem Verfahren durch die gestrichelten Kanten ersetzt [23]

V) Das von Gendreau et al. (1992) entwickelte Verbesserungsverfahren basiert auf einer verallgemeinerten Einfügemethode, bekannt als Generalized Insertion Procedure (***GENI***). Im ersten Schritt, dem sogenannten Unstringing-Schritt, wird ein Knoten aus einer Rundreise herausgenommen. Anschließend erfolgt der Stringing-Schritt, bei dem der Knoten mithilfe eines 3-opt- oder 4-opt-Verfahrens wieder eingefügt wird. In Abbildung 7 wird gezeigt, wie der Knoten 8 durch einen 3-opt-Tausch wieder in die Rundreise integriert wird. Um den Aufwand zu minimieren und somit die Nachbarschaftsgröße zu verringern, werden nur Einfüge- und Herauslöseoptionen untersucht, bei denen „nahe gelegene" Knoten beteiligt sind. Numerische Tests haben ergeben, dass dieses Verfahren bessere Ergebnisse erzielt als das **Or-Opt**-Verfahren [23]. Hierbei hat jeder Insert eine Komplexität von O($np^4 + n^2$) [6].

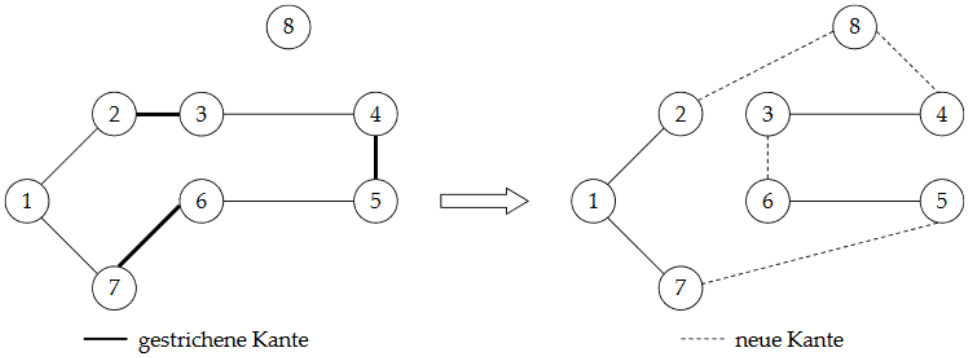

Abbildung 10 GENI-Verfahren, Graph vor einfügen von Knoten 8 (links), Graph nach einfügen von Knoten 8 mittels GENI (rechts) [23]

3.2.2 Inter-Route

Im Gegensatz zur Intra-Route Heuristiken untersuchten die Inter-Route Heuristiken nicht nur die Nachbarschaft innerhalb einer Tour, sondern sie beziehen auch naheliegende Touren mit ein. Viele Inter-Route Heuristiken sind Verallgemeinerungen von einer Intra-Route Heuristik. Die relevantesten Intra-Route Heuristiken lauten wie folgend:

I) Das ***2-opt**** Verfahren wählt 2 Touren und ändert die Endknoten von jeweils 2 Kanten. Hierbei werden die Endknoten so gewählt, dass die Kostenreduktion maximal ist. Wenn die Kosten für die Wahl der beiden Touren vernachlässigbar sind ist dieses Verfahren wie das auf eine Tour beschränkte ***2-opt*** Verfahren von Ordnung O(n^2). Wie der Name des ***2-opt**** Verfahrens bereits vermuten lässt ist dieses eine Version des ***2-opt*** Verfahrens, welche mehrere Routen betrachten kann. [7]

II) Das ***Insert*** Verfahren (nicht zu verwechseln mit der konstruktiven Insert Heuristik) wählt einen Knoten und fügt diesen an einer anderen Position ein. Wie bei dem auf eine Tour beschränktem ***Relocate*** Verfahren beträgt hierbei die Laufzeit O (n^2) [6].

III) Das sogenannte ***Swap*** Verfahren wählt 2 Knoten und tauscht, wie der Verfahrensname bereits suggeriert deren Positionen. Ein Schritt hat hierbei eine Laufzeit von O(n^2).

IV) Das in [7] vorgeschlagene ***Swap**** Verfahren ist eine erweiterte Version des ***Swap*** Verfahrens bei den nicht direkt die Konten vertauscht werden, sondern die beiden Knoten in jede Position der Tour des anderen Knotens eingefügt werden können. Wenn man dieses Verfahren naiv implementieren würde hätte es eine Laufzeit von O(n^3), aber man kann zeigen, dass man innerhalb der 3 initial kostengünstigsten Inserts garantiert den lokal optimalen Insert findet, womit man die Laufzeit auf O(n^2) reduzieren kann [7].

V) Das ***CROSS*** Verfahren vertauscht zwei Knotenfolgen, wobei auch eine der Folgen leer sein kann. Es handelt sich hierbei um eine Verallgemeinerung des 2-opt*, Insert und Swap Verfahrens. ***CROSS*** hat eine Laufzeit von O(n^4), wobei diese auf O($\lambda^2 n^2$) reduziert werden kann, indem man die Folgenlänge auf ein beliebiges λ beschränkt [6].

VI) ***λ – Interchange*** ist eine weitere Verallgemeinerung des ***CROSS*** Verfahrens, welches erlaubt jede Knotenmenge mit weniger als λ Knoten zwischen 2 Routen zu vertauschen. Hierbei kann diese Menge aus Knoten bestehen, die nicht aufeinander folgen, leer sind oder während des Wiedereinfügens ihre Richtung ändern. [7]

Die bis jetzt genannten Heuristiken tauschen einfach gesagt Laufzeit gegen Lösungsqualität. Hierbei sollte man allerdings auch beachten, dass das Optimum eines lokalen Verfahrens allgemein **nicht** dem Optimum eines anderen lokalen Verfahrens entspricht.

4 Metaheuristiken

Metaheuristiken sind per Definition problemunabhängig und werden als effektive Lösung für viele schwierige Optimierungsprobleme anerkannt [6]. Hierbei unterscheidet man zwischen nachbarschaftsbasierten Lösungen, welche iterativ die Nachbarschaft von einer existierenden Lösung untersucht und evolutionären Algorithmen, welche stärkere Lösungskandidaten sich verstärkt vermehren lässt, während schwächere Lösungszweige aussterben. Eine nachbarschaftsbasierte Metaheuristik unterscheidet sich von einer Verbesserungsheuristik, indem sie unter Umständen lokal suboptimale Lösungen nimmt, um die Konvergenz gegen lokale Optima zu vermeiden. In dieser Arbeit wird ausschließlich auf die Metaheuristik Large Neighbourhood Search [16] eingegangen.

