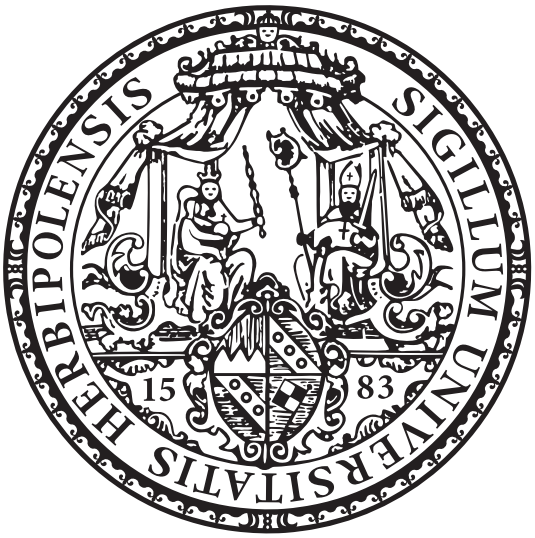
Praktikumsbericht

Carpooling Problem mit flexiblen Fahrern

Jonas Barth

Abgabedatum: 16. Mai 2024
Betreuer: Prof. Dr. Marie Schmidt



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

1 Forschungsthema

In dieser Forschungsarbeit soll es um das Thema Ridesharing gehen. Genauer gesagt wird ein Spezialfall des *Commute Trip Sharing Problem* (CTSP) betrachtet. Dies ist ein Optimierungsproblem bei dem standardmäßig eine Menge von Arbeitern täglich zu ihrer Arbeit und zurückfährt. Dafür müssen statisch Teams, bzw. Carpools gebildet werden. Das Ziel ist vor allem die Fahrtkosten für die einzelnen Arbeitern zu minimieren, sowie den insgesamt anfallenden CO_2-Ausstoß.

Das Carpooling Problem wurde in der Literatur bereits ausgiebig erforscht. Jedoch wurde nur selten der Effekt von unterschiedlichen Arbeitszeiten genauer untersucht. So kann der Fall auftreten, dass es sinnvoll ist die Carpool-Teams zwischen Hin- und Rückfahrt durchzuwechseln. Dies wurde nach meinem Kenntnisstand nur bei dem in [HVHL20] definierten CTSP genauer behandelt. Die Unterschiede zu dieser Arbeit werden jedoch sein, dass alle Arbeiter ein Auto zur Verfügung haben und hier nur ein einzelner Arbeitsort betrachtet wird. Das bedeutet, dass alle Arbeiter stets zum gleichen Ziel wollen.

1.1 Problemstellung

Die Aufgabenstellung zeichnet sich allgemein durch die folgenden Charakteristiken aus. Es geht um eine Menge von Arbeitern einer Firma, die jeden Tag zur Arbeit pendeln möchten. Jeder Arbeiter hat eigene Arbeitszeiten. Also eine Uhrzeit zum Arbeitsbeginn und einen Zeitpunkt, ab dem er sich wieder auf den Weg nach Hause machen möchte.

Außerdem hat jeder Arbeiter ein Auto zur Verfügung mit dem auch andere mitgenommen werden können, wobei in einem Auto natürlich nur begrenzt viel Platz ist. Wichtig ist auch, dass ein Auto nicht zwingend gefahren werden muss, sondern man es auch stehen lassen kann. Ein Arbeiter möchte nicht beliebig weit um sein Ziel fahren, oder lange warten müssen, deshalb ist er nur gewillt eine gewisse Gesamtreisezeit in Kauf zu nehmen.

Insgesamt wird eine Partitionierung der Arbeiter in Gruppen bzw. Teams gesucht, die dann gemeinsam ein Auto für eine Fahrt füllen. Zusätzlich muss für jedes Team ein Fahrer gewählt, sowie die genaue, kürzeste Route gefunden werden. Verschiedene Optimierungsziele sind dabei relevant. Zum einen sollte die Anzahl an Autos, bzw. die insgesamt zurückgelegte Fahrtstrecke aller Fahrzeuge minimiert werden. Aber auch die Fahrtzeit der einzelnen Arbeiter muss möglichst gering gehalten werden.

Das Besondere an diesem Problem ist, dass die Teams zwischen Hin- und Rückfahrt unterschiedlich sein können. Dies macht vor allem aufgrund der verschiedenen Arbeitszeiten Sinn. Jedoch ist das Problem dadurch zusätzlich erschwert, da die Lösung eine gültige Zuweisung von Fahrern haben muss. Also ein Arbeiter, der als Fahrer gewählt wird, muss sowohl bei Hin- und Rückfahrt fahren.

2 Zielsetzung

Zu den Vorhaben der Arbeit gehört ein Überblick über den Stand in der Forschung zu diesem Thema, vor allem zu *Carpooling*. Da in der Literatur leider an vielen Stellen eine inkonsistente Namensvergebung zu finden ist, soll so zukünftige Forschung auf diesem Gebiet erleichtert werden. Außerdem gehört eine präzise und überschaubare Definition der Problemstellung zu den ersten Aufgaben.

Das hauptsächliches Ziel ist es einen greedy, heuristischen Algorithmus zu konzipieren. Dieser soll eine best-effort Lösung in annehmbarer Zeit liefern und so auch für große Probleminstanzen geeignet sein. Ein modularer Aufbau wäre von Vorteil, um Zwischenschritte mit verschiedenen Strategien prüfen zu können.

Zum Test des Algorithmus soll auch ein exakter Lösungsalgorithmus implementiert werden. Die beiden Ansätze sollen miteinander verglichen werden, und zwar mit Hilfe von Datensätzen aus dem *CaMoS* Framework von [Feh24]. In diesem sind die Wohnorte der Studenten der Universität Würzburg in Stadt und Umland vermerkt, die täglich zur Uni reisen möchten.

3 Forschungsstand

3.1 Ridesharing

Auf dem übergeordneten Gebiet *Ridesharing* gibt es bereits einige Arbeiten. Unter *Ridesharing* versteht man das Problem von Passagieren und Fahrern, die je über einen persönlichen Start- und Zielort, sowie zeitliche Constraints verfügen. Es soll ein Plan gefunden werden, nach diesem die Fahrer Passagiere einmalig abholen und dann wieder absetzen. Hier sind unterschiedliche Zielfunktionen denkbar. Wer als Fahrer im System hinterlegt ist, benutzt in jedem Fall auch sein Auto für den Trip. Die Passagiere haben kein Auto zur Verfügung und können sich im klassischen Fall auch nicht zu Fuß fortbewegen. Eine gute Zusammenfassung dieses allgemeinen Problems ist in [AESW10] zu finden. Dabei wird zwischen den folgenden Problemstellungen unterschieden:

- Ein Fahrer - Ein Passagier
- Ein Fahrer - Mehrere Passagiere
- Mehrere Fahrer - Ein Passagier
- Mehrere Fahrer - Mehrere Passagiere

Hierbei bezieht sich die Anzahl an Fahrern darauf, dass Passagiere gewillt sind entweder nur mit einem Fahrer zu fahren, oder auch mehrfach umsteigen können und so ihren Fahrer wechseln. Fahrer hingegen sind gewillt entweder nur einen Passagier zu transportieren, oder mehrere gleichzeitig. Zusätzlich kann das Problem statisch oder dynamisch betrachtet werden. Im ersten Fall sind alle Teilnehmer und Bedingungen von Beginn an bekannt. Im zweiten Fall tauchen die Nutzer erst nach und nach auf und man muss den bereits ausgeführten Plan dynamisch anpassen.

