TECHNISCHE HOCHSCHULE LÜBECK

Praktikum Intelligente Systeme

Routenplanung für Elektroautos

Blend Salihu & Igor Greszta
So, 03.12.2023

In diesem Dokument befindet sich die eigenständige Erarbeitung des Berichts von der Aufgabe "Routenplanung für Elektroautos "

Inhaltsverzeichnis

Bearbeitung der Aufgaben

Aufgabe 1

Unsere Implementierung des A*-Algorithmus ist darauf ausgerichtet, den optimalen Pfad zwischen Städten zu finden, wobei sowohl die Entfernungen zwischen den Städten als auch mögliche Ladevorgänge an Ladestationen berücksichtigt werden. Die Logik der Zustände des Suchraums werden durch die einzelnen Knoten (Node) übernommen, wobei jeder Zustand Informationen über die Stadt, ihre Verbindungen zu anderen Städten und den aktuellen Ladezustand eines Fahrzeugs enthält.

Die Implementierung verwendet eine PriorityQueue für die offenen Knoten, wobei die Priorität durch die bisherigen Pfadkosten + der derzeitige heuristische Wert * der Ladezustand bestimmt wird. Eine Map wird genutzt, um bereits besuchte Knoten zu speichern. Die Suche startet beim angegebenen Startknoten und durchläuft die Schleife, solange die offene Liste nicht leer ist. In jedem Schleifenzyklus wird der Knoten mit den geringsten Gesamtkosten ausgewählt. Wenn dieser Knoten das Ziel ist, erfolgt die Rekonstruktion des gefundenen Pfads und die Rückgabe.

Während des Suchprozesses erfolgt eine dynamische Aktualisierung des Ladens unter Berücksichtigung möglicher Ladevorgänge an Ladestationen. Ein Mechanismus entscheidet darüber, ob das Fahrzeug aufgeladen werden soll oder nicht, basierend auf dem aktuellen Ladezustand im Vergleich zu den geschätzten Kosten bis zum Ziel. Dabei ist der Agent bereit, extra Schritte zu gehen, um eine Ladestation zu erreichen.

Die Struktur des Codes besteht aus fünf Klassen: AStarSearch, City, Helper, Connection und Node. Die Klasse AStarSearch orchestriert den A*-Algorithmus und enthält die Logik für die Suche und Pfadrekonstruktion. Die Klasse City repräsentiert Städte im Graphen und enthält Informationen wie Name, Heuristikwert, Kosten, geografische Koordinaten und Verbindungen zu anderen Städten. Die Klasse Helper bietet Hilfsfunktionen zum Einlesen von Städten, Verbindungen und Testfällen aus Dateien. Die Klasse Node repräsentiert einen Zustand im Suchraum und enthält eine Stadt sowie die bisherigen Kosten und den

Ladezustand.

Zusammenfassend wird bei der Definition der Zustände des Suchraums auf die Information über Städte, Verbindungen und den Ladezustand eines Fahrzeugs geachtet. Bei der Expansion eines Zustands liegt der Fokus auf der Berücksichtigung von Verbindungskosten, Ladevorgängen, der Aktualisierung des Ladezustands und einer Entscheidung, ob das Fahrzeug aufgeladen werden sollte.

Aufgabe 2

In Aufgabe 2 haben wir zunächst aus Verständnisgründen sämtliche Testfälle grafisch dokumentiert, um einen besseren Überblick zu gewinnen:

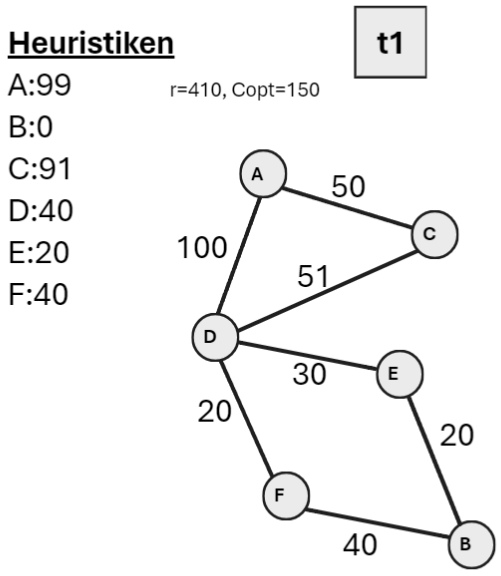

Abbildung 1: Test case 1

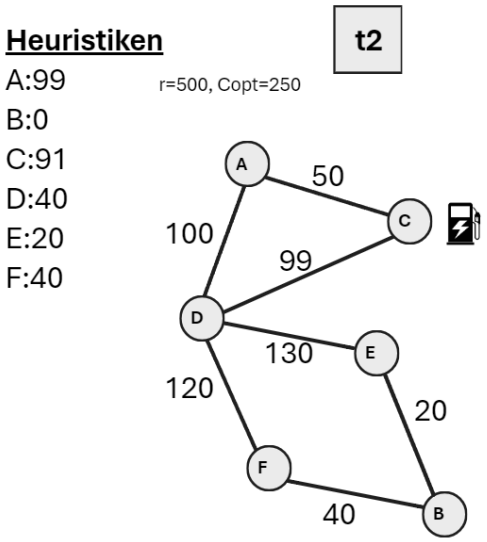

Abbildung 2: Test case 2

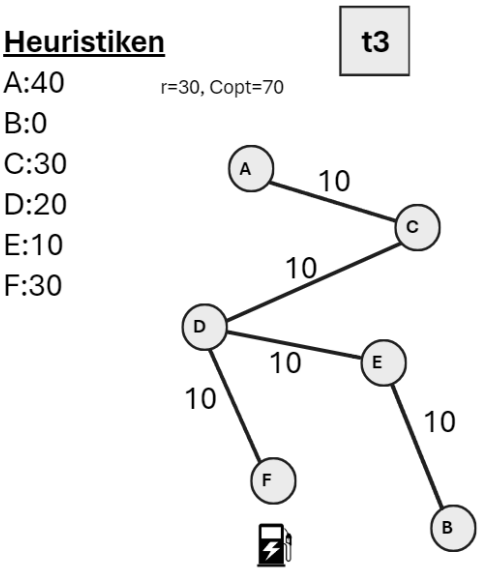

Abbildung 3: Test case 3

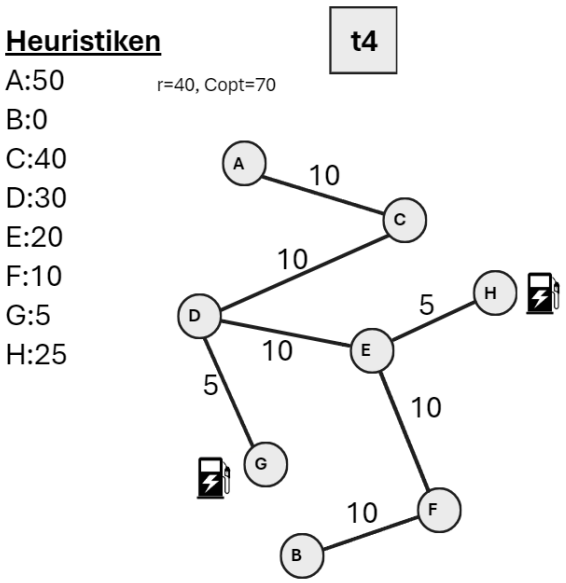

Abbildung 4: Test case 4

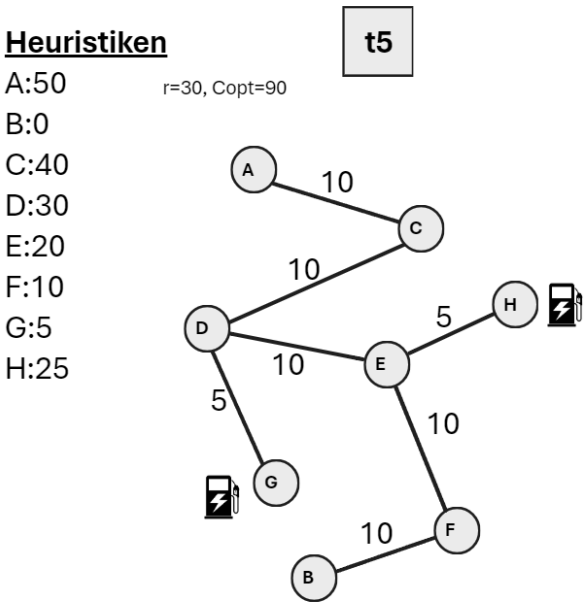

Abbildung 5: Test case 5

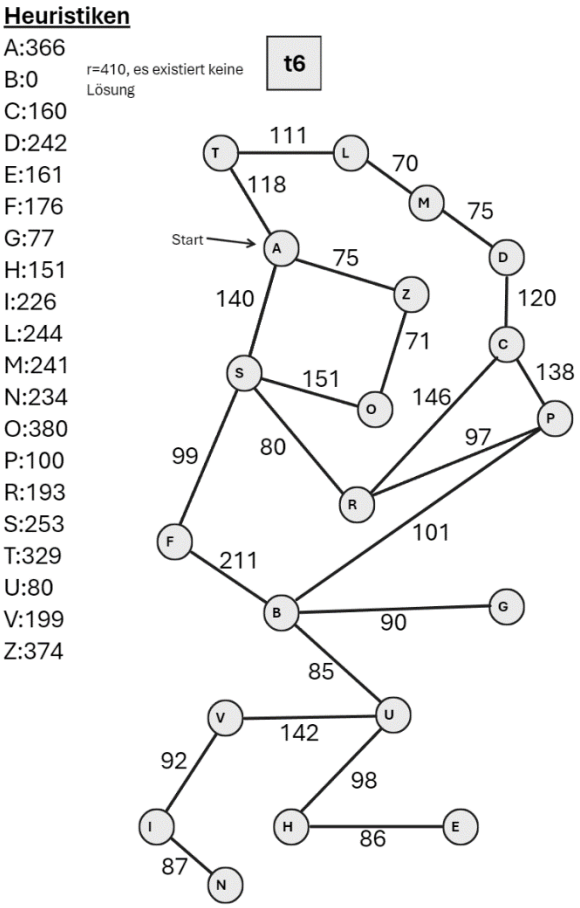

Abbildung 6: Test case 6

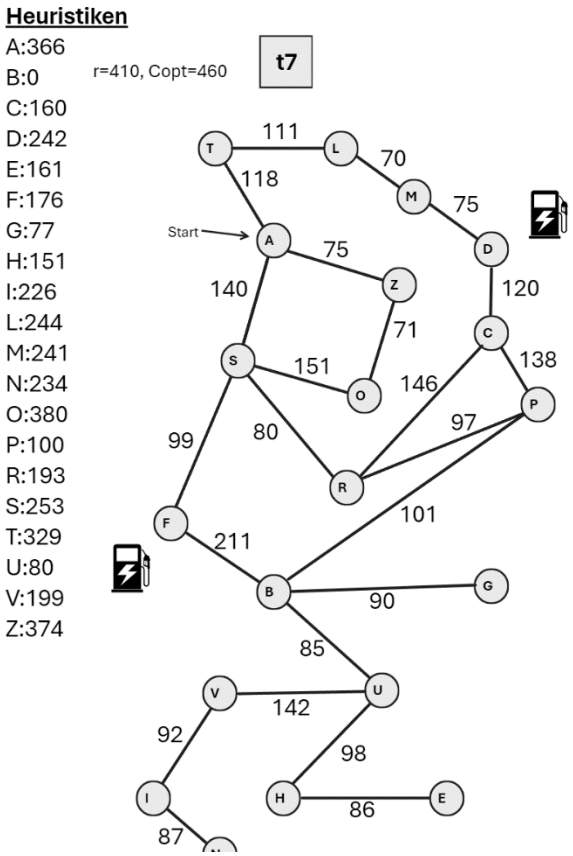

Abbildung 7: Test case 7

Wir haben festgestellt, dass Testfall 1 und 2 (siehe Abbildung 1 und 2) relativ ähnlich sind, abgesehen davon, dass Testfall 2 eine Ladestation bei C aufweist. Ebenso zeigen Testfall 4 und 5 (siehe Abbildung 4 und 5) sowie 6 und 7 (siehe Abbildung 6 und 7) Ähnlichkeiten. Jedoch unterscheidet sich Testfall 3 (siehe Abbildung 3) deutlich von den anderen Testfällen, was darauf hindeutet, dass es sich um eine besondere Situation handelt, die entsprechende Schwierigkeiten mit sich bringt.

Für die einzelnen Testfälle ergaben sich die folgenden optimalen Routen:

- Testfall 1: A → D → E → B
- Testfall 2: A → D → E → B
- Testfall 3: A → C → D → F → F(charged) → D → E → B