4.1 Large Neighbourhood Search

Die von Shaw [17] entwickelte **Large Neighbourhood Search** (LNS) wird implizit durch die sogenannten *destroy* und *repair* Methoden definiert. Die *destroy* Methode zerstört einen Teil der derzeitigen Lösung, während die *repair* Methode die zerstörte Lösung repariert. Typischerweise enthält die *destroy* Methode stochastische Elemente, so dass bei jedem Aufruf verschiedene Elemente zerstört werden. Man definiert die Nachbarschaft N(x) einer Lösung als genau die Menge an Konstellationen, welche durch die sequenzielle Anwendung einer *destroy* und *repair* Methode erreicht werden können [17]. Ein Beispiel für eine *destroy* Methode wäre das zufällige entfernen von 10% aller Knoten, während man für *repair* z.B. die **Nearest-Neighbour** Heuristik verwenden könnte. Eine Algorithmische Darstellung der **large Neighbourhood Search** lautet wie folgt:

Algorithm 1 Large neighborhood search

```
1: input: a feasible solution x
2: x^b = x;
3: repeat
4:     x^t = r(d(x));
5:     if accept(x^t, x) then
6:         x = x^t;
7:     end if
8:     if c(x^t) < c(x^b) then
9:         x^b = x^t;
10:    end if
11: until stop criterion is met
12: return x^b
```

Abbildung 11 [16] Seite 10 large Neighbourhood search Algorithmus

Hierbei stellt x^b die derzeit beste Lösung, x eine mögliche Startlösung, r(.) die *repair* Methode, d(.) die *destroy* Methode, x^t die temporäre Lösung, und c(x) die Methode, welche den summierten Distanzwert einer Lösung x ausgibt dar. Wenn eine temporäre Lösung besser als die bisherig beste Lösung ist wird sie die neue beste Lösung. Es gibt auch laut [16] modernere Ansätze für das updaten der besten Lösung, welche sich unter anderem bei der Metaheuristik Simulated Annealing bedienen, aber diese zu erklären würde den Rahmen dieser Arbeit sprengen.

5 Forschung

Wie Abb. 1 bereits zeigt sind Heuristiken zur Lösung des VRP stand aktiver Forschung. Laut [6] lauten die 3 größten Forschungsthemen wie folgt:

1. Einheitliche Heuristik: Da für jede VRP Problemklasse eigene Lösungsheuristiken benötigt ergab eine Umfrage aus [6], dass es einen klaren Konsens dafür gibt, dass die Notwendigkeit für einen einheitlichen Algorithmus besteht, der VRPs mehrerer oder bestenfalls jeder Problemklasse lösen kann. Da besonders die Industrie Interesse an einem einheitlichen Solver für VRPs hat sind einheitliche oder mehrattributige Heuristiken momentan angesagte Forschungsthemen.
2. Automatisierte Erstellung von Heuristiken: Trotz der Beliebtheit von VRPs verbraucht das Erstellen einer effizienten Heuristik eine Menge Zeit und benötigt tiefgreifende Expertenkenntnisse. Auf Grund dieser Einschränkungen besteht ein großes Interesse an der Automatisierung der Erstellung von Heuristiken. Hierbei unterscheidet man zwischen automatisiertem Hyperparametertuning, welches versucht die Hyperparameter einer Heuristik automatisiert zu finden, automatisierter Algorithmusauswahl, welche versucht automatisiert den besten Algorithmus aus einer vorgegebenen Auswahl zu finden und automatisierter Algorithmuskomposition, welche versucht automatisiert einen Algorithmus aus bekannten Komponenten zu erstellen.
3. Machine Learning: Auf Grund der Vielseitigkeit von Machine Learning Algorithmen und deren Erfolg in der Anwendung bei anderen kombinatorischen Optimierungsproblemen wird seit 2 Jahrzehnten versucht Machine Learning auf VRPs anzuwenden. Auf diese Thematik wird in dem folgenden Abschnitt genauer eingegangen.

5.1 Machine und Deep Learning

Auf Grund des rasanten Fortschrittes von Machine und Deep Learning Techniken wird vermehrt versucht diese auf das VRP anzuwenden. Hierbei unterscheidet man zwischen dem iterativen Ansatz,

welcher ähnlich wie eine Heuristik iterativ eine bereits existierende Lösung verbessert und dem Ende zu Ende Ansatz, welcher ein Modell benutzt, um direkt aus einem gegebenen Problem eine Lösung zu generieren.

Iterative Ansätze können zwar in den meisten Fällen gute Lösungen finden, sind aber auf Grund ihres größeren Suchraumes teurer als Ende zu Ende Ansätze [10]. Da kleinere CVRP mit überschaubaren Kosten (5h für ein CVRP50 [10]) exakt gelöst werden können beschränkt sich diese Arbeit auf CVRP mit einer Mindestgröße von 100. Hierbei ist die Größe die Anzahl an Kunden, welche innerhalb eines isolierten Problems besucht werden müssen. Die laut [10] performanteste Methode für CVRP100 ist die **Neural Large Neighbourhood Search** [11], welche einen speziell für die **large Neighbourhood Search** [16] entwickelten Attention-Mechanismus [14] verwendet. Hierbei wird bei jedem Schritt zufällig entweder der punktbasierte Destroyoperator, welcher die n naheliegendsten Kunden eines zufällig ausgewählten Kunden entfernt oder der tourbasierte Destroyoperator, welcher die n naheliegendsten Routen eines zufällig ausgewählten Kunden zerstört gewählt. Anschließend setzt das Deep Learning Modell die Routen wieder zusammen.

	Instance Characteristics					Avg. Costs				Avg. Time (s)			
Instance Set	# Cust.	Dep.	Cust. Pos.	Dem.	Q	NLNS	LNS	LKH3	UHGS	NLNS	LNS	LKH3	UHGS
XE_1	100	R	RC(7)	1-100	206	30243.3 (**0.32%**)	30414.0 (0.89%)	30785.0 (2.12%)	30146.4	191	192	372	36
XE_2	124	R	C(5)	Q	188	36577.3 (**0.55%**)	36905.8 (1.45%)	37226.9 (2.33%)	36378.3	191	192	444	64
XE_3	128	E	RC(8)	1-10	39	33584.6 (**0.44%**)	34122.5 (2.05%)	33620.0 (0.54%)	33437.8	190	192	122	74
XE_4	161	C	RC(8)	50-100	1174	13977.5 (**0.72%**)	15002.5 (8.11%)	13984.9 (0.78%)	13877.2	191	194	32	54
XE_5	180	R	C(6)	U	8	26716.1 (0.58%)	27246.4 (2.58%)	26604.4 (**0.16%**)	26562.4	191	193	65	86
XE_6	185	R	R	50-100	974	21700.0 (**1.09%**)	24071.2 (12.14%)	21705.2 (1.15%)	21465.7	191	195	100	101
XE_7	199	R	C(8)	Q	402	29202.4 (2.03%)	30273.1 (5.78%)	28824.9 (**0.72%**)	28620.0	191	195	215	142
XE_8	203	C	RC(6)	50-100	836	20593.0 (**0.51%**)	21038.2 (2.68%)	20768.6 (1.37%)	20488.8	612	618	123	103
XE_9	213	C	C(4)	1-100	944	12218.7 (2.26%)	12828.4 (7.37%)	12078.6 (**1.09%**)	11948.2	613	624	66	145
XE_10	218	E	R	U	3	117642.1 (**0.04%**)	119750.8 (1.83%)	117699.8 (0.08%)	117600.1	612	616	112	138
XE_11	236	E	R	U	18	27694.7 (**0.65%**)	30132.4 (9.50%)	27731.5 (0.78%)	27517.0	613	621	67	190
XE_12	241	E	R	1-10	28	79413.6 (**3.10%**)	80383.3 (4.36%)	79626.8 (3.38%)	77023.7	614	618	266	257
XE_13	269	C	RC(5)	50-100	585	34272.3 (**0.82%**)	35256.8 (3.71%)	34521.5 (1.55%)	33994.7	613	618	343	214
XE_14	274	R	C(3)	U	10	29032.5 (1.60%)	29168.0 (2.08%)	28646.3 (**0.25%**)	28573.9	614	620	77	320
XE_15	279	E	R	SL	192	45440.9 (1.81%)	47144.9 (5.63%)	45224.6 (**1.32%**)	44633.5	615	629	347	329
XE_16	293	C	R	1-100	285	49259.3 (**0.57%**)	51080.8 (4.29%)	50415.9 (2.93%)	48981.4	614	624	560	251
XE_17	297	R	R	1-100	55	37096.3 (1.41%)	39324.0 (7.50%)	37031.9 (**1.23%**)	36582.1	615	623	152	250