Des Weiteren geben die Autoren aus [AESW10] einen exakten und effizienten Algorithmus zum Lösen des statischen, einfachen Problems mit einem Fahrer und einem Passagier pro Fahrzeug an. Dieser basiert auf einem Matching, das mit Flussalgorithmen gelöst wird. Die anderen Varianten sind deutlich schwieriger zu lösen, vgl. [AESW10].

Für das Problem mit einem Fahrer und mehreren Passagieren haben beispielsweise [HW12] einen genetischen Algorithmus vorgeschlagen. Sie haben sowohl das statische Problem betrachtet, sowie eine Vorgehensweise zum dynamischen Einfügen von Akteuren.

Ein weiterer *Ridesharing* Algorithmus kommt von [MJ17]. Die Autoren versuchen das dynamische Problem mit mehreren Fahrern und Passagieren exakt zu lösen. Dabei greifen sie auf dynamische Programmierung zurück. Jedoch ist der Algorithmus letztlich bei großen Instanzen zu laufzeitintensiv und ist besser für nur einen Passagier pro Fahrzeug geeignet. In der Arbeit von [MJ17] ist zudem ein guter Überblick über weitere Algorithmen zum Thema *Ridesharing* zu finden.

3.2 Carpooling

Generell geht es beim *Carpooling* darum eine Menge von Arbeitern zu einem Zielort, bzw. Firma und wieder nach Hause zu bringen. Manche sind Fahrer und andere Passagiere. Zudem gibt es wieder zeitliche Constraints, also verschiedene Arbeitszeiten. Damit hat das Thema viele Überschneidungen mit *Ridesharing* und kann auch als Spezialfall dessen gesehen werden. Jedoch gibt es unterschiedliche Varianten des Problems, die sich vor allem in den Punkten aus Tabelle 1 unterscheiden.

Probleme	Fahrer frei	Passagiere	mehrere Ziele	Zeitconstraints	Teams wechseln	Route gesucht
DCPP		✓		✓		✓
LCPP	✓			✓		
CSP		✓	✓			✓
CSPTW		✓	✓	✓		✓
CTSP	✓	✓	✓	✓	✓	✓
SDCTSP	✓			✓	✓	✓

Tab. 1: Fahrer frei: Fahrer beliebig wählbar, nicht Teil des Inputs. **Passagiere:** es gibt Teilnehmer ohne Auto. **mehrere Ziele:** unterschiedliche Zielorte sind vorgesehen, an denen Nutzer abgesetzt werden können. **Zeitconstraints:** Teilnehmer haben verschiedene An- und Abfahrtszeiten. **Teams wechseln:** Teams müssen für Hin- und Rückfahrt nicht gleich sein. **Route gesucht:** die exakte Route ist Teil der Lösung.

Alle mir bekannten Formulierungen in der Literatur gehen von einem Fahrer aus. Somit ist kein Umsteigen der Arbeiter vorgesehen, wobei es durchaus Fälle geben kann, in denen so Fahrtzeit eingespart wird.

Eine bekannte Problemformulierung ist das *Daily Car Pooling Problem*, bei welchem von festen Fahrern ausgegangen wird. Außerdem gibt es für den Teilnehmer eine früheste

Zeit zu der er bereit ist loszufahren, sowie eine späteste Zeit um anzukommen und eine maximale Fahrtzeit. Gleiches gibt es auch für den Rückweg. Für jeden Passagier gibt es einen Penalty, falls er nicht mitgenommen wird. Es wird eine Route für jeden Fahrer gesucht mit dem Ziel, dass möglichst viele Passagiere mitgenommen werden können bei geringen Fahrtkosten.

Die Autoren in [BMM04] lösten dieses Problem exakt mit einem MILP. Außerdem wurde eine Heuristik vorgestellt. Hier wird eine Lagrange-Relaxierung durchgeführt, bei der das Mengen-Partitionierungs Constraint weggelassen wird. Daraufhin wird das Ergebnis mit einem heuristischen Algorithmus zu einer validen Lösung transformiert.

An diesem Problem arbeiteten auch [CdLHM04], die eine Softwareanwendung für den gesamten Prozess entwickelten. Für die Planung wurde ein heuristischer Algorithmus implementiert, der Passagiere nach einem greedy-approach zuordnet.

Es ist jedoch anzumerken, dass die Algorithmen der Autoren in [BMM04] und [CdLHM04] nur für eine spezielle Variante des Problems geeignet sind. Und zwar wird jeweils der Hin-, oder Rückweg komplett unabhängig vom Gegenstück angeschaut. Es wird also nicht beachtet, ob ein Passagier für beide Seiten eingeteilt wird. Somit ist das Problem sehr viel näher am normalen *Ridesharing*. Nach Definition aus [BMM04], wenn beide Richtungen gleichzeitig betrachtet werden, ist kein Durchwechseln der Teams vorgesehen.

Eine weiteres definiertes Problem ist das *Long Term Car Pooling Problem*(LCPP). Es zeichnet sich dadurch aus, dass alle Arbeiter über Autos verfügen. Außerdem gibt es auch hier zusätzliche zeitliche Constraints, wie maximale Fahrtzeit und Zeitfenster in der ein Teilnehmer losfahren möchte. Speziell ist, dass bei diesem Problem nur die Einteilung in Teams für einen längeren Zeitraum gesucht ist. Eine Festlegung der Fahrer, bzw. einer genauen Route ist zunächst nicht vorgesehen, stattdessen sollen sich die Arbeiter abwechseln können. Es wird versucht die Anzahl an Teams zu minimieren, aber auch die Fahrtkosten der potenziellen Fahrer eines Teams im Mittel möglichst gering zu halten.

Die erste Erwähnung des Problems stammt aus [VMS02]. In dieser Definition ist leider kein Durchwechseln der Teams zwischen Hin- und Rückfahrt vorgesehen ist, obwohl es die Problemstellung durchaus zulassen würde. Stattdessen fährt jedes Team gemeinsam hin und zurück. Zuletzt beschäftigte sich [KBB21] mit LCPP und nutzte eine Metaheuristik. Generell finden sich in der Literatur viele Heuristiken hierfür, die auch in [KBB21] Erwähnung finden.

