Planung des Ländlichen On-Demand-Verkehr - Probleme, Analyse und Algorithmen

Sven Löffler[1], Ilja Becker[1], Petra Hofstedt[1], André Nitze[2], Silvia Hennig[3], Alexander Klinge[2]

Abstract: On-Demand-Verkehre können helfen, Lücken im ländlichen öffentlichen Personennahverkehr (ÖPNV) zu schließen. Hohe Lizenzgebühren für proprietäre Software halten jedoch viele Kommunen von der Einführung entsprechender Verkehre ab, insbesondere in strukturschwachen Regionen, die diese besonders nötig hätten. Hinzu kommt, dass entstehende digitale Angebote meist Software-Inseln bleiben und verfügbare Routing-Algorithmen derzeit noch weitgehend ungeeignet sind, bestehendem Linienverkehr zuzuliefern und Parallelverkehre zu vermeiden. In dieser Arbeit werden Ergebnisse einer Machbarkeitsstudie von ländlichem On-Demand-Verkehr im Rahmen eines Projektes vorgestellt und diskutiert. Wir betrachten dazu die aktuelle Situation in der gewählten Modellstadt Spremberg und analysieren die Anwendbarkeit verschiedener Algorithmen zur Planung von On-Demand-Verkehr. Ziel ist eine niedrigschwellige Open Source-Lösung für ein intermodales Routing im ländlichen Raum. Diese soll in die Open-Data-Plattform bbnavi integriert werden können, um die Nutzung bestehender Mobilitätsdaten zu vereinfachen, Interoperabilität herzustellen und die Übertragbarkeit in andere Regionen zu erleichtern. In einem geplanten Folgeprojekt soll auf Basis der Machbarkeitsstudie eine entsprechende Software entwickelt und in einem Modellprojekt in Spremberg erprobt werden. Die zu entwickelnde Open Source-Lösung soll den Kostendeckungsgrad von Letzte-Meile-Verkehren verbessern und es kommunalen Akteuren ermöglichen, selbst zu Mobilitätsdienstleistern zu werden und entsprechende Geschäftsmodelle zu entwickeln.

Keywords: Open Source; ÖPNV; On-Demand-Verkehr; Vehicle Routing; Shortest Path Problem

1 Einleitung

Der öffentliche Nahverkehr ist in großen Städten wie Berlin, Hamburg oder München in der Regel sehr gut ausgebaut und hoch frequentiert, so dass ein schnelles Erreichen beliebiger Zielorte von beliebigen Ausgangsorten in der Regel sowohl kostengünstig als auch zeitnah möglich ist. Bewegt man sich allerdings weg von den großen Städten, hin zu den ländlicheren Gebieten, so lassen der generelle Streckenausbau und die Frequentierung schnell stark nach. Ein Erreichen, selbst von teilweise nahe gelegenen Zielen, ist dann in der Regel nur noch mit dem Privatverkehr oder erheblichem zeitlichen Aufwand möglich.

[1] BTU Cottbus - Senftenberg, FG Programmiersprachen und Compilerbau, Konrad-Wachsmann-Allee 5 03046 Cottbus, Deutschland {SvenLoeffler,Hofstedt,Ilja.Becker}@b-tu.de

[2] Technische Hochschule Brandenburg, Fachbereich Wirtschaft, Magdeburger Straße 50, 14770 Brandenburg an der Havel, Deutschland andre.nitze@th-brandenburg.de

[3] neuland21 e.V., Klein Glien 25, 14806 Bad Belzig, Deutschland, Geschäftsführerin Vereinsvorsitzende silvia.hennig@neuland21.de, Programmbereichsleiter Mobilität alexander.klinge@neuland21.de

Oftmals muss mittels ÖPNV eine Route über den nächst größeren Ort eingeschlagen werden um einen viel näher gelegenen angrenzenden Ort zu erreichen. Dieser, in der Regel im ländlichen Raum sternförmig ausgeprägte Verbindungsausbau, bietet damit oftmals keine ernstzunehmende Alternative zum eigenen Auto.

Einen weitaus dynamischeren Ansatz als den statischen Linienverkehr versprechen On-Demand-Verkehre. Diese könnten durch klassische Busse des Personennahverkehrs, die nicht auf festen Linien fahren sondern sich dynamisch den Bedürfnissen der Bevölkerung anpassen, realisiert werden. Vorteile dieses Vorgehens sind zum einen, dass keine hohen Anschaffungskosten entstehen und zum anderen, dass mit angepasster Streckenführung das Sternproblem umgangen werden und somit der bereits vorhandene ÖPNV unterstützt werden kann. Es entsteht daher keine Konkurrenz zu den bestehenden Anbietern, sondern eine Erweiterung und gleichzeitig eine bessere Erreichbarkeit des bestehenden Angebots.

Die in der Machbarkeitsstudie zu konzipierende Software für den ländlichen On-Demand-Verkehr in der Modellstadt Spremberg soll zusätzlich als Open Source-Software gestaltet werden. Insbesondere soll die bereits existierende Software bbnavi [23a] Verwendung finden. Die ausschließliche Nutzung von Open Source-Software soll eine leichtere Übertragbarkeit auf andere Städte und Kommunen erleichtern.

