Der Auslieferungsfahrer

Finde die kürzeste Route und hilf dem Traveling Salesman!

Autor: Daniel Prinz

26. April 2018

Kurzfassung

Auf IT-Talents ist eine Aufgabenstellung zu dem Travelling Salesman Problem (TSP) zu finden. Das Travelling Salesman Problem handelt von der Problematik, mehrere Orte in einer Reihenfolge besuchen zu wollen, dass die Wegkosten hierfür möglichst gering ausfallen. In der vorliegenden Aufgabenstellung wird zusätzlich vorausgesetzt, dass es sich bei dem Fahrzeug um einen Elektro-LKW handelt, der beim Bergab fahren Energie zurückgewinnen kann, beim Bergauf fahren aber mehr Energie benötigt. Zusätzlich zu den normalen Wegkosten des Travelling Salesman Problem fließen daher auch die Rückwegkosten mit in die Berechnung ein.

Schlagwörter: Schlagwort 1, Schlagwort 2, ...

Inhaltsverzeichnis

1 Anforderungen an das Programm

Es soll ein Programm entwickelt werden, das alle Anforderungen abdeckt. Es soll die Möglichkeit geboten werden, einen Startpunkt und bis zu 10 weitere Zieladressen anzugeben, wobei eine GUI nicht unbedingt notwendig ist. Bei der Routenplanung geht es dann in erster Linie um die Kosten der Fahrt für einen Elektro-LKW, weniger um die gefahrene Strecke, wobei diese natürlich zuvor in die Berechnung mit einfließt. Dabei gilt der in Tabelle 1.1 angegebenen Energieverbrauch.

Tabelle 1.1: Der Energieverbrauch des Elektro-LKWs

Steigung	Energieverbrauch
Bergab (Steigung < 3 %)	90kw/h pro 100km
Normal	100kw/h pro 100km
Bergauf (Steigung > 3 %)	120kw/h pro 100km

2 Unterschiede gegenüber dem einfachen Travelling Salesman Problem

Bei dem fortgeschrittenen Travelling Salesman Problem, das in diesem Programm gelöst werden soll, sind die veränderlichen Wegkosten je nach Wegrichtung mit einzuberechnen. Einfache Algorithmen bestimmen die Wegkosten in beide Richtungen, was aber auch unter anderen, möglicherweise zukünftigen, Bedingungen problematisch werden könnte, wie z.B. der Implementierung des Verkehrsaufkommens oder des Wetters. Daher ist es besser, die anfallenden Kosten zu trennen und separat zu berechnen.

3 Der Algorithmus

3.1 Basis

Zunächst wurden einige einfache Analysen manuell durchgeführt. Hierzu wurde ein Cluster von 4 Wegpunkten (A, B, C und D) aufgebaut, die jeweils zweifach mit allen anderen Punkten verbunden wurden (Hinweg und Rückweg). Die Gewichtungen der Wege wurde zufällig zwischen 2 und 9 vorgenommen.

AB - 4	BA - 7
AC - 7	CA - 9
AD - 7	DA - 3
BC - 2	CB - 2
BD - 8	BD - 6
CD - 4	CD - 6

Abbildung 3.1: Erste Wegkosten

Daraufhin wurden manuell alle möglichen Wegkombinationen gebildet und die Gesamtkosten für die jeweiligen Routen ausgerechnet.

ABCDA = 13	BACDB = 24	CABDC = 27	DABCD = 13
ABDCA = 27	BADCB = 22	CADBC = 24	DACBD = 20
ACBDA = 20	BCADB = 24	CBADC = 22	DBACD = 24
ACDBA = 24	BCDAB = 13	CBDAC = 20	DBCAD = 24
ADBCA = 24	BDACB = 20	CDABC = 13	DCABD = 27
ADCBA = 22	BDCAB = 27	CDBAC = 24	DCBAD = 22
(a) Startpunkt A	(b) Startpunkt B	(c) Startpunkt C	(d) Startpunkt D

Abbildung 3.2: Erste Wegberechnung

Aus Abb. 3.2 kann nun der günstigste Weg abgelesen werden (Wegkosten = 13). Es fällt auf, dass alle Wegkosten in unterschiedlicher Reihenfolge in jeder einzelnen Spalte vorkommen und dabei die Reihenfolge der besuchten Orte gleich ist. Es reicht daher aus, für einen beliebigen Startpunkt den günstigsten Weg zu finden und die Orte solange in ihrer Position zu shiften, bis der Startpunkt übereinstimmt. Beachten muss man hierbei nur, dass der Endpunkt nicht geshiftet werden darf, sondern immer synchron zum Startpunkt bleibt.