Das *Carpool Service Problem*(CSP) aus [HJL14] ist ähnlich zum DCPP, da auch hier die Fahrer im Input bereits festgelegt sind. Jedoch gibt es mehrere mögliche Ziele. Hier geht man von einem gleichzeitigen Start aller Nutzer aus und es gibt keine Zeitconstraints. Erst im erweiterten *Carpool Service Problem with Time Windows*(CSPTW) von [HJL18] gibt es noch maximale Fahrtzeiten und früheste Startzeiten. Es wird bei beiden Problemen versucht möglichst viele Passagiere zu transportieren, sowie die Fahrtdistanzen und Zeiten zu minimieren. Ähnlich wie beim DCPP ist es leider so, dass die Probleme eher normalen *Ridesharing* ähneln, da nicht Hin- und Rückweg gleichzeitig betrachtet werden. Es wird also nicht darauf geachtet, ob Passagiere sowohl auf Hin- und Rückweg transportiert werden.

In jüngerer Vergangenheit wurde in [HVHL20] auf diesem Gebiet geforscht. Sie defi-

nierten das CTSP und stellten hierfür zwei exakte Algorithmen vor. Im Gegensatz zu anderen Arbeiten werden hier, sowohl unterschiedliche Arbeitszeiten beachtet und eine freie Auswahl über die Fahrer ist möglich. Somit ist dies meines Wissens die erste und einzige Arbeit, die Carpooling mit Durchwechseln der Teams vorsieht. Noch dazu kommt, dass eine exakte Route gefunden wird und sogar mehrere Zielorte gleichzeitig betrachtet werden.

4 Definition

Im folgenden möchte ich das Problem formal als das *Single Destination Commute Trip Sharing Problem* (SDCTSP) definieren. Es gibt einen Zielort O und eine Menge von Nutzern D. Ein Nutzer $d \in D$ hat einen Pick-up Ort $o(d)$ von dem er an einem Tag startet und zudem er wieder zurückkehren will. Jeder Nutzer hat ein Auto, dieses hat eine maximale Kapazität $c(d)$ an Passagieren inklusive Fahrer.

Des weiteren hat ein Nutzer ein Zeitintervall $e_{\text{to}} = [t_{\text{to}_{\text{begin}}}, t_{\text{to}_{\text{end}}}]$. Zu einem Zeitpunkt, der in diesem Intervall liegt, muss der Nutzer am Zielort ankommen. Genauso gibt es ein Intervall $e_{\text{from}} = [t_{\text{from}_{\text{begin}}}, t_{\text{from}_{\text{end}}}]$ zu dem er wieder vom Zielort abfährt. Letztlich verfügt er über eine maximale Fahrtzeit für einen Trip $t_{\max}(d)$. Somit darf jeweils für Hin- und Rückweg die Reisezeit diesen Wert nicht überschreiten.

Eine Route $R_{d,\text{To}}, R_{d,\text{From}}$ ist eine Sequenz von Nutzern mit dem Fahrer d zum Zielort, bzw. von ihm weg. Die Sequenz beschreibt die Reihenfolge in der die Nutzer abgeholt, oder abgesetzt werden. Die Menge von *machbaren* Routen $\mathcal{R}_{\text{To}}$, bzw. $\mathcal{R}_{\text{From}}$, beschreibt alle Routen, die für jeden ihrer zugehörigen Nutzer alle genannten zeitlichen Constraints einhält und die Kapazität des Autos des Fahrers nicht übersteigt.

Gesucht ist also eine Auswahl von machbaren Routen für Hin- und Rückfahrt. Im einfachsten Fall soll die Anzahl an Teams, bzw. Routen minimiert werden.

Das Problem kann wie folgt, als ILP formuliert werden. Dafür werden zunächst durch brute-force alle machbaren Routen ermittelt. Nun erstellt man für jede
Route $R_{d,\text{To}} \in \mathcal{R}_{\text{To}}$ und $R_{d,\text{From}} \in \mathcal{R}_{\text{From}}$ eine Variable $x_{R,d} \in \{0, 1\}$. Somit ergibt sich:

$$\min \quad \sum x_{R,d} \tag{1}$$

$$\sum_{R_{d',\text{To}} :\, d \in R_{d',\text{To}}} x_{R,d'} = 1 \qquad \forall d \in D \tag{2}$$

$$\sum_{R_{d',\text{From}} :\, d \in R_{d',\text{From}}} x_{R,d'} = 1 \qquad \forall d \in D \tag{3}$$

$$\sum_{R_{d,\text{To}}} x_{R,d} = \sum_{R_{d,\text{From}}} x_{R,d} \qquad \forall d \in D \tag{4}$$

Die Zielfunktion ist in (1) zu sehen. Die Gleichung (2) fordert, dass jeder Nutzer genau einmal zur Arbeit mitgenommen wird, egal ob als Fahrer oder Passagier. In (3) sichert man das gleiche für die Rückfahrt ab. Zuletzt sorgt (4) dafür, dass die Anzahl wie oft ein Nutzer Fahrer ist, bei Hin- und Rückfahrt identisch ist. Durch die beiden vorherigen

Gleichungen ist bereits klar, dass die Summen sowieso nur 0 oder 1 sein können. Also ist so gesichert, dass die Fahrer von Hin- und Rückfahrt zueinander passen.

5 Methodik

Für die exakte Lösung ist ein *Mixed Integer Linear Program* denkbar, wie es auch bereits von [HVHL20] aufgestellt wurde. Wobei die Autoren bereits komplexere Lösungsansätze implementiert haben, um die Laufzeit noch weiter zu verbessern, basierend z.B. auf *column-generation*. Dieser Algorithmus wäre für dieses Problem auch passend, jedoch ist die Implementierung nicht öffentlich verfügbar. Aus ihrem Paper geht jedoch hervor, dass die einfache MILP Variante, mit Solvern wie CPLEX auch annehmbare Ergebnisse erzielt auf kleineren Datensätzen. Nach einem räumlichen Preclustering auf eine Größe von 100 Arbeitern pro Instanz, mit 5 Plätzen per Auto meldeten die Autoren ca. 16 Minuten Berechnungszeit für ein Cluster. Genaueres zu diesem Ansatz ist in Abschnitt 6.1 zu lesen.