Das weitere Papier ist wie folgt aufgebaut. In Abschnitt 2 wird auf die Situation des ÖPNV und Besonderheiten der Modellstadt Spremberg als Vertreter für Städte und Gemeinden im ländlichen Bereich eingegangen. Zur Umsetzung des On-Demand-Service als Unterstützung für den ÖPNV wird ein Shortest Path-Algorithmus für die statische Planung des ÖPNV (Abschnitt 3) und ein Vehicle Routing-Algorithmus für die flexible Planung des On-Demand-Services (Abschnitt 4) benötigt. Mittels der Shortest Path-Algorithmen werden dann in der Umsetzung die schnellsten, günstigsten, bestgelegenen ÖPNV-Verbindungen ermittelt, wohingegen die Vehicle Routing-Algorithmen den eigentlichen On-Demand-Service planen und somit die optimalen Touren für die zur Verfügung stehende Fahrzeugflotte berechnen. Im Abschnitt 5 wird die Kombinierbarkeit von Shortest Path-Algorithmen mit Vehicle Routing-Algorithmen analysiert. Abschließend wird in Abschnitt 6 eine Zusammenfassung der bisherigen Arbeit und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Besonderheiten des ÖPNV im ländlichen Raum am Beispiel der Modellstadt Spremberg

Spremberg ist eine Kleinstadt im Süden von Brandenburg an der Grenze zu Sachsen. Die Stadt verfügt über 21.000 Einwohner (stand vom 31.12.2021 [21a]) auf einer Fläche von knapp über 200 km^2. Das ergibt im Schnitt etwa 105 Einwohner je km^2. Im Vergleich dazu leben in Berlin ca. 4.100, in Hamburg ca. 2.500 und in München ca. 4.800 Einwohner je km^2. Aber nicht nur hinsichtlich der Bevölkerungsdichte sondern auch im Ausbau des ÖPNV unterscheidet sich Spremberg klar von den großen Städten. So gibt es z.B. 31 Regionalverkehrsrouten (Bahn), die Berlin als Start, Ziel oder Zwischenhalt bedienen. Für

die Stadt Spremberg ist das eine, die Verbindung von Zittau nach Cottbus Hauptbahnhof (stündlich zwischen 4:30 Uhr und 0:30 Uhr) und wieder (stündlich zwischen 5:22 Uhr und 23:30 Uhr) zurück. Über eine IC-, ICE oder S-Bahn-Anbindung verfügt Spremberg nicht [21b]. Alternativ zur fehlenden Bahnanbindung existiert eine Buslinie (Plus Bus 800), welche die Städte Cottbus, Spremberg, Schwarze Pumpe und Hoyerswerda miteinander verbindet (stündlich zwischen 5:00 Uhr und 21:00 Uhr).

Aufgrund dieser zahlenmäßigen Unterschiede, sowohl in der Bevölkerungsdichte als auch im Streckenausbau und der angebotenen Bus-/Bahnfrequenz, können für ländliche Gebiete, wie die Modellstadt Spremberg, nicht einfach die gleichen Methoden zur Bewältigung des ÖPNV angewendet werden wie in großen Städten oder Ballungsgebieten. In gut ausgebauten Großstädten genügt für das Finden eines schnellsten Transportes von einem Ort zum anderen ein sogenannter Shortest Path-Algorithmus [KV12]. Dies setzt allerdings voraus, dass alle Orte in akzeptabler Zeit mit dem ÖPNV erreichbar sind.

Straßen- und Verkehrsnetze, wie die der Stadt Spremberg und Umgebung, können durch gewichtete Graphen repräsentiert werden. Ein Shortest Path-Algortihmus findet dann den kürzesten Weg zwischen einem Start ($s \in V$) und einem Zielknoten ($z \in V$). Ausgangspunkt dabei ist ein gewichteter Graph $G = (V, E)$ mit Gewichtsfunktion $d : E \to \mathbb{R}$, wobei V eine Menge von Knoten (englisch vertices) und $E \in V \times V$ eine Menge von Kanten (englisch edges) bezeichnet.

Mit Hilfe der Abstraktion des gewichteten Graphen und einem Algorithmus zum Lösen des Shortest Path-Problems kann sowohl für den privaten Straßenverkehr als auch für den ÖPNV ein kürzester Weg ermittelt werden. Für den Straßenverkehr sind die Knoten Kreuzungen und Orte die durch Straßen (Kanten) miteinander verbunden sind. Die Gewichtsfunktion ordnet dabei jeder Straße eine geschätzte Fahrtdauer zu. Analog kann für den ÖPNV ein Graph verwendet werden, bei denen die Busstationen und Bahnhöfe die Knoten darstellen und die Schienen bzw. Buslinien durch Kanten repräsentiert werden. Auf Algorithmen zum Lösen des Shortest Path-Problems wird in Abschnitt 3 eingegangen. Da in Spremberg (und auch in vielen anderen ländlichen Städten und Gemeinden) kein flächendeckendes Bus- oder Bahn-Netz existiert und damit bestimmte Knotenpunkte nicht oder nur mit großen Umwegen erreichbar sind, müssen die bestehenden ÖPNV-Verbindungen durch flexible On-Demand-Services unterstützt werden. Eine Fahrt vom Friedhof Cantdorf (Spremberg) zum nächstgelegenen Ortsteil Weskow (Spremberg, Luftlinie 3 km, Fahrtzeit Auto 9 min, Fußweg 40 min) kann gegenwärtig z.B. nur mit erheblichem Zeitaufwand mittels ÖPNV bewältigt werden (Fahrzeit ÖPNV 46 min inklusive 24 min Fußweg).