- Testfall 4: A → C → D → H → H(charged) → E → F → B
- Testfall 5: A → C → D → G → G(charged) → D → E → H → H(charged) → E → F → B
- Testfall 6: Es existiert keine Lösung für diesen Testfall.
- Testfall 7: A → S → F → F(charged) → B

Aufgabe 3

In Aufgabe 3 stießen wir auf ein erhebliches Problem, das durch das unnötige Hinzufügen von Knoten in den Frontier verursacht wurde. Es wurden kontinuierlich schlecht bewertete Knoten eingefügt, was zu einem erheblichen Anstieg des Speicherbedarfs führte und letztendlich das Laden des Outputs verhinderte. Dieses Problem wurde erfolgreich behoben, indem zusätzliche Bedingungen eingeführt wurden. Diese Bedingungen basierten auf Metriken, die sicherstellten, dass nur notwendige Knoten im Frontier verblieben. Nach dieser Optimierung wurde eine signifikante Reduzierung des Speicherbedarfs festgestellt, was schließlich die Generierung eines Outputs ermöglichte, aber dann lief Aufgabe 2 nicht korrekt. Dies war leicht zu beheben durch das Beheben der Heuristischen Berechnung, denn da wurde mit einem falschem Operator gerechnet. Dies hat dann bei beiden Aufgaben zu den gewünschten Ergebnissen geführt.