Abbildung 12 Methodikvergleich aus [11]; NLNS steht für Neural Large Neighbourhood Search[11], LNS steht für Large neighbourhood Search[16], LKH3 steht für Lin-Kerhinghan-Helsaun Solver [26] Version 3, UHGS steht für Unified Hybrid Genetic Search [25]

Wie man in Abbildung 12 sehen kann ist momentan noch die **Unified Hybrid Genetic Search** [25] der **NLNS** [11] voraus. Trotzdem zeigt die **NLNS**, dass der geschickte Einsatz von Deep Learning Methoden das Potential hat Metaheuristiken zu verbessern, da die **NLNS** Probleme schneller und kostenärmer löst als die pure Metaheuristik **NLS**, auf der sie basiert.

6 Vergleich der konstruktiven Heuristiken anhand verschiedener Benchmarks

Da nun eine Vielzahl an Heuristischen Verfahren für die Lösung von CVRP vorgestellt wurden stellt sich nun die Frage, wie effektiv die behandelten Verfahren in Hinsicht auf Lösungsoptimalität und Rechenzeit sind. Um dies zu überprüfen werden sogenannte Benchmarks benutzt. Eine Benchmark ist in diesem konkreten Fall ein CVRP mit öffentlich bekannter Lösung. Außerdem beschränken sich die Tests nur auf Probleme, welche 100 oder mehr Kunden beinhalten, da kleinere Probleme wie in 4.1 beschrieben in den meisten Fällen nicht praxisrelevant sind, da sie in vertretbarer Zeit exakt lösbar sind. Die Benchmarkdateien sind konsistent nach dem Schema X-n{Anzahl Knoten}-k{Anzahl minimal benötigter Fahrzeuge} benannt. Demnach kann man aus dem Dateinamen ‚X-n101-k10‘ ermitteln, dass die Benchmarkdatei 101 Knotenpunkte enthält, während man mindestens 10 Fahrzeuge benötigt um alle Kunden beliefern zu können.

6.1 Benchmarkstruktur

6.1.1 Instanzen

Die innerhalb dieser Arbeit verwendeten Benchmarks stammen alle aus [19], welche eine Sammlung von verschiedenen Benchmarks beinhaltet. Innerhalb dieser Sektion wird eine Auswahl an Benchmarks aus [20] benutzt, um die Effektivität der vorgestellten Heuristiken zu überprüfen. Im Allgemeinen werden Benchmark-Instanzen wie folgend dargestellt:

```
NAME : X-n101-k25
COMMENT :       "Generated by Uchoa, Pecin, Pessoa, Poggi, Subramanian, and
Vidal (2013)"
TYPE : CVRP
DIMENSION :     101
EDGE_WEIGHT_TYPE :      EUC_2D
CAPACITY :      206
NODE_COORD_SECTION
1       365     689
2       146     180
…
100     954     950
101     615     750
DEMAND_SECTION
1       0
2       38
…
100     78
101     35
DEPOT_SECTION
        1
        -1
EOF
```

Hierbei wird zuerst der Name des Problems, gefolgt von einem Kommentar der Verfasser deklariert. Anschließend werden Problemtyp und Problemdimension genannt. Darauffolgend wird der Knotengewichtstyp deklariert und die Fahrzeugkapazität festgelegt. Allgemeint gibt es hier 2 Arten von Knotengewichtstypen: ‚EUC_2D' legt das Gewicht zwischen 2 Knoten auf deren euklidische Distanz fest, wobei jeder Knoten von jedem beliebigen anderen Knoten aus erreicht werden kann. Der Gewichtstyp ‚EXPLICIT' definiert eine Gewichtsmatrix, welche im Allgemeinen nicht quadratisch ist und somit bei diesem Gewichtstyp nicht garantiert ist, dass jeder Knoten jeden anderen Knoten direkt erreichen kann. Da innerhalb dieser Arbeit nur ‚EUC_2D' Gewichtstypen behandelt werden wird an dieser Stelle bewusst auf die Erklärung des ‚EXPLICIT' Gewichtstyps verzichtet. Innerhalb der ‚NODE_COORD_SECTION' wird ein Knoten mitsamt seiner X und Y Koordinate deklariert, in dem oberen Beispiel wäre Knoten 1 an Position (365, 689). Innerhalb der ‚DEMAND_SECTION' wird die Nachfrage der einzelnen Knoten deklariert, hierbei hat Knoten 1 eine Nachfrage von 0 und Knoten 2 eine Nachfrage von 38. Den Depotknoten kann man daran erkennen, dass seine Nachfrage auf 0 gesetzt ist. Demnach befindet sie das Depot des obigen Beispiels bei Knoten 1. Alle Instanzdateien werden durch die Endung ‚.vrp' gekennzeichnet.

6.1.2 Lösungen

Um die Ergebnisse eines Algorithmus angemessen bewerten zu können werden innerhalb dieser Arbeit nur Benchmarks mit bekannten Lösungen verwendet. Die Struktur einer Lösungsdatei ist wie folgend gegeben:

```
Route #1: 31 46 35
Route #2: 15 22 41 20
…
```

```
Route #25: 75 93
Route #26: 24 95 73 53 33 32
Cost 27591
```

Wobei jede Route in sequentieller Abfolge aus nicht Depotknoten angegeben ist. Die Kosten werden berechnet, indem man die ganzzahlig gerundete euklidische Distanz zwischen allen Knoten innerhalb einer Route über alle Routen summiert. Die ganzzahlige Rundung von Distanzen in Benchmarks ist in der CVRP-Community ein kontroverses Thema, da einige Parteien behaupten, dass die Rundung ganzzahlig rechnende Algorithmen bevorzugt. Allerdings sei das Problem, wenn man nicht Rundet, dass verschiedene Systeme oder Programme verschiedene Präzision besitzen, was zu unverhältnismäßigen Vergleichen führen kann. Besonders Problematisch sei die Rundung von Benchmarks in Fällen, bei denen Methoden so performant sind, dass Differenzen nur im Nachkommabereich auftreten. Mehr dazu findet man in [20]. Alle Lösungsdateien werden durch die Endung ‚.sol' gekennzeichnet.