Listing 3.1 - Algorithmus Pseudocode

```
for(startpunkt : [A,B,C,D]) {
    while(!alleBesucht()) {
        guenstigstenNachbarnFinden();
        zuNachbarnLaufen();
    }
}
```

In Listing 3.1 wird der Algorithmus noch einmal verdeutlicht.

3.2 Testing

Der so entwickelte Algorithmus wurde anschließend weiteren Test unterzogen, um auch auf Spezialfälle eingehen oder Optimierungen finden zu können. Zunächst wurde der Datensatz hierfür leicht verändert, indem eine ursprünglich teure Wegstrecke deutlich vergünstigt wurde. Es wurde erwartet, dass nun eine neue Wegstrecke gefunden wird, die günstiger als zuvor ist.

AB - 4	BA - 7
AC - 7	CA - 9
AD - 7	DA - 3
BC - 2	CB - 2
BD - 1	BD - 6
CD - 4	CD - 6

Abbildung 3.3: Zweite Wegkosten

Auch für diesen Durchlauf wurden händisch die zu erwartenden Ergebnisse ausgerechnet, um einen Fehler oder mögliche Optimierungen des Algorithmus schnell finden zu können.

ABCDA = 13	BACDB = 24	**CABDC = 20**	DABCD = 13
ABDCA = 20	BADCB = 22	CADBC = 24	**DACBD = 13**
ACBDA = 13	BCADB = 24	CBADC = 22	DBACD = 24
ACDBA = 24	BCDAB = 13	**CBDAC = 13**	DBCAD = 24
ADBCA = 24	**BDACB = 13**	CDABC = 13	**DCABD = 20**
ADCBA = 22	**BDCAB = 20**	CDBAC = 24	DCBAD = 22
(a) Startpunkt A	(b) Startpunkt B	(c) Startpunkt C	(d) Startpunkt D

Abbildung 3.4: Zweite Wegberechnung

Die Bereiche, in denen sich im Vergleich zur ersten Wegberechnung Daten verändert haben, wurden in Abb. 3.4 **fett** hervorgehoben. Die durch den in Abschnitt 3.1 beschriebenen Algorithmus gewonnenen, günstigsten Wegstrecken für die jeweiligen Startpunkte wurden in Abb. 3.5 dargestellt. Der Algorithmus wird hierdurch bestätigt.

ABDCA = 20
BDACB = 13
CBDAC = 13
DABCD = 13

Abbildung 3.5: Algorithmus auf zweite Wegkosten angewandt

3.3 Gleiche Wegkosten - Gemeinsam oder getrennt betrachten?

Die Differenzierung „günstiger“ oder „teurer“ ist nicht immer ganz so einfach. In den beiden bisherigen Beispielen kam es nicht vor, dass die Wegstrecken von einem Punkt aus gleich teuer – oder gleich günstig – waren. Das soll sich nun ändern. Die neuen Kosten werden in Abb. 3.6 aufgeführt. Die Wege vom Wegpunkt B nach C und D kosten jetzt beide 1. Wir berechnen erneut die zu erwartenden Kosten zunächst manuell und dann mit Hilfe des Algorithmus.

AB - 4	BA - 7
AC - 7	CA - 9
AD - 7	DA - 3
BC - 1	CB - 2
BD - 1	BD - 6
CD - 4	CD - 6

Abbildung 3.6: Dritte Wegkosten

Es zeigt sich, dass auch in diesem Fall die günstigsten Wegkosten gefunden wurden. Es stellt sich jedoch schnell die nächste Frage, ob man bei einer Kostengleichheit keinem, nur einem oder allem Wegen folgen muss. War es nur Zufall, dass die Wegkosten 12 mit dem Startpunkt C auf Anhieb gefunden werden konnten?