Bei einem On-Demand-Service soll ein Fahrzeug oder eine Flotte von Fahrzeugen dazu genutzt werden, die Transportwünsche möglichst vieler Teilnehmer zu erfüllen und dabei die geringsten Kosten zu verursachen. Die beschriebene Problemstellung wird in der Literatur als Tourenplanung (englisch Vehicle Routing Problem, VRP) bezeichnet. Ziel ist dabei die Gruppierung und optimale Aneinanderreihung verschiedener Transportaufträge [TV02]. Aus den Gegebenheiten in Spremberg leiten sich weitere Einordnungen unseres Problems

in Unterklassen des VRP [Zh22] ab. So handelt es sich um ein Capacitated Vehicle Routing Problem (CVRP), d.h. es müssen also die verschiedenen Kapazitäten der Fahrzeuge (Auto, Kleinbus, Bus) berücksichtigt werden. Es genügt auch nicht, die Transporte einfach nur durchzuführen, sondern dies muss auch innerhalb zulässiger Zeitfenster geschehen (Vehicle Routing Problem with Time Window, VRPTW). Diese Zeitfenster steuern auf der einen Seite, dass die Fahrgäste ihren Anschlusszug rechtzeitig erreichen können und auf der anderen Seite, dass die Gesamtdauer des Transports für den jeweiligen Gast nicht zu groß wird (dann könnte auch der bisherige ÖPNV genutzt werden). Um eine kurzfristige Buchung des On-Demand-Services zu ermöglichen, müssen im späteren Verlauf dynamisch weitere Fahrgäste und Transporte zu bestehenden Routen hinzugefügt werden können (Dynamic Vehicle Routing Problem, DVRP). Auf Lösungsansätze für das VRP und seine Varianten wird in Abschnitt 4 eingegangen.

Für die Unterstützung des bestehenden ÖPNV mittels eines On-Demand-Services ist es nicht nur von Nöten, geeignete einzelne Verfahren zum Finden kürzester Wege und optimierter Routen zu finden, sondern auch diese geeignet zu kombinieren. So besteht zum Beispiel die Möglichkeit von Spremberg nach Bauzen mittels Bussen (Linie 800 und 500) oder Zügen (RB 65 und RB60) zu gelangen. Dem Fahrgast ist dabei die Art der Verbindung egal, was bedeutet, dass er entweder zu einer der Bushaltestellen oder zum Bahnhof gebracht werden muss, ehe er mit dem ÖPNV weiterfahren kann. Die Tourenplanung (im Folgenden auch "Routing") soll dabei gewährleisten, dass der Fahrgast so schnell wie möglich und rechtzeitig zu einem der Haltepunkte gebracht wird, wohingegen ein Shortest Path-Algorithmus zunächst die infrage kommenden Haltepunkte ermitteln muss. Wie die beiden Verfahren kombiniert werden können, wird in Abschnitt 5 diskutiert.

3 Shortest Path-Algorithmen

Das Shortest Path-Problem ist ein sehr gut erforschtes und in verschiedene Unterkategorien differenzierbares Problem. In [Ma17] sind diverse Unterkategorien des Shortest Path-Problems aufgeführt. Die Problemstellung des statischen Ermittelns der schnellsten Bus- oder Bahnverbindung fällt in die Kategorie (static) Single Source Shortest Path (SSSP). In diesem Abschnitt werden verschieden Shortest Path-Algorithmen zur Lösung des SSSP-Problems vorgestellt und miteinander verglichen.

Der Dijkstra-Algorithmus Der Dijkstra-Algorithmus [Co01] ist ein bekannter Algorithmus zur Berechnung des kürzesten Pfades in einem gewichteten Graphen mit nicht-negativen Kantengewichten. Der Algorithmus arbeitet inkrementell und wählt iterativ den Knoten mit dem geringsten Abstand vom Startknoten aus. Wurde der Knoten mit dem geringsten Abstand ausgewählt, so werden die Abstände aller Knoten, die noch nicht besucht wurden und mit dem aktuellen Knoten verbunden sind, angepasst und der aktuelle Knoten als besucht markiert. Der Abstand eines markierten Knotens kann sich nicht mehr verringern

und ist somit der kürzeste Abstand vom Startknoten zu diesem Knoten. Wird der Abstand zu einem bestimmten Knoten gesucht, so bricht der Algorithmus ab, sobald der gewünschte Knoten besucht wurde.

Der Dijkstra-Algorithmus ist exakt und hat eine Zeitkomplexität von $O(n^2)$, wobei n die Anzahl der Knoten ist. Ein Vorteil des Algorithmus ist, dass er nicht alle Kanten untersuchen muss. Dies ist besonders nützlich, wenn die Gewichte an einigen Kanten teuer sind. Ein Nachteil besteht darin, dass der Algorithmus nur mit nicht-negativ gewichteten Kanten umgehen kann und er nur für statische Graphen anwendbar ist.