6.2 Aufbau

Um faire Vergleiche zu ermöglichen werden die konstruktiven Heuristiken auf einer Teilmenge der Benchmarks aus [20], welche per Hand ausgesucht wurden um die Fälle [Depot in Randlage mit zufällig verteilten Kunden (X-n1001-k43), Depot in Randlage mit geclusterten Kunden (X-n979-k58), Depot in Zentrallage mit zufällig verteilten Kunden (X-n936-k58) und Depot in Zentrallage mit geclusterten Kunden (X-n819-k171)] zu repräsentieren getestet. Zusätzlich dazu wurde ein Testlauf auf einer größeren Benchmark namens Antwerp1 aus [30] getätigt, um zu zeigen, wie sich die betrachteten Eröffnungsheuristiken bei sehr großen Problemen verhalten. Die Lösungszeit wird in Sekunden angegeben, die Distanz zwischen 2 Punkten wird vor der Summierung der Gesamtkosten ganzzahlig gerundet. Die Algorithmen werden anhand ihrer Lösungszeit und ihrer Kosten im Vergleich zur optimalen Lösung bewertet. Die Benchmarktests werden alle auf einem AMD Ryzen™ 9 5950 X mit 16 Cores und 32 Thread Prozessoren ausgeführt.

6.3 Ergebnisse

6.3.1 Depot in Randlage mit zufällig verteilten Kunden (X-n1001-k43)

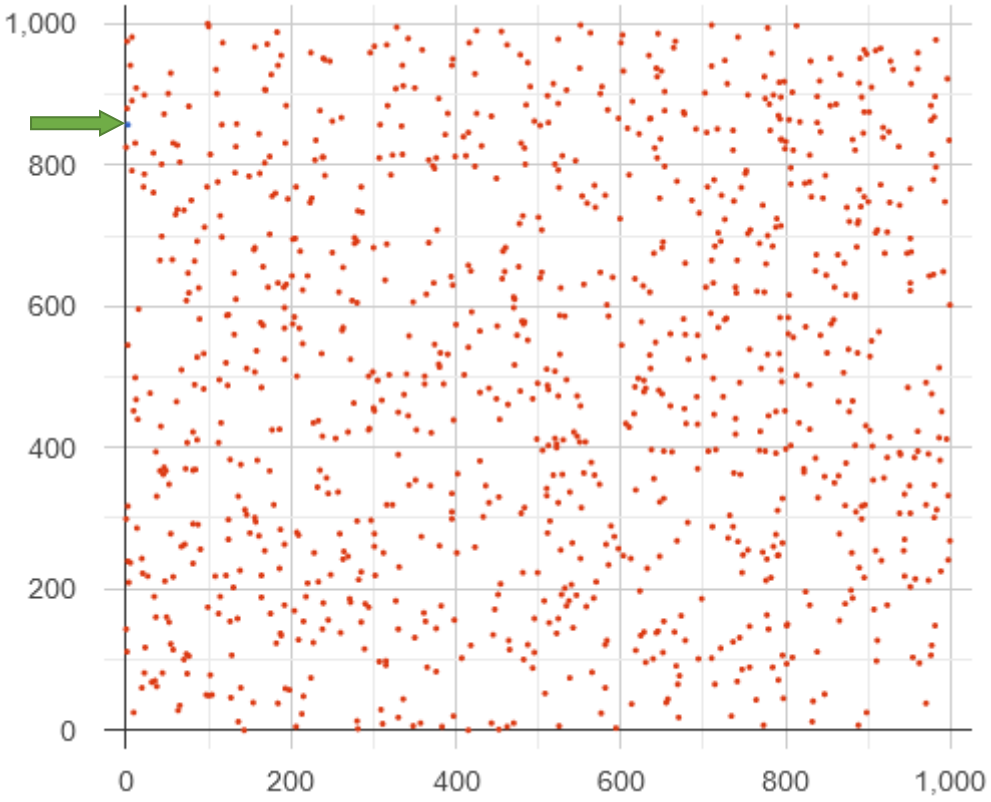

Abbildung 13: Plot der Benchmark X-n1001-k43, das Depot(blau) wird oben links von dem grünen Pfeil markiert, die Kunden (rot) sind zufällig verteilt; Quelle: [19]

In dieser Untersuchung analysieren wir das in Abbildung 13 gezeigte Szenario, bei dem sich das Depot am Rand eines definierten Gebiets befindet und die Kunden zufällig innerhalb dieses Gebiets verteilt sind. Solche Szenarien sind typisch für die Verteilung von Gütern aus Industriegebieten und urbanen Lagern, welche üblicherweise am Stadtrand angesiedelt sind.

Algorithmus	Lösungszeit	Routenkosten	Relative Differenz zur optimalen Lösung
cheapest Insert	1.817034	87133.0	20.42%
nearest Insert	2.366258	84265.0	16.46%
farthest Insert	2.358029	105938.0	46.41%
random Insert	0.007203	189298.0	161.62%
Sweep + cheapest Insert	0.035933	90209.0	24.68%
Sweep + nearest Insert	0.060136	88336.0	22.09%
Sweep + farthest Insert	0.060533	90689.0	25.34%
Sweep + random Insert	0.009349	90545.0	25.14%
Savings	10.440748	77700.0	7.39%
Nearest Neighbour	0.019190	84030.0	16.14%

Abbildung 14 Testergebnisse der Benchmark X-n1001-k43

Die Ergebnisse zeigen, dass der **Savings-Algorithmus** mit einer relativen Differenz zur optimalen Lösung von nur 7.39% die besten Ergebnisse in Bezug auf die Routenkosten liefert. Der **Nearest Neighbour-Algorithmus** und der **Nearest Insert-Algorithmus** schneiden ebenfalls relativ gut ab, mit relativen Differenzen von 16,14% bzw. 16,46%. Die Lösungszeit des **Random Insert** ist mit 0.007203 Sekunden am geringsten, allerdings ist sind dessen Routenkosten mehr als doppelt so hoch wie die Kosten der optimalen Lösung, weshalb wenn Geschwindigkeit gefragt ist tendenziell eher zwischen den **Sweep**-Varianten und dem **Nearest Neighbour** abgewägt werden sollte, da deren initiale Lösungen zu ähnlichen Laufzeiten weniger als halb so kostspielig sind wie die des **Random Inserts**.

Da der **Farthest Insert Algorithmus** die zweitschlechteste Lösung mitsamt der drittschlechtesten Laufzeit besitzt fällt er in diesem Fall als besonders ineffizient auf.