In der Aufgabenstellung wurde gefordert, den Mittelwert für jedes Maß über alle Testfälle zu berechnen. Die entsprechenden Werte sind hier angegeben:

- Durchschnittliche maximale Frontier-Größe für alle Testfälle: 902,36
- Durchschnittliche Anzahl erweiterter Knoten für alle Testfälle: 506,73
- Durchschnittliche Reisekosten: 271,83

Aufgabe 4

Wir haben uns für die Manhattan-Distanz entschieden, in der Hoffnung auf eine Reduzierung des Speicherbedarfs und der Laufzeit. Leider war dieser Ansatz nicht erfolgreich, da der Speicherbedarf sogar höher ausfiel als bei der Haversine-Distanz. Dies führte sogar dazu, dass das Programm nicht erfolgreich ausgeführt werden konnte.

Dennoch haben wir uns entschieden, bei der aktuellen Verwendung der Haversine-Distanz die Metrik zu ändern, die die Rundung der Erde repräsentiert. Konkret haben wir die Erhöhung dieser Metrik vorgenommen. Dies führte dazu, dass die durchschnittlichen Reisekosten zwar gesunken sind, jedoch gleichzeitig die durchschnittliche maximale Frontier-Größe und die durchschnittliche Anzahl erweiterter Knoten zugenommen haben.