Der heuristische Algorithmus funktioniert auf Basis von Clusterings. Der Pseudocode ist zu sehen in 1. Zunächst werden die Arbeiter passend zu ihren Arbeitszeiten in Cluster eingeteilt. Daraufhin wird pro Cluster nochmals räumliches Clustering angewendet und somit weiter unterteilt. Für diesen Zwischenschritt sind verschieden Algorithmen denkbar. Da ein wichtiger Punkt das Zusammenpassen der Fahrer bei Hin- und Rückfahrt ist, muss im folgenden eine Fahrerauswahl stattfinden und ggf. Cluster nochmals angepasst werden. Dafür kann das Problem als bipartiter Graph modelliert und ein maximum Matching gefunden werden.

Ziel soll es sein die Anzahl an Teams möglichst gering zu halten. Dadurch erhoffe ich mir ebenfalls eine geringe Gesamtfahrtdistanz. Dahingehend soll der Algorithmus auch evaluiert und mit einer exakten Lösung verglichen werden. Sowie auch der Unterschied was die Laufzeit der Algorithmen angeht.

Im Pseudocode 1 sind einige Funktionsaufrufe zu sehen. Sie sorgen für einen modularen Aufbau und ermöglichen es die Struktur wiederzuverwenden, dabei können in den Zwischenschritten unterschiedliche Strategien angewendet werden. So können verschiedene Zielfunktionen je nach Implementierung angelegt werden. Zunächst habe ich mich für folgende Ansätze entschieden.

Die Methode *makeBubbles* teilt die Nutzer in zeitliche Cluster, im folgenden Bubbles genannt, ein basierend auf den Intervallen t_{to} oder t_{from}. Konkret wird dies erreicht über iteratives wählen von maximalen Cliquen im zugehörigen Intervallgraphen. Diese werden jeweils aus dem Graphen herausgeschnitten, bis dieser leer ist.

Dann wird durch *makeCluster* für eine zeitliche Bubble räumliche Cluster erstellt. Unterschiedliche Clustering Algorithmen sind denkbar. Ich habe mich für einen hierarchischen Ansatz entschieden. Dieser startet mit einem Cluster für jeden einzelnen Nutzer. Dann werden immer die Paare von Clustern zuerst gemerged, die über den geringsten Abstand zwischen ihren Centroids verfügen. Dafür muss zum einem bei jedem merge geprüft werden, ob die Größe des resultierenden Clusters überhaupt mit den Fahrzeugkapazitäten der zugehörigen Nutzer kompatibel ist. Komplizierter ist es sicherzustellen,

Algorithmus 1: createTeams(D, o)

```
1  teams = list()
2  bubblesToWork = list()
3  bubblesFromWork = list()
4  for d ∈ D do
5      bubblesToWork = makeBubbles(D.t_start)
6      bubblesFromWork = makeBubbles(D.t_end)
7  for bubble ∈ (bubblesToWork ∪ bubblesFromWork) do
8      teams.append(makeCluster(bubble, o))
9  matching = makeMatching(teams)
10 for team ∈ teams do
11     if team not matched yet then
12         partner = createCluster(team)
13         matching.append((team, partner))
14 for pair ∈ matchedTeams do
15     findDriver(pair)
16 return matching
```

ob das mögliche Cluster überhaupt eine *machbare* Route hat.

Dafür muss jede Permutation der Teilnehmer geprüft werden, bis eine passende gefunden wird. Ansonsten wird der merge nicht durchgeführt. Fahrer für die eine Permutation gefunden wurde, mit ihnen an der ersten Stelle, werden als *mögliche* Fahrer hinterlegt.

In der Methode *makeMatching* wird ein bipartiter Graph erstellt, mit Teams für die Hinfahrt auf der einen und den Rückfahrtsteams auf der anderen Seite. Zwei Teams, bzw. Knoten haben eine Kante genau dann, wenn sie sich in ihren *möglichen* Fahrern überschneiden. Dann wird ein maximum Matching im bipartiten Graphen gesucht. Möglichst viele Teams werden zu Paaren gematched.

Für Teams die ungematched bleiben, muss im folgenden einer ihrer möglichen Fahrer nochmals aus seinem bisherigen Cluster auf der Gegenseite entnommen und als neues Team hinzugefügt werden. Diese Cluster werden letztlich noch zusammen gematched.

Abschließend gilt es nur noch für jedes Team einen festen Fahrer festzulegen, bzw. die exakte Route zu ermitteln. Dies kann durch brute-force erreicht werden, da nur es jeweils nur wenige Teammitglieder gibt.

6 Evaluation

Im *CaMoS*[1] Framework von [Feh24] sind fünf verschiedene Datensätze hinterlegt. Sie unterscheiden sich anhand der Heimatorte der Nutzer bzw. Studenten. Außerdem wurden

[1] https://github.com/H-Fehler/CaMoS

ihnen zufällige Hin- und Rückfahrtszeiten zugeordnet. Die Zielorte sind vier Fakultäten der Universität Würzburg. Pro Datensatz sind ca. 6000 Studenten vermerkt. Die Wohnorte liegen alle zwischen 2 und 25 km Luftlinie vom jeweiligen Zielort entfernt. Der Datensatz zeichnet sich vor allem durch eine hohe Variabilität der Studenten aus. Im Gegensatz zu typischen Büroarbeitszeiten, die sehr homogen sind, wollen die Teilnehmer hier teilweise sehr früh, oder erst nachmittags zur Universität, was in diesem Kontext auch Sinn ergibt.

6.1 Ähnliche Algorithmen

Zum Vergleich unseres konzipierten Algorithmus haben wir noch weitere Lösungsansätze geprüft. Diese stellen wir im folgenden vor.

Im *All Single* Ansatz fährt jeder Nutzer selbst, dies bedeutet den Worst Case was die gesamte Fahrtdauer- und Kilometerzahl aller Autos betrifft. Die durchschnittliche Auslastung ist offensichtlich immer 1. Die durchschnittliche Reisezeit, also wie lange ein Nutzer im Mittel im Auto verbringt pro Fahrt, ist hier stets minimal. Da in diesem Fall kein wirklicher Algorithmus gebraucht wird, ist die Berechnungszeit 1 Minute bzw. meistens sogar etwas kürzer.

Des Weiteren wird der *Ridesharing* Algorithmus von [Feh24] getestet. Hier hat jeder Nutzer eine maximale Gehdistanz, so weit ist er bereit sich zum Startpunkt eines anderen Fahrers zu bewegen. Standardmäßig ist diese auf 1200 Meter gesetzt. Die Fahrer holen also niemanden auf dem Weg ab, aber sie können Personen an anderen Campus absetzen. Dies ist ein Vorteil gegenüber des vorgeschlagenen Clustering Algorithmus. Bei diesem werden Agenten pro Zielort einzeln betrachtet. Aber auch in diesem Lösungsansatz darf die maximale Fahrtzeit nicht überschritten werden. Im Gegenatz zu allen anderen betrachteten Algorithmen, werden Anfragen hier dynamisch entgegengenommen. Matchings werden nach und nach heuristisch für die einzelnen Teilnehmer gesucht, mit dem Ziel möglichst wenige Fahrzeuge zu nutzen. Ein Nachteil des Algorithmus ist es, dass auch Nutzer verloren gehen können, also nicht zwingend ein Partner zur Heimreise gefunden wird.