ABCDA = 12	BACDB = 24	CABDC = 20	**DABCD = 12**
ABDCA = 20	BADCB = 22	**CADBC = 23**	DACBD = 13
ACBDA = 13	**BCADB = 23**	CBADC = 22	DBACD = 24
ACDBA = 24	**BCDAB = 12**	CBDAC = 13	**DBCAD = 23**
ADBCA = 23	BDACB = 13	**CDABC = 12**	DCABD = 20
ADCBA = 22	BDCAB = 20	CDBAC = 24	DCBAD = 22
(a) Startpunkt A	(b) Startpunkt B	(c) Startpunkt C	(d) Startpunkt D

Abbildung 3.7: Dritte Wegberechnung

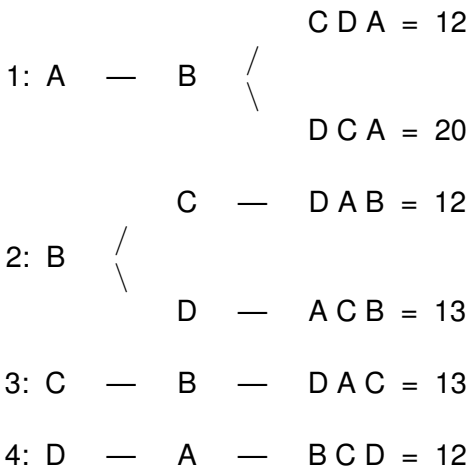

Abbildung 3.8: Algorithmus auf dritte Wegkosten angewandt

3.4 Wegkostenberechnung: Folgen, nicht folgen oder nur teilweise folgen?

Um auch diese Problematik zufriedenstellend beantworten zu können, wurden die Testdaten aus der dritten Berechnung (Abb. 3.6) so verändert, dass das Ergebnis deutlich wird. Durch einfache Überlegung kommt man zu dem Schluss, dass in jedem Fall allen neuen Zweigen gefolgt werden muss, da ansonsten bei zufälliger, wenn auch unwahrscheinlicher, Gleichheit aller Wege der verbleibenden Adresspunkte die Berechnung abbrechen würde. Da es nicht das Ziel ist, den LKW-Fahrer unentschlossen auf halber Wegstrecke stehen zu lassen, wird daher die Berechnung für alle Zweige durchgeführt. Sich zufällig für nur einen der Zweige zu entscheiden ist ebenfalls nicht korrekt, wie aus Abb. 3.8 ersichtlich wird. Die Entscheidung für den unteren AB-Zweig würde in Verbindung mit einem etwas modifizierten Kostenschema zu einem falschen Endergebnis führen.

3.5 Robustheitsprüfung

Ein letztes Mal wird der Algorithmus jetzt mit den nachfolgenden Daten getestet. Sie sind so gewählt, dass ein Subzweig erreicht wird. Der Algorithmus kann aber auch hier den günstigsten Weg finden – zumal die Wahrscheinlichkeit eines gleich günstigen Wegs in der Realität sehr gering ist (hierzu wäre eine so genaue Kombination aus Strecke und Gefälle/Steigung notwendig, dass sehr viele Nachkommastellen exakt übereinstimmen).