6.3.2 Depot in Randlage mit geclusterten Kunden (X-n979-k58)

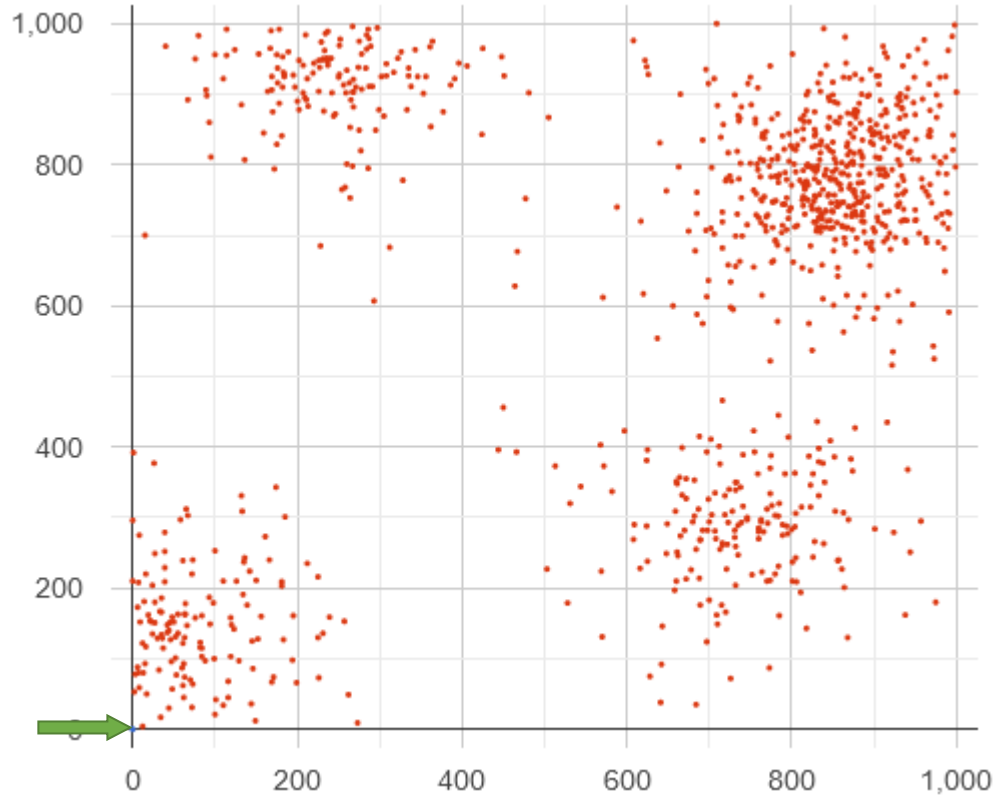

Abbildung 15 Plot der Benchmark X-n979-k58, das Depot(blau) wird unten links von dem grünen Pfeil markiert, die Kunden (rot) befinden sich innerhalb von 4 Clustern; Quelle: [19]

In dieser Benchmark analysieren wir ein Szenario, bei dem sich das Depot am Rand eines definierten Gebiets befindet und die Kunden auf 4 Ballungszentren verteilt sind. Dies ist ein typisches Szenario für die Verteilung von Waren aus Logistikzentren und regionalen Verteilungszentren, welche oftmals Güter innerhalb von mehreren Städten verteilen.

Algorithmus	Lösungszeit	Routenkosten	Relative Differenz zur optimalen Lösung
cheapest Insert	1.793758	133044.0	11.82%
nearest Insert	2.283612	128765.0	8.23%
farthest Insert	2.120272	145282.0	22.11%
random Insert	0.005748	211028.0	77.37%
Sweep + cheapest Insert	0.023967	143366.0	20.50%
Sweep + nearest Insert	0.046947	142696.0	19.94%
Sweep + farthest Insert	0.046744	146009.0	22.72%
Sweep + random Insert	0.007737	145415.0	22.22%
Savings	12.059800	123857.0	4.10%
Nearest Neighbour	0.018648	129548.0	8.89%

Abbildung 16 Testergebnisse der Benchmark X-n979-k58

Die Versuchsergebnisse zeigen, dass der **Savings-Algorithmus** mit einer relativen Abweichung von 4.10% zur optimalen Lösung erneut deutlich geringere Routenkosten liefert, als vergleichbare Verfahren. Falls die Lösungsgeschwindigkeit kritisch ist bietet sich in diesem Fall der **Nearest Neighbour Algorithmus** an, der trotz seiner geringen Laufzeit die drittbesten Ergebnisse erzielt. Trotz seiner erstaunlichen prozentualen Verbesserung zum vorherigen Fall erstellt der laut [21] bei TSP performanteste **Farthest Insert Algorithmus** in diesem Fall die viertlängste Route. Der **Random Insert** berechnet erneut die mit Abstand längste Route.

6.3.3 Depot in Zentrallage mit zufällig verteilten Kunden (X-n936-k151)

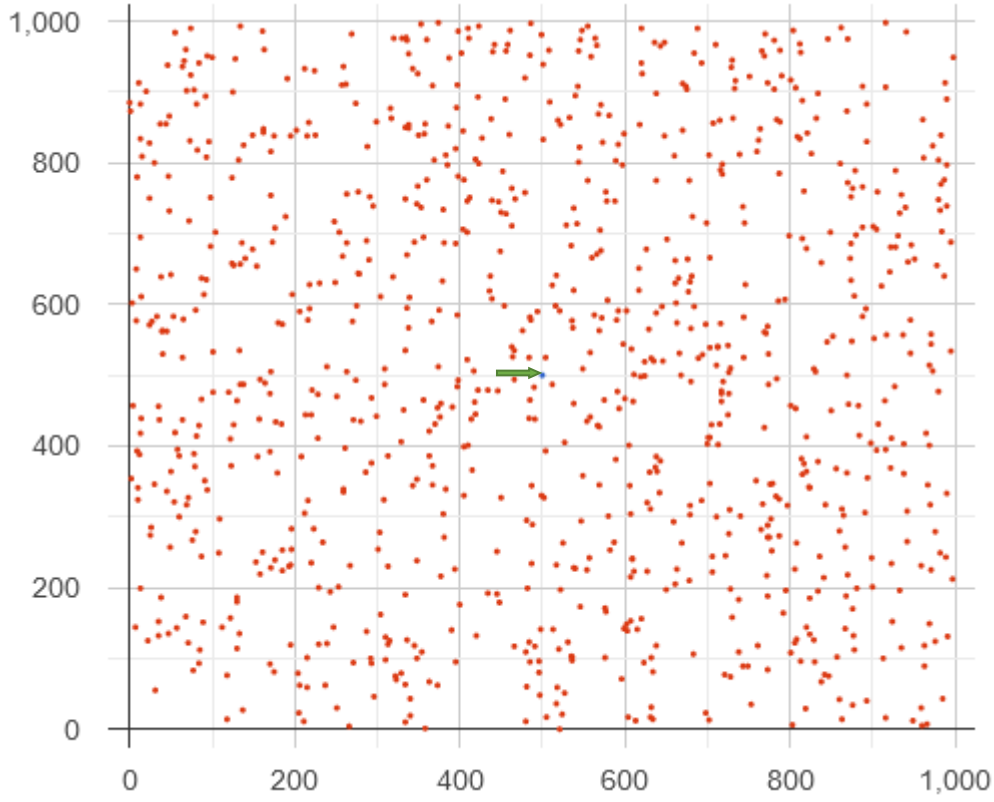

Abbildung 17 Plot der Benchmark X-n936-k151, das Depot(blau) wird in der Mitte von dem grünen Pfeil markiert, die Kunden (rot) sind zufällig verteilt; Quelle: [19]