Der Algorithmus *Direct Path* stammt aus der Bachelorarbeit von [Hol24]. Die Fahrer fahren in diesem Fall stets die direkte Route, die sie auch nehmen würden, wenn sie alleine fahren. Aber auch in diesem Ansatz gibt es eine maximale Gehdistanz. Die Passagiere sind bereit diese Distanz zurückzulegen, und sich entlang der Route eines Fahrers zu platzieren. An dieser Stelle können sie dann mitgenommen werden. Auch dieser Algorithmus geht heuristisch vor, um Partner bzw. Teams zu finden, mit dem Ziel möglichst wenige Personen alleine fahren zu lassen.

Letztlich wurde der *Route Enumeration Algorithm*(REA) von [HVHL20] implementiert. Es müssen alle *machbaren* Teamkombinationen geprüft und jeweils die kürzeste Route gefunden werden. Alle Mitglieder eines Teams fahren dabei zu einem gemeinsamen Ziel. Letztlich wird mit einem MILP-Solver das in Gleichung 1 beschriebene Problem gelöst. Als Zielfunktion wurde die Gesamtfahrtzeit aller Autos verwendet, die so minimiert werden soll. Nun können unsere heuristischen Ergebnisse einer exakten Lösung gegenüber gestellt werden. Diese ist jedoch sehr laufzeitintensiv. Um dem etwas entge-

genzuwirken ist ein räumlicher Preclustering Schritt nötig. Das Clustering wird ahnhand einer radialen Sortierung erstellt. Die Clustergrößen wurde für die folgenden Tests auf maximal 100 gesetzt. Der exakte Algorithmus, sowie der Clustering Algorithmus wurde im Zuge dieser Arbeit von mir implementiert[2].

6.2 Testergebnisse

Nun habe ich den Clustering Algorithmus mit den zuvor erwähnten Lösungsansätzen verglichen und einige wichtige Kennzahlen herausgearbeitet. Erste Ergebnisse sind in Tabelle 2 vermerkt. Es wurden 3 Datensätze betrachtet. Während beim ersten alle Nutzer berücksichtigt wurden, beschränken wir bei den anderen Datensätzen die räumliche Dichte, sodass bei Datensatz 2 und 3 nur ca. 50% bzw. 20% der möglichen Anzahl an Nutzern zu finden ist.

Alle Tests wurden auf einem Intel Core i5-9500 Prozessor mit sechs 3 GHz Kernen durchgeführt und einem RAM-Speicher von 16 GB.

[2]https://github.com/barjon0/CaMoS

	Dataset 1 (6524 agents)					Dataset 2 (3298 agents)					Dataset 3 (1303 agents)				
	AS	RS	DP	Clust.	REA	AS	RS	DP	Clust.	REA	AS	RS	DP	Clust.	REA
Total km travelled (cars)	120103	91088	58436	63754	54138	62380	51064	35168	38010	30340	24239	21650	16449	16606	13828
Total min. travelled (cars)	130833	99276	71779	85469	68973	67644	55216	41806	49589	38252	26341	23449	19151	20053	17506
Avg. Seat Count	1	1.49	3.02	2.65	3.19	1	1.32	2.49	2.31	2.97	1	1.19	2.00	2.40	2.63
Number of rides alone	13048	6452	700	1475	780	6644	3925	726	1060	494	2606	1861	535	403	249
Avg. min. travelled (user)	10.03	11.49	15.59	12.96	13.32	10.18	11.35	18.31	13.01	13.57	10.10	10.89	17.13	13.20	13.50
Lost users	0	537	0	0	0	0	272	0	0	0	0	76	0	0	0
Walked km	0	36	5115	0	0	0	14	3855	0	0	0	4	1318	0	0
Wall time (min.)	1	5	46	32	593	1	2	12	19	287	1	1	2	15	42

Tab. 2: Vergleich verschiedener Algorithmen: *All Single*(AS): jeder Teilnehmer fährt mit eigenem Auto. *Ridesharing*(RS): Algorithmus von [Feh24]. *DirectPath*(DP): Algorithmus von [Hol24]. *Clustering Alg.*: mein vorgeschlagener Clustering Algorithmus. *Route Enumeration Alg.*(REA): exakte Lösung des Problems, vgl. [HVHL20].

Insgesamt kann man erkennen, dass desto kleiner der Datensatz, bzw. geringer die Dichte ist, die Einsparungen durch *Carpooling* im Vergleich zum *All Single* Fall geringer werden. Beispielsweise in Datensatz 1 wird durch den REA die zurückgelegte Kilometerdistanz aller Autos von 120.103 auf 54.138 reduziert und somit um weit mehr als die Hälfte. Im kleinsten Datensatz 3, konnte die Kilometerzahl nur von 24.239km auf 13.828km verringert werden. Dieser Trend wird auch besonders beim Blick auf die durchschnittlich Auslastung einer Fahrt deutlich. Die Werte aller Algorithmen verschlechtern sich mit abnehmender Dichte. Dies ist absehbar, da weniger potenzielle Matchingpartner vorhanden sind, so ist es schwieriger die Fahrzeuge voll auszulasten.

Die exakte Lösung, die über den REA mit Preclustering errechnet wurde, konnte in allen Datensätzen überzeugen. Es wurden stets Lösungen mit den geringsten Gesamtkilometern errechnet und die höchste durchschnittliche Auslastung erzielt. Dies wirkt sich offensichtlich negativ auf die durchschnittliche Reisezeit der Nutzer aus. Sie ist vergleichsweise hoch mit mindestens 13 Minuten je nach Datensatz, was zu einem großen Teil den Stoppzeiten geschuldet ist, da für jeden Pickup eine Minute Wartezeit kalkuliert wird. Des Weiteren ist die Berechnungszeit auffällig, was für eine exakte Lösung erwartbar war. Beim größten Datensatz belief sie sich auf ca. 8 Stunden. Somit ist die Nutzung einer Heuristik durchaus gerechtfertigt, vor allem da keine der anderen Methoden über 1 Stunde brauchte. Wobei man auch sagen kann, dass die Begrenzung der Cluster auf 100 dafür zu sorgen scheint, dass die Laufzeit, ab einem gewissen Punkt, nicht exponentiell mit der Anzahl an Nutzern steigt.