Fredman und Tarjan [FT87] verbesserten den Algorithmus von Dijkstra, indem sie einen Fibonnaci-Heap (F-Heap) verwenden. Diese Implementierung erreicht eine Laufzeit von $O(n * log(n) + m)$ (n = Anzahl Knoten, m = Anzahl Kanten), da die gesamte anfallende Zeit für die Heap-Operationen $O(n * log(n) + m)$ ist und die anderen Operationen $O(n + m)$ kosten. Fredman und Willard [FW90; FW93; FW94] konnten eine Erweiterung unter Verwendung von AF-Heaps (allocation free Heaps) entwickeln, die lediglich einen Aufwand von $O(m + n * log(n)/log(log(n)))$ hat. Der AF-Heap liefert konstante amortisierte Kosten für die meisten Heap-Operationen und $O(log(n)/log(log(n)))$ amortisierte Kosten für das Löschen.

Der Dijkstra-Algorithmus kann sowohl vom Startknoten als auch vom Endknoten aus gestartet werden. Der *bidirektionale Dijkstra*-Algorithmus [Dr69] verbindet beide Ansätze und führt jeweils einen Schritt vorwärtsgehend vom Startknoten und rückwärtsgehend vom Zielknoten aus. Sobald die beiden Vorgehen einen Knoten als besucht von beiden Richtungen ermittelt haben, wurde ein kürzester Pfad zwischen Start- und Zielknoten ermittelt. Ein Vorteil des bidirektionalen Vorgehens ist, dass der Suchraum dadurch verkleinert werden kann.

Der A*-Algorithmus Der A*-Algorithmus [HNR68] ist eine Verallgemeinerung des Dijkstra-Algorithmus, der diesen um eine Kostenschätzfunktion erweitert. Eine gute Kostenschätzfunktion kann die Menge der Knoten, die untersucht werden müssen, bevor die Lösung gefunden wird und somit den Suchraum, signifikant verkleinern. Es wird dabei in jedem Schritt der Knoten als nächster ausgewählt, dessen geschätzte Kosten zum Zielknoten zusammen mit den bisherigen Kosten am geringsten sind. Wenn die verwendete Kostenschätzfunktion die tatsächlichen Kosten nie überschätzt, so ist der A*-Algorithmus optimal, das heißt er findet immer die Lösung mit den niedrigsten Kosten.

Der A*-Algorithmus hat analog zur Breitensuche das Problem, dass viele Knoten gespeichert werden müssen und die Knoten darüber hinaus auch noch nach ihrem Kostenschätzwert sortiert sein müssen. Um diese Probleme zu reduzieren, kann innerhalb der A*-Suche das Vorgehen des iterativen Vertiefens (iterative deepening) angewendet werden. Das iterative Vertiefen entspricht einer Tiefensuche mit festgelegter Tiefenschranke n. Findet die Tiefensuche keine Lösung bis Schranke n, so wird eine neue Tiefensuche mit höherem

n-Wert gestartet. Das Vorgehen des iterativen Vertiefens ist sowohl vollständig als auch optimal und benötigt nur die Rechenzeit der Breitensuche (b^d) und den Speicherbedarf der Tiefensuche ($b * d$, b = Verzweigungsgrad, d = durchsuchte Tiefe des Baumes).

Der Arcflag-Algorithmus Der *Arcflag*-Algorithmus [Hi06] ist eine zielgerichtete Beschleunigungstechnik für den Dijkstra-Algorithmus zur Suche des kürzesten Pfades zwischen zwei Knoten in einem gewichteten Graphen. Die Grundidee besteht darin, die Menge der zu betrachtenden Kanten geschickt auf einen Bruchteil im Vergleich derer zu verringern, welche bei Ausführung des Dijkstra-Algorithmus betrachtet werden müssten. Dabei wird zunächst jede Kante des Graphen um Flaggeninformationen angereichert, welche schließlich bei der Pfadsuche entscheiden, ob diese Kante für die Suche in Betracht gezogen werden muss. Die Vorberechnungsphase des Arcflags-Algorithmus verläuft in zwei Schritten. Zuerst wird das Straßennetz in Regionen eingeteilt. Anschließend wird für jede Region bestimmt, über welche Kanten kürzeste Wege in diese Region führen. Die Information, ob eine Kante Teil eines kürzesten Pfades ist, wird Arcflag genannt.

In der Anfragephase bestimmt der Algorithmus zunächst die Region des Zielknotens. Anschließend wendet er den Dijkstra-Algorithmus an, folgt dabei aber nur den Kanten, die laut Zusatzinformationen in die Zielregion führen. Er lässt also gezielt Kanten aus, die nichts mit der Zielanfrage zu tun haben. Die Komplexität des Dijkstra-Algorithmus kann durch das Verwenden von Arcflaqs nicht reduziert werden, nichtsdestotrotz verringert sich in der Praxis der Suchraum durch das Weglassen von Kanten, die keine kürzesten Wege zur Zielregion darstellen, signifikant.