Die Benchmark X-n936-k151 untersucht den Fall eines zentralgelegenen Depots mit zufällig verteilten Kunden. Dieses Szenario ist insbesondere für E-Commerce, Paketdienstleister, Kurierdienste, On-Demand Services wie Flink und Notfalldienste relevant, da für diese kurze Strecken und somit schnellere Lieferzeiten relevant sind, weshalb deren Depots meistens innerhalb eines Ballungszentrums liegen.

Algorithmus	Lösungszeit	Routenkosten	Relative Differenz zur optimalen Lösung
cheapest Insert	0.660046	181111.0	36.47%
nearest Insert	1.715541	174848.0	31.75%
farthest Insert	1.703322	317203.0	139.01%
random Insert	0.003655	390888.0	194.53%
Sweep + cheapest Insert	0.009071	177665.0	33.87%
Sweep + nearest Insert	0.018654	177217.0	33.53%
Sweep + farthest Insert	0.018774	200229.0	50.87%
Sweep + random Insert	0.005477	191366.0	44.19%
Savings	10.083122	145897.0	9.93%
Nearest Neighbour	0.019415	178119.0	34.21%

Abbildung 18 Testergebnisse der Benchmark X-n936-k151

Laut den Testergebnissen ist erneut der **Savings Algorithmus** der optimalen Lösung mit Abstand am nächsten. Obwohl der **Nearest Insert** das zweitbeste Ergebnis liefert stellt sich hier die Frage, ob es nicht effizienter ist eine der performanteren **Sweep** Variationen oder den **Nearest Neighbour** Algorithmus zu benutzen um eine Startlösung zu generieren und die extra 1,5 Sekunden für eine Verbesserungsheuristik zu verwenden. Da die **Farthest** und **Random Insertion** wiederholt unkompetetive Ergebnisse liefern werden diese ab diesem Punkt nicht mehr kommentiert.

6.3.4 Depot in Zentrallage mit geclusterten Kunden (X-n819-k171)

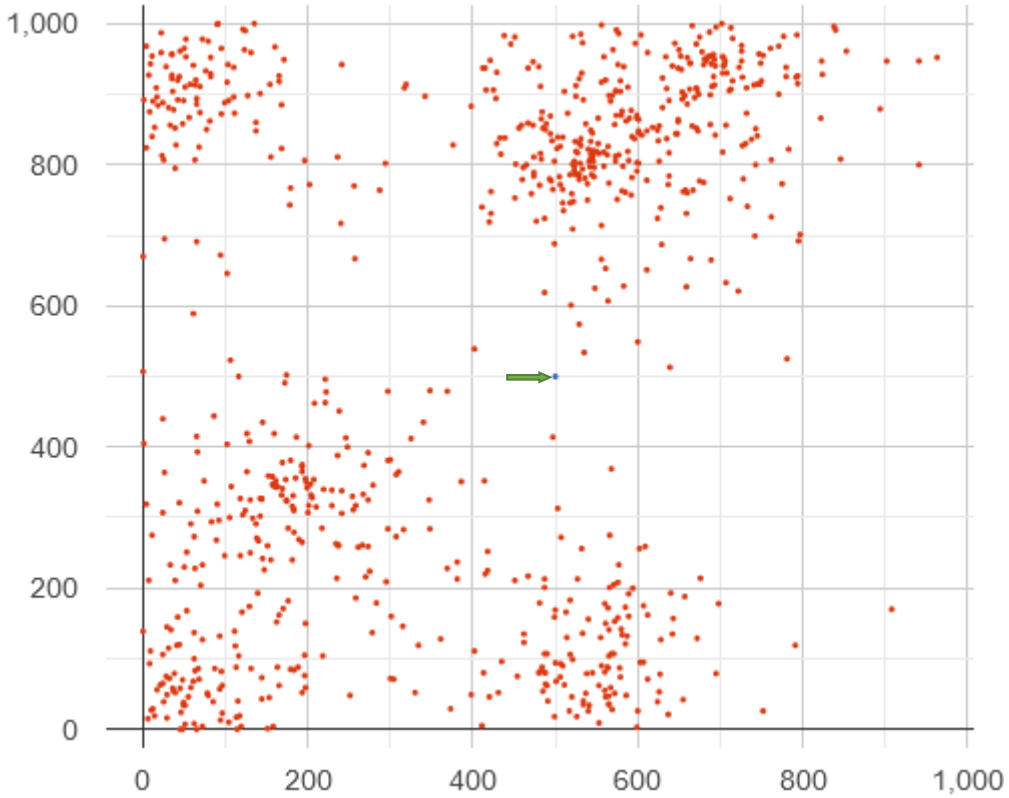

Abbildung 19 Plot der Benchmark X-n819-k171, das Depot(blau) wird in der Mitte von dem grünen Pfeil markiert, die Kunden (rot) sind auf 4 Cluster verteilt; Quelle: [19]

Innerhalb der Benchmark X-n153-k22 werden Waren aus einem zentralen Depot auf 2 Ballungszentren verteilt. Dieses Szenario tritt unter anderem häufig bei der Lebensmittel- und Getränkelieferung, Abfallentsorgung und Bürobedarfslieferung auf, da in diesen Disziplinen oftmals die belieferten Kunden oder Unternehmen geballt lokalisiert sind.

Algorithmus	Lösungszeit	Routenkosten	Relative Differenz zur optimalen Lösung
cheapest Insert	0.374009	179017.0	13.22%
nearest Insert	1.270930	176651.0	11.72%
farthest Insert	1.287469	349772.0	121.21%
random Insert	0.002826	393529.0	148.88%
Sweep + cheapest Insert	0.005900	191768.0	21.28%
Sweep + nearest Insert	0.011812	191663.0	21.21%
Sweep + farthest Insert	0.011869	195673.0	23.75%
Sweep + random Insert	0.004690	195133.0	23.41%
Savings	7.982050	166547.0	5.33%
Nearest Neighbour	0.015602	176804.0	11.82%

Abbildung 20 Testergebnisse der Benchmark X-n819-k171

Der **Savings-Algorithmus** identifiziert zwar die deutlich kürzeste Route, weist jedoch auch die längste Laufzeit aller erwähnten Algorithmen auf. Der **Nearest Insert** formuliert zwar die zweitbeste Lösung, aber der **Nearest Neighbour** Algorithmus liefert ähnlich kurze Route zu einer deutlich billigeren Laufzeit.