Der *Direct Path* Algorithmus von [Hol24] hat sehr gute Ergebnisse bei den Gesamtkilometerzahlen und durchschnittlichen Auslastungen geliefert. Diese sind in den ersten beiden Datensätzen sogar besser als beim Clustering Algorithmus. Jedoch ist die durchschnittlich Reisezeit der Nutzer sehr hoch. Tatsächlich ist die durchschnittlich Fahrtzeit eines Nutzers pro Fahrt sehr gering. Je nach Datensatz kamen durch die zusätzlich gelaufenen Kilometer, im Schnitt ca. $4-7$ Minuten pro Nutzer dazu. Dies kann man auch daran sehen, dass die Gesamtdistanz beim Gehen sehr hoch ist und ein Nutzer je nach Datensatz im Schnitt etwa 500 Meter am Tag extra laufen müsste. Ein Ziel in der Arbeit von [Hol24] war es möglichst wenige Fahrer alleine zu lassen. Dies ist auch hier gut aufgegangen, in Datensatz 1 sind es sogar weniger Fahrten mit nur einer Person als beim REA.

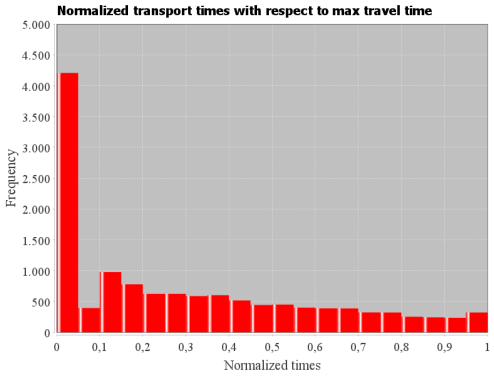
Außerdem haben wir noch den *Ridesharing* Algorithmus von [Feh24] der ebenfalls über zusätzlich Kilometer zu Fuß verfügt. Jedoch sind es deutlich weniger und auch die weiteren Zahlen können in diesem Test nicht mithalten. Dafür scheinen die Möglichkeiten nach denen hier Matches gefunden werden, zu gering zu sein. Außerdem ist es der einzige Algorithmus der dynamisch vorgeht und nicht über alle Anfragen zu Beginn verfügt. Somit ist die durchschnittliche Auslastung stets unter 2 pro Fahrt und für ca. 8% der Nutzer fand sich keine passende Rückfahrmöglichkeit.

Zuletzt betrachten wir die Ergebnisse des Clustering Algorithmus. Im kleinsten Datensatz steht zwischen ihm und dem REA nicht so viel, beispielsweise bei der durchschnittlichen Auslastung von 2,40 gegenüber 2,63. Aber in den ersten beiden Datensätzen ist der Unterschied deutlicher zu erkennen. Besonders was die Gesamtfahrtzeit der Autos angeht, herrscht eine große Diskrepanz, die anhand der Kilometer nicht so stark absehbar

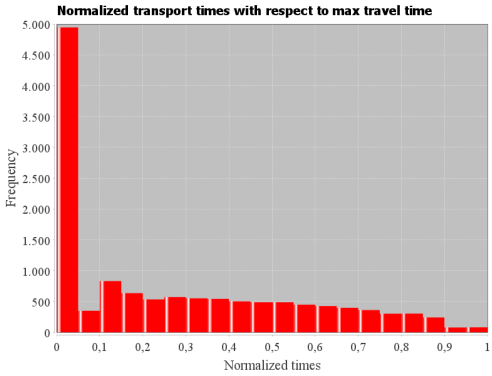
ist. Dies wird daran liegen, dass der REA auf diesen Parameter speziell optimiert hat. Das kann man auch in der durchschnittlichen Fahrtzeit sehen, die ist relativ nah beieinander, während bei der Auslastung eine signifikante Lücke klafft. Dennoch kann man insgesamt sagen, dass die Ergebnisse für eine Heuristik durchaus vielversprechend sind und besonders im Vergleich zu den Zahlen ohne *Carpooling* könnten so, mit geringem Rechenaufwand, viele Kilometer eingespart werden.

6.3 Visuelle Auswertung der Tests

In diesem Abschnitt wird mithilfe von einigen Grafiken ein genauerer Blick auf die Ergebnisse des REA und Clustering Algorithmus geworfen. Zunächst geht es um die Reisezeiten der Nutzer. Es ist fraglich, ob die angesetzte maximale Fahrtdauer eines Nutzers meist ausgereizt, oder eher selten erreicht wird. In Abbildung 1 sind die Häufigkeiten an relativen Reisezeiten dargestellt.



(a) Ergebnisse des REA in Datensatz 1



(b) Ergebnisse des Clustering Algorithmus in Datensatz 1

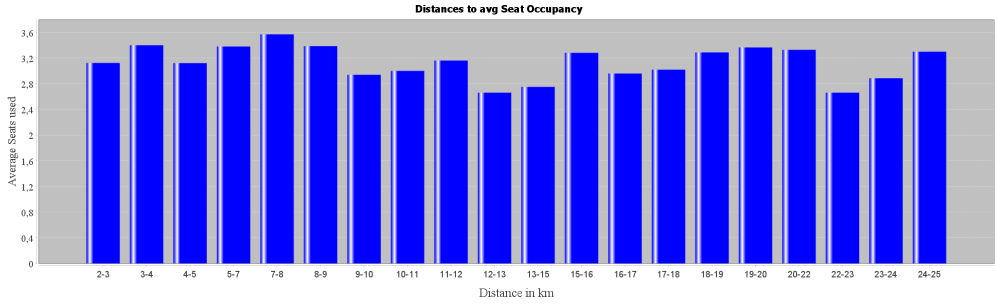
Abb. 1: Übersicht über relative durchschnittliche Reisezeiten der Nutzer pro Fahrt. Der Wertebereich wurde in 20 Töpfe zerlegt und dafür je ein Balken visualisiert.
Relative Reisezeit: $\frac{t-t_{min}}{t_{max}-t_{min}}$

Insgesamt lässt sich sagen, dass die beiden Grafiken über viele Ähnlichkeiten verfügen. Der mit Abstand größte Balken ist der erste. Sehr viele Nutzer haben keine oder nur eine sehr geringe Verzögerung durch *Carpooling*. Darunter fallen natürlich alle die weiterhin alleine fahren, aber auch die Nutzer die als letztes zu-, bzw. als erstes aussteigen. Der REA hat ca. 700 Ausschläge weniger an dieser Stelle, die sich scheinbar relativ gleichmäßig auf die restlichen Balken verteilen.