Der ALT-Algorithmus ALT steht für *A* Search*, *Landmarks* und *Triangle Inequality*, da dies die Hauptbestandteile des Algorithmus sind. Der ALT-Algorithmus verwendet als Kostenschätzfunktion für den A*-Algorithmus sogenannte Landmarken [GH05]. Landmarken sind eine Teilmenge der Knoten des Graphen, für die die Distanzen aller anderen Knoten zu diesen Landmarken aus der Vorverarbeitungsphase bekannt sind. In der Folge können Entfernungsschätzungen innerhalb des Graphen mittels der Dreiecksungleichung durchgeführt werden. Die Dreiecksungleichung besagt, dass sich für beliebige drei Knoten u, v, l im Graphen der minimale Abstand von u und v kleiner oder gleich dem Abstand von u zu l plus dem Abstand von l zu v sein muss: $d(u, v) \leq d(u, l) + d(l, v)$. Aus dieser Ungleichung lassen sich Grenzen ableiten: Wie in Abbildung 1 dargestellt, ist der Abstand von u nach v kleiner als $dist(l_1, v) - dist(l_1, u)$ sowie $dist(u, l_2) - dist(v, l_2)$. Eine Frage die verbleibt ist: Wie müssen geeignete Orientierungspunkte aus dem Eingabegraphen ausgewählt werden?

Zusammenfassung Shortest Path-Algorithmen. Zusammenfassend lässt sich sagen, dass das Shortest Path-Problem für den statischen ÖPNV mittels Dijkstra-Algorithmus bzw. Weiterentwicklungen davon lösen lässt. Die Eingangsbedingung, dass keine negativen

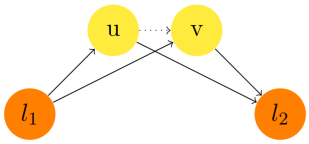

Abb. 1: Eine Visualisierung der Dreiecksungleichung (Triangle Inequality).

Kantengewichte vorliegen, ist sowohl für das Berechnen kürzester Wege als auch für das Berechnen preisgünstigster Wege erfüllt. Nach einer Studie von [Ba22] ist bereits der Dijkstra-Algorithmus in seiner Grundform in der Lage europaweite SSSP-Anfragen in ca. 6s zu beantworten. Die vorgestellten weiterentwickelten Formen benötigten dafür teilweise sogar nur wenige Millisekunden. Eine Anfrage für ländliche Regionen mit deren abgehenden und eingehenden ÖPNV-Verbindungen sollte somit mit allen vorgestellten Algorithmen realisierbar sein.

4 Vehicle Routing-Algorithmen

Ebenso wie das Shortest Path-Problem, ist auch das Vehicle Routing-Problem sehr gut erforscht. Es existiert eine Vielzahl an Algorithmen, die das Problem entweder zeitaufwändig exakt oder schnell heuristisch lösen. Abbildung 2 klassifiziert existierende Lösungsverfahren für Vehicle Routing-Probleme nach Korrektheit und heuristischem Vorgehen. An dieser Stelle soll aufgrund der großen Vielfalt an existierenden Algorithmen nur auf je ein vielversprechendes exaktes und ein heuristisches Verfahren eingegangen werden, die verwendet werden können um das VRP für den On-Demand-Service zu lösen.

4.1 Ein Constraint-Ansatz (Branch and bound)

Die Constraint-Programmierung ist ein mächtiges Werkzeug zum Modellieren und Lösen komplexer Probleme [MS98]. Das generelle Vorgehen bei der Constraint-Programmierung unterteilt sich in zwei Teile: 1. die deklarative Modellierung des Problems als Constraint-Modell, 2. das selbstständige Lösen des Constraint-Modells durch einen Constraint-Solver. Der Constraint-Anwender ist dabei nur dafür verantwortlich, das Constraint-Modell zu erzeugen und den Solver zu konfigurieren. Der Solver selbst fungiert als Blackbox und reduziert schrittweise durch Propagation (Branch and Bound) und einer Backtracking-basierten Tiefensuche den Lösungsraum solange, bis eine (optimale) Lösung gefunden wurde (falls eine solche existiert).

Für das sich aus unserer Anwendung ergebene VRP inklusive der spezifizierten Subklassen Capacitated Vehicle Routing Problem (CVRP) und Vehicle Routing Problem with Time Window (VRPTW) sind in [Zh22] bereits mathematische Beschreibungen angegeben, die sich prinzipiell direkt in Constraints überführen lassen. Einziger Stolperstein, der bei der