AB - 4	BA - 7
AC - 4	CA - 9
AD - 7	DA - 3
BC - 1	CB - 2
BD - 1	BD - 6
CD - 4	CD - 6

Abbildung 3.9: Vierte Wegkosten

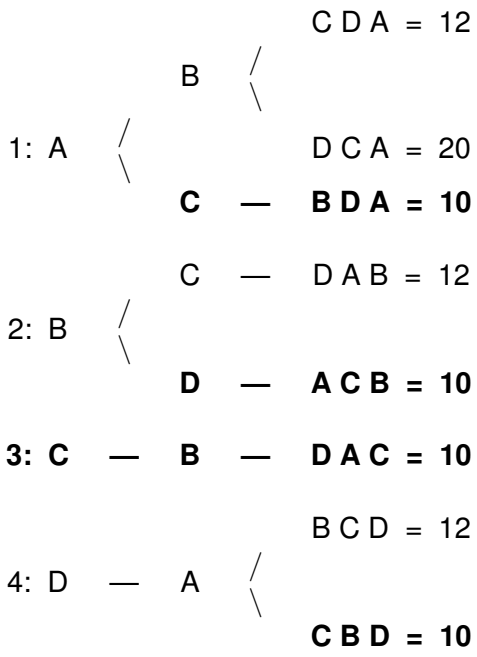

Abbildung 3.10: Algorithmus auf vierte Wegkosten angewandt

3.5.1 Standard Höhenfeststellung

Die einfache Vorgehensweise ist es, die Höhenunterschiede für die Berechnung der Kosten durch das Delta der Höhen der beiden Punkte zu bilden. Für den Anfang ist das ausreichend, für das weitere Vorgehen sollte jedoch eine Methode wie in Abschnitt 3.5.2 verwendet werden.

3.5.2 Fortgeschrittene Höhenfeststellung

Die fortgeschrittene Höhenmessung besteht darin, die Höhenunterschiede differenziert auszurechnen, um auch Höhenunterschiede auf der Strecke berücksichtigen zu können. In diesem Programm wurde dies durch die von den Weganweisungen der Google-API generierten Zwischenpunkte implementiert.

4 Die GUI

Die GUI wurde simpel gehalten. Es stehen 11 Texteingabefelder zur Verfügung, in die die Adressen eingegeben werden können. Ein Einstellungsfenster lässt den Nutzer den LKW-Verbrauch on-the-fly ändern. Nach Betätigung des Suche-Buttons wird im Hintergrund der Algorithmus (siehe Kapitel 3) ausgeführt. Nach Beendigung des Vorgangs, dessen Fortschritt man in der Titelleiste betrachten kann, erscheint ein neues Fenster, in dem links ein Bild der Karte und rechts Weganweisungen erscheinen.

5 Bekannte Probleme

Während der Entwicklung traten Probleme auf, die nicht behoben wurden.

5.0.1 GUI: Alternative Routen

Der Algorithmus unterstützt alternative Routen, die per Zufall genauso günstig sind wie die dargestellte Route. Diese sind jedoch aus Zeitgründen nicht in die GUI implementiert.

5.0.2 GUI: Polylinien

Eine der Polylinien der Static Map scheint manchmal als gerader Strich dargestellt zu werden. Ob es sich dabei um ein Problem der Anfrage an die Google-API oder um ein Problem beim Auswerten der Rückgabewerte handelt, ist zur Zeit unbekannt. Weiterhin tritt ein Fehler auf, wenn zu viele Polylinien angefragt werden (die Grenze ist zwichen 641 und 1934 Polylinien). Dies kann behoben werden, indem z.B. jede dritte oder vierte Polylinie, je nach Länge, für das Rendern entfernt wird. Dies wurde aus zeitlichen Gründen nicht implementiert, stattdessen wird nur eine Wegbeschreibung angezeigt.

5.0.3 Anfragen zusammenfassen

Google bietet bei einigen APIs die Möglichkeit, Anfragen zusammenzufassen. Dies erhöht den Umfang der Auswertung der Daten jedoch leicht, weshalb auch diese Lösung aus zeitlichen Gründen nicht implementiert wurde. Weiterhin kann ein TimeManager implementiert werden, der aufzeigt, wie viel Zeit des Algorithmus für API-Anfragen oder Rechenzeit aufgewendet werden müssen. Anhand dessen kann weiter optimiert werden.

5.0.4 Abfahrzeit

In dem Programm wird der Verkehr auf den Straßen berücksichtigt. Wenn die Abfahrzeit später ist, sollte dies in den Einstellungen geändert werden können. Aus zeitlichen Gründen nicht implementiert.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingsverzeichnis

Abkürzungsverzeichnis

GUI Graphical User Interface

TSP Travelling Salesman Problem