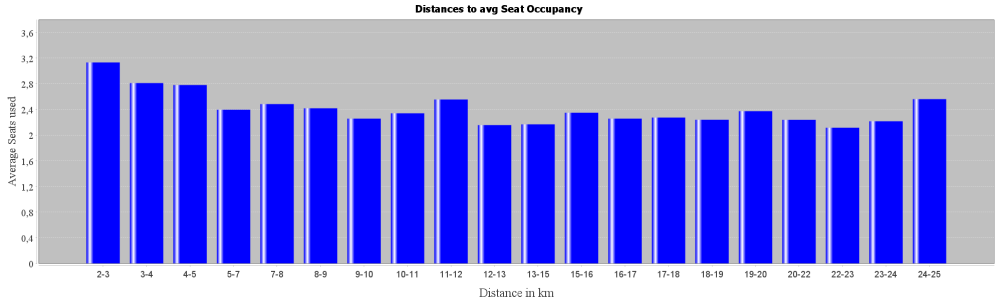
Ein weiterer Spike ist beim dritten Balken zu sehen, darunter fallen Nutzer mit einer relativen Fahrtzeit im Intervall $[\frac{2}{20}, \frac{3}{20}[$. Hier sammeln sich wohl viele Nutzer die einen Zwischenstopp miterleben und somit 1 Minute Wartezeit hinnehmen müssen. Daraufhin nehmen die Häufigkeiten langsam ab. Nur zum Ende ist noch ein Unterschied zwischen den beiden Algorithmen zu erkennen. Während beim REA die Häufigkeiten beständig bleiben und sogar noch einmal leicht ansteigen, gibt es an dieser Stelle kaum Fälle in der

Lösung des Clustering Algorithmus. Die dargestellten Trends sind auch in den Grafiken zu anderen Datensätzen erkennbar.

Des Weiteren hat sich mir die Frage gestellt, ob die Algorithmen für alle Nutzer ähnlich gut funktionieren. Oder ob möglicherweise der Wohnort eine größere Rolle spielt. Insbesondere ist es ein wichtiges Kriterium, dass ein Algorithmus nicht nur die Nutzer, die bereits nah am Zielort wohnen verstärkt miteinander matched, sondern auch die weiter entfernten unterbringt. Dafür wurden die Grafiken aus Abbildung 2 angefertigt. Sie sollen die durchschnittliche Auslastung der Fahrzeuge zeigen, abhängig von der Luftliniendistanz des Fahrers zum Zielort.



(a) Ergebnisse des REA in Datensatz 1



(b) Ergebnisse des Clustering Algorithmus in Datensatz 1

Abb. 2: Durchschnittliche Fahrzeugauslastung abhängig von der Distanz des Fahrers zum Zielort (Luftlinie)

In diesen Grafiken ist auf den ersten Blick kein klares Muster zu erkennen, jedoch gibt es ein paar Balken die auffällig sind. Offensichtlich ist die durchschnittliche Auslastung generell höher bei der Lösung des exakten Algorithmus. Besonders bei Distanzen von $5 - 9$ km und $18 - 22$ km ist es gelungen vergleichsweise viele Mitfahrer zu finden. Insgesamt zeichnet sich bei beiden Grafiken ein sehr gleichartiger Verlauf ab, also die relativen Differenzen zwischen den Balken sind in beiden Abbildungen sehr ähnlich. Möglicherweise hat dies mit der räumlichen Verteilung der Nutzer zu tun, die bei beiden identisch ist.

Nur bei den sehr nahen Nutzern ist das nicht zu erkennen. Der Clustering Algorithmus scheint sich hier deutlich leichter zu tun. Vor allem der erste Balken ist sehr hoch, aber

auch die Balken zu $3-5$ km stechen hervor gegenüber dem Rest. Ansonsten ist die Auslastung eindeutig geringer.

6.4 Tests ohne lokale Nutzer

In Abbildung 2 hat sich gezeigt, dass der Clustering Algorithmus besonders gut Nutzer matched, die nahe am Zielort wohnen. Diesen Effekt wollen wir nun genauer untersuchen und überprüfen, dass dadurch die vorherigen Ergebnisse nicht verfälscht wurden. Dafür führten wir weitere Tests auf den Datensätzen durch. Diesmal wurden alle in Würzburg lebenden Nutzer ignoriert, also jeder Student mit einer Würzburger Postleitzahl. In Tabelle 3 sind die Ergebnisse aus Datensatz 1 zu sehen. Getestet wurden nur der Clustering Algorithmus und der REA. Zum Vergleich ist noch der Fall ohne Carpooling, *All Single* zu sehen.

Dataset 1 (2478 agents)

	AS	Clust.		REA	
Total km travelled (cars)	79905	46934	−41%	36330	−54%
Total min. travelled (cars)	74924	52284	−30%	40162	−46%
Avg. Seat Count	1	2.04	+104%	2.77	+177%
Overall rides	4956	2428	−51%	1790	−63%
Rides alone	4956	1050	−78%	428	−91%
Avg. min. travelled (user)	15.12	18.02	+19%	18.96	+25%
Wall time (min.)	1	17	-	146	-

Tab. 3: Testergebnisse von REA und Clustering Algorithmus auf Datensatz 1 ohne Nutzer mit Wohnsitz in Würzburg. Zusätzlich die prozentuale Differenz der Ergebnisse gegenüber der *All Single* Herangehensweise, bei der kein *Carpooling* stattfindet.

Tatsächlich hat dieser Filterungsschritt dazu geführt, dass nur noch 2478 Agenten übrig geblieben sind. Es gab also zuvor sehr viele lokale Nutzer. Die durchschnittliche Auslastung unterschied sich zuvor um 0.54 und 0.67 bei den größeren Datensätzen. Diese Lücke ist nun weiter angestiegen und der REA ist nun um 0.73 in dieser Kategorie besser.

Ansonsten ist auch in dieser Tabelle zu erkennen, dass der Clustering Algorithmus vor allem bei der Gesamtfahrtzeit der Autos nicht ganz mithalten kann. Der REA spart hier 46% ein im Vergleich zu *All Single*. Immerhin kommt der Clustering Algorithmus auf eine Verbesserung von 30%. Die Diskrepanz ist hier am höchsten, da der REA auf diese Zahl hin optimiert.

Der Unterschied bei der durchschnittlichen Reisezeit eines Agentens ist beim Clustering Algorithmus ähnlich wie in den vorherigen Tests. Nutzer sind etwa 3 Minuten länger unterwegs pro Fahrt, als wenn sie selbst direkt fahren würden. Beim REA hingegen ist

die Differenz sogar auf 4 Minuten angestiegen und somit noch einmal höher als in Tabelle 2.

Insgesamt können sich die Ergebnisse des Clustering Algorithmus durchaus sehen lassen. Die lokalen Nutzern scheinen in den ersten Tests durch ihre kurzen Fahrtwege nicht zu sehr ins Gewicht gefallen zu sein und haben das Gesamtergebnis nicht übermäßig beeinflusst. Der heuristische Algorithmus scheint in der Lage zu sein auch mit weiter entfernten Agenten gut umzugehen.