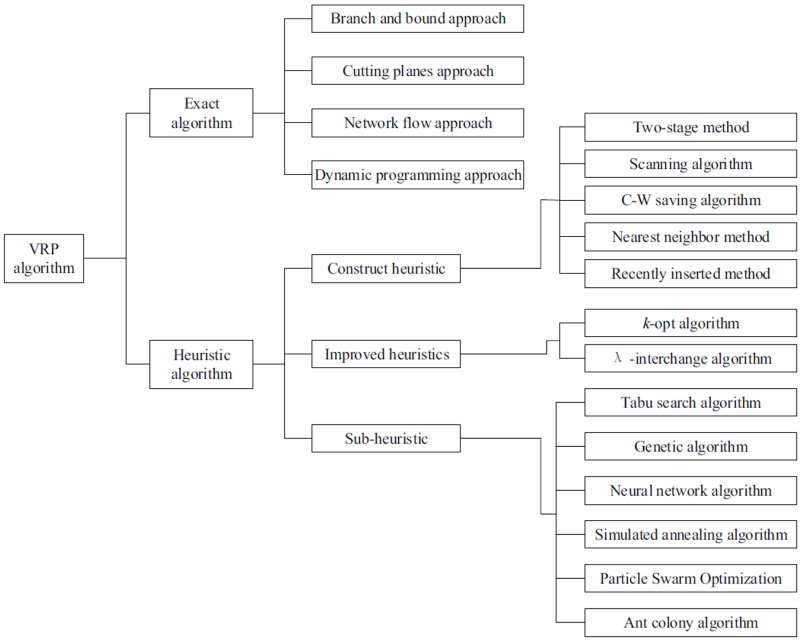

Abb. 2: Existierende Algorithmen zum Lösen von Vehicle Routing-Problemen [Zh22].

Verwendung der Constraint-Programmierung, wie bei jedem anderen exakten Verfahren auch, auftreten kann, ist dass das Problem und damit verbunden dessen Suchraum zu groß wird, als dass in akzeptabler Zeit eine gute oder sogar eine optimale Lösung gefunden werden kann. Da hier allerdings Graphen betrachtet werden, die den ländlichen Raum widerspiegeln (und keine Großstädte), sollten deren Ausmaße handhabbar sein. Sollte die Problemstellung wieder Erwarten trotzdem zu groß sein, um einen Constraint-Solver anzuwenden, so wird auf ein heuristisches Verfahren umgestellt.

4.2 Ein Zwei-Phasen-Ansatz (Two-stage method)

Der Zwei-Phasen-Ansatz probiert in Phase 1 eine initiale Lösung für das Routing-Problem zu finden und in Phase 2 die bestehende Lösung sukzessive zu verbessern [Wa14].

Die erste Phase wird dabei oftmals erneut in zwei Phasen unterteilt. In Phase 1.1 wird dabei zunächst ein Clustering $C_1, ..., C_n$ der abzufahrenden Knotenpunkte angelegt. In Phase 1.2 wird dann eine initiale Tour innerhalb jedes Clusters von einem Startknoten (Depot) hin über alle Knoten und wieder zum Startknoten zurück angelegt.

Mittels verschiedener lokaler Optimierungen wird in Phase 2 dann versucht die initiale Lösung zu verbessern. Mögliche Optimierungsoperationen sind dabei z.B. *Relocate*-, die

Relocate Split, die *2-opt* Exchange* und die *Swap-Move*-Operation. Bei der *Relocate*-Operation wird die Position eines Knoten innerhalb einer Route geändert. Ein Knoten wird also von seiner aktuellen Position innerhalb der Route zu einer anderen Position in der Route verschoben. Bei der *Relocate Split*-Operation wird ein Knoten von einer Route in eine andere Route verschoben. Die *2-opt* Exchange*-Operation verändert die Reihenfolge von zwei Knoten innerhalb einer Route. Es werden dabei zunächst zwei Knoten ausgewählt, und dann wird die Route zwischen diesen beiden Knoten umgekehrt. Die *Swap-Move*-Operation tauscht die Positionen zweier Knoten innerhalb einer Route.

Im Gegensatz zu den exakten Verfahren handelt es sich bei diesem Vorgehen um eine lokale Suche, die ein lokales Optimum findet. Ob das gefundene lokale Optimum nah am globalen Optimum ist oder dieses sogar erreicht, kann an dieser Stelle nicht garantiert werden.

5 Kombinationsmöglichkeiten

Weil der On-Demand-Service als Zubringer für den weiteren ÖPNV dienen soll, müssen die beiden vorgestellten Problemstellungen SSSP und VRP kombiniert werden. Ein Fahrgast möchte zum Beispiel von Spremberg nach Schwarze Pumpe fahren. Ihm ist es dabei egal, ob er mit dem On-Demand-Service direkt von Zuhause zum Zielort gebracht wird, oder ob er mit dem On-Demand-Service zu einer der regulären Haltestellen der entsprechenden Verbindung innerhalb von Spremberg gebracht wird und von dort aus mit dem ÖPNV weiterfahren kann. Für den Fahrgast ist dabei in der Regel nur die benötigte Zeit von Start bis Ziel interessant. Für die Routenberechnung macht es allerdings einen erheblichen Unterschied, zu welchen Bahnhof der Passagier gebracht werden muss. Befinden sich mehrere solcher Fahrgäste, mit flexiblen Zielen, gleichzeitig in einer Tour, so erschwert das die Tourenplanung zusätzlich erheblich. Nachfolgend werden zwei Vorgehen erläutert, mit solchen Alternativen umzugehen.

- **Single-SSP-VRP**: Es wird zunächst der statische ÖPNV (SSSP) geplant und im Anschluss daran der On-Demand-Service (VRP).
- **Alternative-VRP**: Das VRP wird um Alternativen in der Routenplanung erweitert. Es ergeben sich somit für jeden Fahrgast verschiedene mögliche Start- und Zielorte und -Zeiten.