6.3.5 Großes Testproblem (Antwerp1)

Damit nicht der Eindruck entsteht, dass **Savings** in fast jedem Fall die beste algorithmische Wahl ist wird anhand eines wirklich großes Problem Namens Antwerp1 aus [30] mit 6000 Kunden demonstriert, dass die Laufzeit von **Savings** für große Probleme rasch zeitintensiv werden kann.

Algorithmus	Lösungszeit	Routenkosten	Relative Differenz zur besten bekannten Lösung
cheapest Insert	48.328827	540877.0	13.33%
nearest Insert	78.062326	521784.0	9.33%
farthest Insert	76.889488	1137170.0	138.26%
random Insert	0.040057	2308141.0	383.61%
Sweep + cheapest Insert	0.139714	659547.0	38.19%
Sweep + nearest Insert	0.276438	657304.0	37.72%
Sweep + farthest Insert	0.274976	692959.0	45.19%
Sweep + random Insert	0.057263	691799.0	44.95%
Savings	1251.897105	498916.0	4.53%
Nearest Neighbour	0.617074	525458.0	10.09%

Abbildung 21 Testergebnisse der Benchmark Antwerp1

Anhand von Antwerp1 kann man die kubische Komplexität des Savings Algorithmus gut erkennen: Obwohl die Anzahl der Kunden nur um das 6-Fache erhöht wurde ist die Lösungszeit um ungefähr das $6^3 = 216$ – Fache angestiegen. In diesem Fall wird es schwer vorstellbar, dass eine von dem **Nearest Neighbour** ausgehende Startlösung, welche anschließend mehr als 20 Minuten Zeit hat um Verbesserungs- und Metaheuristiken anzuwenden keinen kürzeren Tourplan finden kann.

6.4 Einordnung & Diskussion der Ergebnisse

Da die **Random Insertion** konstistent überdurchschnittlich lange Routen erstellt fällt es trotz ihrer kurzen Laufzeit schwer ihre Anwendung zu rechtfertigen. Da die **Farthest Insertion** in keinem einzigen Testfall kompetetive Ergebnisse liefern konnte wird von ihrer Benutzung abgeraten. Die hohe Lösungsqualität des **Savings-Algorithmus** zeigt hingegen, warum dieser so weit verbreitet ist. Bei laufzeitkritischen Problemen kann jedoch insbesondere die Verwendung des **Nearest-Neighbour-Algorithmus** oder des **Sweep-Algorithmus sinnvoll** sein, letzterer insbesondere bei geclusterter Kundenverteilung. Diese Algorithmen sind deutlich laufzeitärmer, wodurch die gewonnene Zeit für Verbesserungs- oder Metaheuristiken genutzt werden kann, welche die Lösung der Eröffnungsheuristik iterativ verbessern können. Dies ist wie anhand von Antwerp1 gezeigt wurde besonders relevant bei größeren Problemen, da der **Savings-Algorithmus** wegen seiner kubischen Komplexitätsordnung bei wachsender Problemgröße vergleichsweise laufzeitintensiv wird. Allerdings kann man laut den Testergebnissen aller Benchmarks aus [20] für jedes CVRP Problem mit 1000 oder weniger Kunden bedenkenlos den **Savings-Algorithmus** einsetzen, da dieser in unter 20 Sekunden Rechenzeit bei dem in 6.2 angegebenem Setup die unter den Eröffnungsheuristiken deutlich beste Lösung liefert. Bei deutlich größeren Problemen empfiehlt es sich anhand der Ergebnisse aus Antwerp1 aus Laufzeitgründen eher auf den **Nearest Neighbour Algorithmus** zurückzugreifen.

Eine interessante Frage ist, weshalb die Ergebnisse der Benchmarks so stark von den Ergebnissen der TSP-Heuristiken aus [21] abweichen. Allerdings ist dies unter genauerer Betrachtung nicht allzu überraschend, da bei einem TSP eine optimierte Route für alle Punkte eines Graphen gefunden werden soll, während bei dem CVRP in sinnvollen Anwendungsfällen mehrere Routen gebildet werden müssen. Da innerhalb eines TSP nur eine Route gebildet wird können die zu Beginn ineffizienten und weit auseinanderliegenden Punkte durch die Inserts deutlich effizienter gemacht werden, während bei einem CVRP die Route aufgrund der Kapazitätsbeschränkungen meist

abgebrochen werden muss, bevor die Anzahl der Knoten groß genug wird um die ineffizienten Zwischenrouten zu eliminieren.

Um Redundanz zu vermeiden wurden nur einige ausgewählte Benchmarktests präsentiert, allerdings wurden die Ergebnisse aller Benchmarktests, inklusive einer aggregierten Übersicht der Performanz aller Heuristiken in verschiedenen Benchmarkszenarien auf github [24] veröffentlicht. Außerdem kann der Python Code auf github [24] verwendet werden, um eigene Benchmarktests auszuführen oder die Ergebnisse dieser Arbeit zu validieren.

7 Fazit

In dieser Seminararbeit wurde das Vehicle Routing Problem (VRP) eingehend untersucht, indem grundlegende Definitionen und Anwendungsfälle dargelegt wurden. Durch die detaillierte Betrachtung verschiedener konstruktiver Heuristiken wie **Nearest Neighbour**, verschiedene Einfügeverfahren (**Cheapest, Nearest, Farthest, Random Insertion**), **Savings** und **Sweep** sowie Verbesserungsheuristiken wurde ein umfassender Überblick über die gängigsten Methoden zur Lösung des VRP geschaffen.

Anschließend wurden einige **Intra**- und **Inter-Route** Verbesserungsheuristiken vorgestellt, welche benutzt werden können, um eine bereits existierende Lösung zu verbessern.

Die Einbeziehung aktueller Forschungsthemen wie Machine Learning und Deep Learning zeigte, dass die Lösung des VRP ein dynamisches und innovatives Forschungsfeld ist, das kontinuierlich weiterentwickelt wird.

Zusätzlich wurde anhand des Beispiels **Large Neighbourhood Search** und der **Neural Large Neighbourhood Search** gezeigt, dass es möglich ist lernbasierte Methodiken zu verwenden, um bereits bestehende Heuristische Methoden weiter zu verbessern.

Ein Schwerpunkt dieser Arbeit lag auf dem Vergleich der verschiedenen konstruktiven Heuristiken anhand von Benchmarks. Die Ergebnisse zeigten, dass der **Savings-Algorithmus** bei mittelgroßen Problemen mit 100-1000 Kunden konsistent die deutlich kürzesten Routen unter den Eröffnungsverfahren liefert, allerdings benötigt er hierfür eine vergleichsweise deutlich erhöhte aber dennoch rechtfertigbare Laufzeit, was sich insbesondere bei größeren Problemen bemerkbar machte. Es wurde anhand eines großen Problems mit 6000 Kunden gezeigt, dass die Laufzeit des **Savings-Algorithmus** bei größeren Problemen schnell kritisch wird, weshalb bei diesen empfohlen wird den schnelleren aber dennoch recht performanten **Nearest-Neighbour-Algorithmus** zu verwenden.