7 Zusammenfassung

In dieser Arbeit haben wir zunächst den Forschungsstand im Bereich *Carpooling* analysiert. Dabei ist aufgefallen, dass selten eine umfassende Definition angewendet wurde, die wir in der Praxis für sinnvoll halten. Besonders eine freie Wahl der Fahrer, oder auch ein Durchwechseln der Teams zwischen Hin- und Rückfahrt war häufig nicht vorgesehen. Deshalb haben wir uns mit dem SDCTSP eine eigene Problemstellung überlegt, die stark an dem von [HVHL20] vorgestellten CTSP angelehnt ist.

Daraufhin wurde unter anderem der exakte REA von [HVHL20] vorgestellt und auf diesem Problem getestet. Obwohl dieser aufgrund des Preclustering Schritts selbst heuristisch ist, liefert er mit Abstand die besten Ergebnisse auf den Datensätzen von Studenten der Universität Würzburg. Somit ist er für den statischen Fall dieses Problems gut geeignet. Aufgrund von hohen Laufzeiten, mit bis zu 8 Stunden, ist der Algorithmus in einem dynamischeren Setting mit vielen Nutzern nicht praktikabel.

Dafür bietet sich der in dieser Arbeit vorgestellte Clustering Algorithmus an. Mit ihm wurde ein Framework für eine Heuristik erstellt, bei der die einzelnen Schritte mit verschiedenen Strategien gelöst werden können. Die Implementierung, für die wir uns entschieden haben, kam maximal auf eine Laufzeit von 32 Minuten. Auch mit diesem Algorithmus konnte eine erhebliche Einsparung von Gesamtkilometern erreicht werden, sowie eine hohe durchschnittliche Auslastung der Fahrten. Es kristallisierte sich jedoch heraus, dass der Algorithmus besonders mit lokalen Nutzern, die sehr dicht besiedelt beieinander wohnen, das Ergebnis positiv beeinflussen konnte. Mit den weiter entfernten Agenten tat er sich schwerer, im Gegensatz zum REA.

Insgesamt fiel der Unterschied beider Algorithmen in einer weniger dichten Nutzerverteilung eher gering aus. Das spricht für den Clustering Algorithmus als ressourcensparende Alternative.

8 Ausblick

Im Zuge meiner Arbeit an diesem Praktikum wurden viele Überlegungen angestellt, Algorithmen implementiert und Tests durchgeführt. Jedoch hätte es noch einige weitere Möglichkeiten gegeben, um das Projekt zu erweitern.

Zum einen hätte man den REA mit verschiedenen Optimierungszielen testen können. Es hätte sich zum Beispiel geeignet die Gesamtkilometer oder die maximale Wartezeit zu minimieren. Fraglich wäre wie stark sich diese Änderung im Endergebnis zeigt. Außerdem

wurde der REA durch den verfügbaren RAM limitiert. Mit einem besseren Rechner wären wohl auch größere Preclustering Größen mit $150 - 200$ möglich gewesen. Auch hier wäre ein Vergleich zu den angeführten Testergebnissen interessant gewesen.

Des Weiteren stellt sich die Frage, wie andere Implementierungen des Clustering Algorithmus abschneiden würden. Besonders für den zeitlichen und räumlichen Clusteringschritt sind viele andere Strategien denkbar. Generell könnte man nach dynamischen Ansätzen für dieses Problem suchen, um mit diesen Lösungen kurzfristig modifizieren zu können.

Letztlich sind auch noch weitere Tests denkbar. Im *CaMoS* Framework sind die Nutzer als Studenten modelliert und haben deshalb sehr unterschiedliche Zeiten. Es ist zu prüfen, wie die vorgestellten Algorithmen mit einem Datensatz von berufstätigen Personen umgehen. Dabei könnte es zu mehr zeitlichen Überschneidungen kommen und sich so das Ergebnis der Heuristiken verbessern.

Literatur

[AESW10] Niels Agatz, Alan Erera, Martin Savelsbergh und Xing Wang: Sustainable passenger transportation: Dynamic ride-sharing. 2010. https://ssrn.com/abstract=1568676.

[BMM04] Roberto Baldacci, Vittorio Maniezzo und Aristide Mingozzi: An exact method for the car pooling problem based on lagrangean column generation. *Operations research*, 52(3):422–439, 2004, 10.1287/opre.1030.0106.

[CdLHM04] Roberto Wolfler Calvo, Fabio de Luigi, Palle Haastrup und Vittorio Maniezzo: A distributed geographic information system for the daily car pooling problem. *Computers & Operations Research*, 31(13):2263–2278, 2004, 10.1016/S0305-0548(03)00186-2.

[Feh24] Helena Fehler: Modeling and Evaluation of Sustainable Mobility Concepts for the Campus Mobility. Master's thesis, University of Wuerzburg, 2024. https://www.informatik.uni-wuerzburg.de/fileadmin/10030100/2024/Update_-_Modeling_and_Evaluation_of_Sustainable_Mobility_Concepts_for_the_Campus_Mobility_-_Helena_Fehler.pdf.

[HJL14] Shih Chia Huang, Ming Kai Jiau und Chih Hsiang Lin: A genetic-algorithm-based approach to solve carpool service problems in cloud computing. *IEEE Transactions on intelligent transportation systems*, 16(1):352–364, 2014, 10.1109/TITS.2014.2334597.

[HJL18] Shih Chia Huang, Ming Kai Jiau und Yu Ping Liu: An ant path-oriented carpooling allocation approach to optimize the carpool service problem with time windows. *IEEE Systems Journal*, 13(1):994–1005, 2018, 10.1109/JSYST.2018.2795255.

[Hol24] Marie Holfelder: Simulation eines neuen Matching-Algorithmus´ für Studierenden-Ridesharing. Bachelor's thesis, University of Wuerzburg, 2024. Chair of Algorithms and Complexity.

[HVHL20] Mohd Hafiz Hasan, Pascal Van Hentenryck und Antoine Legrain: The commute trip-sharing problem. *Transportation Science*, 54(6):1640–1675, 2020, 10.1287/trsc.2019.0969.

[HW12] Wesam Herbawi und Michael Weber: A Genetic and Insertion Heuristic algorithm for solving the dynamic ridematching problem with time windows. 2012, 10.1145/2330163.2330219.

[KBB21] Rachid Kaleche, Zakaria Bendaoud und Karim Bouamrane: An improved biogeography-based optimization for the long-term carpooling problem. *Applied Artificial Intelligence*, 35(10):745–764, 2021, 10.1080/08839514.2021.1935586.

[MJ17] Neda Masoud und R Jayakrishnan: A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system. *Transportation Research Part B: Methodological*, 106:218–236, 2017, 10.1016/j.trb.2017.10.006.

[VMS02] Klaus Varrentrapp, Vittorio Maniezzo und Thomas Stützle: The Long Term Car Pooling Problem. 2002.