Single-SSP-VRP In diesem Fall würden zunächst die ÖPNV-Verbindungen der einzelnen Fahrgäste ermittelt werden. Diese würden mit den vorgestellten Algorithmen (siehe Abschnitt 3) berechnet werden. Dabei würde als Start und Ziel jeweils die für den einzelnen Reisenden passendste Haltestelle mit zugehöriger Start- und Ankunftszeit gewählt werden. Dieses Vorgehen würde für alle Reisenden analog erfolgen, so dass alle eine minimale ÖPNV-Reisedauer haben. Im Anschluss daran wird der On-Demand-Service für die zuvor ermittelten Start- und Ankunftsorte und -Zeiten ermittelt. Bei diesem Vorgehen handelt es sich um einen

greedy-Algorithmus, bei dem versucht wird, die lokalen Optima (die ÖPNV-Verbindungen für die einzelnen Fahrgäste) zu einer Gesamtlösung (durch den On-Demand-Service) zu vereinen. Positiv daran ist, dass das VRP dabei so klein wie möglich gehalten wird und auch die einzelnen ÖPNV-Verbindungen nur einmal für jeden Fahrgast mittels SSSP-Algorithmen gelöst werden müssen. Dieses Vorgehen kann sehr schnell eine erste, gute Lösung ermitteln, allerdings kann nicht sichergestellt werden, dass ein globales Optimum gefunden wird. Denkbar ist z.B., dass durch das Ändern des Zielortes einer oder mehrerer ÖPNV-Anbindungen (Wahl eine früheren oder späteren Haltestelle) die Tourenplanung ggf. deutlich bessere Ergebnisse erlaubt.

Eine Erweiterungsmöglichkeit besteht darin, zunächst alle ÖPNV-Anbindungen, die für die einzelnen Fahrgäste infrage kommen, zu berechnen. Das beinhaltet also nicht nur die Haltestelle, die am nächsten am Start- bzw. Zielort ist, sondern auch die, die sich in einer akzeptablen Umgebung für den On-Demand-Service befinden. Anschließend wird für jeden Fahrgast eine Verbindung ausgewählt und das Routing durchgeführt. Nachfolgend wird eine andere Kombination an Fahrgastverbindungen ausgesucht und dafür ein Routing durchgeführt. Sobald ersichtlich ist, dass ein VRP keine bessere Lösung finden kann, kann die Berechnung abgebrochen und die nächste Kombination ausprobiert werden. Mit diesem Vorgehen können globale Optima ermittelt werden, allerdings steigt der Rechenaufwand exponentiell an.

Alternative-VRP Für jeden Fahrgast werden zunächst alle infrage kommenden ÖPNV-Anbindungen ermittelt (SSSP). Anders als zuvor wird nun das VRP nicht auf einer möglichen Kombination dieser ÖPNV-Anbindungen aufgerufen sondern um die möglichen Alternativen erweitert. Das VRP muss dann entweder dafür sorgen, dass Fahrgast F_1 rechtzeitig von Ort *A* zu Ort *B* oder zu Ort *C* gebracht wird. Es ergeben sich also Alternativen für die Start- bzw. Zielorte und -Zeiten. Bei der Constraint-Programmierung können solche Alternativen durch logische Meta-Constraints (*and*, *or* oder *not* die jeweils wieder Constraints als Eingaben erhalten) problemlos eingebaut werden. Auch hier wächst das Problem mit zunehmender Anzahl an Alternativen exponentiell, allerdings können durch den branch and bound-Ansatz deutlich eher solche ÖPNV-Anbindungen ermittelt und ausgeschlossen werden, die in keiner guten Lösung vorkommen können. Im Vergleich zum zuerst vorgestellten Vorgehen Single-SSP-VRP besteht hierbei die Gefahr, dass das Finden einer ersten Lösung länger dauert. Da wir uns aber im ländlichen Raum mit wenigen Alternativen im ÖPNV befinden, scheint eine Umsetzung mittels Constraint-Programmierung realistisch.

6 Zusammenfassung und Ausblick

In diesem Papier wurde das Problem des On-Demand-Verkehrs als Unterstützung für den ÖPNV im ländlichen Bereich diskutiert. Zunächst wurden die Besonderheiten des ländlichen ÖPNV und speziell der Modellstadt Spremberg erörtert, ehe sowohl Verfahren zum Finden

kürzester Wege als auch zur Routenplanung diskutiert wurden. Abschließend wurden zwei Möglichkeiten zur Kombination aus statischer ÖPNV-Planung (SSSP) und flexibler Planung für den On-Demand-Service (VRP) erörtert und Vor- und Nachteile der Verfahren diskutiert.

Im Zukunft müssen die vorgestellten Verfahren für die gegebene Datenlage der Modellstadt Spremberg umgesetzt und praktisch erprobt werden. Wir gehen an dieser Stelle davon aus, dass wir mindestens in der Lage sind mit heuristischen Verfahren eine gute, wenn nicht sogar mit einem exakten Verfahren eine optimale Lösung zu finden.

Danksagung: Das vorgestellte OSLO-Projekt wurde mit Mitteln des „mFUND - Unsere Förderung für die Mobilität der Zukunft“ [23b] (Förderkennzeichen: 19FS1005C) des Bundesministeriums für Digitales und Verkehr gefördert.