Die Analyse und Diskussion der Benchmark-Ergebnisse verdeutlichten, dass die **Sweep- Insert-** und der **Nearest Neighbour-Algorithmus** insbesondere bei größeren Problemen Anwendung finden, da der **Savings-Algorithmus** aufgrund seiner Komplexität mehr kostbare Zeit benötigt, die alternativ für die Verbesserung einer schnell aufgestellten Lösung durch Verbesserungs- und Metaheuristiken genutzt werden könnte.

Insgesamt zeigt diese Arbeit, dass die Lösung des VRP ein komplexes und vielschichtiges Problem darstellt, das eine Vielzahl von Ansätzen und Methoden erfordert und dass heuristische Verfahren mit Bedacht gewählt werden müssen. Durch die Kombination und Weiterentwicklung dieser Techniken kann die Routenplanung optimiert werden, was sowohl wirtschaftliche als auch ökologische Vorteile mit sich bringt.

8 Quellenangaben

[1] P. Gora, D. Bankiewicz, K. Karnas, W. Kaźmierczak, M. Kutwin, P. Perkowski, S. Płotka, A. Szczurek, D. Zięba, On a road to optimal fleet routing algorithms: a gentle introduction to the state-of-the-art, In Intelligent Data-Centric Systems, Smart Delivery Systems, Elsevier, 2020, Pages 37-92

[2] I. Masudin, R. F. Sa'diyah, D. M. Utama, D. P. Restuputri, F. Jie, Capacitated Vehicle Routing Problems: Nearest Neighbour vs. Tabu Search, International Journal of Computer Theory and Engineering, Vol. 11, No. 4, August 2019

[3] P. Lenzner, Greedy Algorithmen - Hasso-Plattner-Institut, hpi.de, 19.06.2024

[4] Vahrenkamp, R., & Mattfeld, D. C. (2007). Logistiknetzwerke. Heusenstamm: Gabler Verlag

[5] Ziegler, H.-J., Binder, G., Niemeier, H.-V. and Schrenk, H. (1988): Computergestützte Transport- und Tourenplanung. Expert Verlag, Ehningen

[6] F. Liu, C. Lu, L. Gui, Q. Zhang, X. Tong, M. Yuan Heuristics for Vehicle Routing Problem: A Survey and Recent Advances, arXiv: 2303.04147, 2023

[7] Vidal, T., 2022. Hybrid genetic search for the CVRP: Open-source implementation and swap* neighborhood. Computers & Operations Research 140, 105643.

[8] Arnold, F., Sörensen, K., 2019a. Knowledge-guided local search for the vehicle routing problem. Computers & Operations Research 105, 32–46.

[9] P. Munari, T. Dellevoet, R. Spliet, A generalized formulation for vehicle routing problems, arXiv:1606.01935, 2017

[10] Ruiyang Shi, Lingfeng Niu, A Brief Survey on Learning Based Methods for Vehicle Routing Problems, Procedia Computer Science, Volume 221, 2023, Pages 773-780

[11] A. Hottung, K. Tierney, Neural large neighborhood search for the capacitated vehicle routing problem, in: ECAI 2020, 2020, pp. 443–450.

[12] Y. Wu, W. Song, Z. Cao, J. Zhang, A. Lim, Learning improvement heuristics for solving routing problems, IEEE Transactions on Neural Networks and Learning Systems (2021) 1–13.

[13] M. Kim, J. Park, J. Kim, Learning collaborative policies to solve np-hard routing problems, Advances in Neural Information Processing Systems 34 (2021) 10418–10430.

[14] A. Vaswani, N. Shazeer, N.Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, 2017, In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[15] Ni, Q.; Tang, Y. A Bibliometric Visualized Analysis and Classification of Vehicle Routing Problem Research. *Sustainability* **2023**, *15*, 7394

[16] D. Pisinger, S. Røpke Large Neighborhood Search, Handbook of Metaheuristics, 2010

[17] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming), volume 1520 of Lecture Notes in Computer Science, pages 417–431, 1998

[18] Christofides, Eilon, CVRPLIB, Set E, 1969

[19] I. Lima, D. Oliveira, E. Queiroga, CVRPLIB, 2014

[20] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, A.Subramanian, New benchmark instances for the Capacitated Vehicle Routing Problem, European Journal of Operational Research, Volume 257, Issue 3,2017, Pages 845-858,ISSN 0377-2217 (The original Paper and dataset were published in 2014 according to CVRPLIB)

[21] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, An Analysis of Several Heuristics for the Travelling Salesman Problem, SIAM Journal on Computing 6(3):563-581, 1977

[22] B. L. Golden, T. L. Magnanti, H. Q. Nguyen, Implementing vehicle routing problems, Networks Volume 7, Issue 2, 1977

[23] R. Lasch, Strategisches und operatives Logistikmanagement: Distribution. Springer Gabler, Wiesbaden, 2020

[24] M. Uhl, CVRP, Github, https://github.com/MaxUhl98/CVRP/tree/main, 2024

[25] T. Vidal, T. G. Crainic, M. Gendrau, C. Prins, A unified solution framework for multi-attribute vehicle routing problems, European Journal of Operational Research, Volume 234, Issue 3, 2014, Pages 658-673, ISSN 0377-2217

[26] K. Helsgaun, An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Travelling Salesman and Vehicle Routing Problems, Department of Computer Science, Roskilde University, DK-4000 Roskilde, Denmark, December 2017

[27] Sbai, I., Krichen, S. & Limam, O. Two meta-heuristics for solving the capacitated vehicle routing problem: the case of the Tunisian Post Office. *Oper Res Int J* **22**, 507–549 (2022)

[28] Karakostas, Panagiotis & Sifaleras, Angelo & Georgiadis, Michael. (2022). Variable neighborhood search-based solution methods for the pollution location-inventory-routing problem. Optimization Letters. 16. 10.1007/s11590-020-01630-y.

[29] Shao, S., Lai, K.K. & Ge, B. A multi-period inventory routing problem with procurement decisions: a case in China. *Ann Oper Res* **324**, 1527–1555, 2023

[30] Arnold, Florian & Gendreau, Michel & Sörensen, Kenneth. Efficiently solving very large scale routing problems. 2017

[31] Visual Generation, Fuel truck hand drawn outline doodle icon. Tanker truck, gasoline station and fuel delivery, cistern concept. Vector sketch illustration for print, web, mobile and infographics on white background.

[32] keen1220, gas station icon logo vector illustration. fuel pump symbol template for graphic and web design collection Free Vector

[33] Imran S M, Multi Objective Optimisation (MOO)| Data Science Optimisation| Vehicle Routing Problem 🚚