Literatur

[21a] Bevölkerungsentwicklung und Flächen der kreisfreien Städte, Landkreise und Gemeinden im Land Brandenburg 2021, Amt für Statistik Berlin-Brandenburg. Letzmals besucht am 2. Mai 2023, 2021, URL: https://download.statistik-berlin-brandenburg.de/0449b1ed6918ad8c/25590f3b9567/SB_A01-04-00_2021j01_BB.pdf.

[21b] Linienverzeichnis, Seh-Netz e.V. 2000 — 2023 ÖPNV-Info — Mobilitätsportal für behinderte Reisende. Letzmals besucht am 2. Mai 2023, 2021, URL: https://www.oepnv-info.de/linienverzeichnis.

[23a] bbnavi, DigitalAgentur Brandenburg GmbH, Gefördert durch das Ministerium für Wirtschaft, Arbeit und Energie des Landes Brandenburg. Letzmals besucht am 2. Mai 2023, 2023, URL: https://bbnavi.de/.

[23b] mFUND – Unsere Förderung für die Mobilität der Zukunft, Bundesministerium für Digitales und Verkehr. Letzmals besucht am 8. Mai 2023, 23, URL: https://bmdv.bund.de/DE/Themen/Digitales/mFund/Ueberblick/ueberblick.html.

[Ba22] Bayer, S.; Kluge, R.; Kohl, A.; Messmer, S.: Studentisches Projekt: Gläserner Routenplaner, Betreuer: M. Kobitzsch, S. Meinert, I. Rutter, Prof. P. Sanders, Prof. D. Wagner. KIT - Karlsruhe Institute of Technology. Letzmals besucht am 5. Mai 2023, 2022, URL: https://i11www.iti.kit.edu/_media/projects/rpkit/techniken_poster.pdf.

[Co01] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C.: Introduction to Algorithms. The MIT Press, 2001, ISBN: 0262032937.

[Dr69] Dreyfus, S. E.: An Appraisal of Some Shortest-Path Algorithms. Oper. Res. 17/3, S. 395–412, 1969, URL: https://doi.org/10.1287/opre.17.3.395.

[FT87] Fredman, M. L.; Tarjan, R. E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34/3, S. 596–615, 1987, URL: https://doi.org/10.1145/28869.28874.

[FW90] Fredman, M. L.; Willard, D. E.: BLASTING through the Information Theoretic Barrier with FUSION TREES. In (Ortiz, H., Hrsg.): Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. ACM, S. 1–7, 1990, URL: https://doi.org/10.1145/100216.100217.

[FW93] Fredman, M. L.; Willard, D. E.: Surpassing the Information Theoretic Bound with Fusion Trees. J. Comput. Syst. Sci. 47/3, S. 424–436, 1993, URL: https://doi.org/10.1016/0022-0000(93)90040-4.

[FW94] Fredman, M. L.; Willard, D. E.: Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths. J. Comput. Syst. Sci. 48/3, S. 533–551, 1994, URL: https://doi.org/10.1016/S0022-0000(05)80064-9.

[GH05] Goldberg, A. V.; Harrelson, C.: Computing the shortest path: *A** search meets graph theory. In: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005. SIAM, S. 156–165, 2005, URL: http://dl.acm.org/citation.cfm?id=1070432.1070455.

[Hi06] Hilger, M.; Köhler, E.; Möhring, R. H.; Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In (Demetrescu, C.; Goldberg, A. V.; Johnson, D. S., Hrsg.): The Shortest Path Problem, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, November, 2006. Bd. 74. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS/AMS, S. 41–72, 2006, URL: https://doi.org/10.1090/dimacs/074/03.

[HNR68] Hart, P. E.; Nilsson, N. J.; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans. Syst. Sci. Cybern. 4/2, S. 100–107, 1968, URL: https://doi.org/10.1109/TSSC.1968.300136.

[KV12] Korte, B.; Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Springer Publishing Company, Incorporated, 2012, ISBN: 3642244874.

[Ma17] Madkour, A.; Aref, W. G.; Rehman, F. U.; Rahman, M. A.; Basalamah, S. M.: A Survey of Shortest-Path Algorithms. CoRR abs/1705.02044/, 2017, arXiv: 1705.02044, URL: http://arxiv.org/abs/1705.02044.

[MS98] Marriott, K.; Stuckey, P. J.: Programming with Constraints - An Introduction. MIT Press, Cambridge, 1998, ISBN: 978-0-262-13341-8.

[TV02] Toth, P.; Vigo, D.: The Vehicle Routing Problem. Society for Industrial und Applied Mathematics, 2002, ISBN: 0-89871-579-2.

[Wa14] Wang, Y.; Ma, X.; Lao, Y.; Yu, H.; Liu, Y.: A two-stage heuristic method for vehicle routing problem with split deliveries and pickups. J. Zhejiang Univ. Sci. C 15/3, S. 200–210, 2014, URL: https://doi.org/10.1631/jzus.C1300177.

[Zh22] Zhang, H.; Ge, H.; Yang, J.; Tong, Y.: Review of Vehicle Routing Problems: Models, Classification and Solving Algorithms. Arch Computat Methods Eng 29/, S. 195–221, 2022.