Optimale Route zu kommunalen Einrichtungen – oder wie kommt das Essen frisch und warm zu den hungrigen Kindern?

Stand der Version: 18. Dezember 2020; 7:15 Uhr

Routenplanung mit R - Einleitung und Problemstellung:

Die Plattform OpenTripPlanner (OTP) ist eine kostengünstige Möglichkeit, routing-fähige Daten auf kommunaler Ebene bereit zu stellen. Dafür wird der Umweg über frei verfügbare Datenquellen (GTFS für ÖPNV Fahrpläne und OpenStreetMap für das Straßennetz) genutzt, mit dem Nachteil, dass diese oft nicht aktuell bzw. amtlich einwandfrei sind. Diesen Nachteil gleicht OTP allerdings aus, da OTP auf einem eigenen städtischen Server betrieben werden kann und dort dann entsprechend der Änderungsroutinen der Open Source Produkte gepflegt und aktualisiert werden kann. Damit hat man beispielsweise die Möglichkeit, neue ÖPNV Zugangsstellen oder Standorte für kommunale Einrichtungen im existierenden Straßen- und Wegenetz zu überprüfen. Ebenso kann man stadtplanerisch den Server nutzen und neue Wege auf ihre Alltagstauglichkeit hin optimieren, indem diese beispielsweise auf der geklonten Kopie von OSM eingetragen werden.

Dieser Artikel zeigt eine Lösung für das Salesmanproblem auf und stützt sich dabei auf einen lokalen OTP-Server. Mit dessen Hilfe und in Verbindung mit dem `R`-Paket `opentripplanner`, das auf die Funktionalität des OTP-Servers zugreift, können Routen von beliebigen Punkten in verschiedenen Modalitäten berechnet werden. Auf dieser Grundlage lässt sich dann die optimale Route nach den Kriterien (minimaler) Zeitverbrauch, um alle Obkjekte anzufahren, (minimale) gefahrene Strecke zwischen den Objekten oder (maximal) ausgelieferten Dingen (im Sprachgebrauch des Artikels `Nkriterium` genannt) bestimmen. Dafür hat der Autor entsprechene `R`-Routinen geschrieben und stellt diese hier vor.

Im Artikel geht es zunächst um die Vorraussetzungen, dann werden die im Artikel benutzten Beispieldaten vorgestellt um dann die Routenplanung in 7 Schritten ausführlich zu beschreiben und anhand des `R`-Codes zu erklären. Eine Zusammenfassung schließt diesen Artikel ab.

R und das Paket opentripplanner

Um `opentripplanner` in `R` nutzen zu können, muss (natürlich) die Software `R` installiert sein. Es empfiehlt sich gleich `R-Studio` als graphische Oberfläche zur Steuerung von `R` mit zu installieren.

Folgende Schritte sind zur Verwendung des R-Paketes `opentripplanner` notwendig:

Ansprechpartner: Volker Holzendorf,
Team Controlling und Statistik, statistik@jena.de

- Download von R
- Download von R-Studio

Im Internet finden sich viele deutsche (und noch mehr englische) Einführungen in `R` als Web-Buch, PDF oder Video. Die Einführung in R der Schweizer Autoren Boris Mayer und Andrew Ellis liest sich gut und ist in ihrem Umfang für Menschen, die mit `R` beginnen wollen, bestens geeignet. Die Einführung enthält Übungsaufgaben und gibt auch die Lösungen dazu mit an.

Für Neueinsteiger*innen in die Welt von `R` bietet sich auch die Arbeitsgruppe KO.R des KOSIS-Verbundes an.

In diesem Paper wird `R` zusammen mit `R-Studio` als Arbeitsgrundlage vorausgesetzt.

Das R-Paket `opentripplanner` wird in einer Vignette des Mitautors Malcom Morgan beschrieben (auf englisch). Diese ist tagesaktuell und basiert auf einem Paper des anderen Autoren des R-Paketes Marcus Young, das als PDF download verfügbar ist.

Systemvoraussetzungen

Optimale Systemvoraussetzung für den reibungslosen Betrieb eines lokalen OTP-Servers ist ein CPU-Prozessor ab Baujahr 2010, ein 64-Bit Betriebssystem, ein Arbeitsspeicher von mindestens 8 GB und eine 64 GB große Festplatte. Minimale Anforderung ist ein 32-Bit Betriebssystem und 2 GB Arbeitsspeicher. Der lokale OTP-Server benutzt standardmäßig 2 GB Arbeitsspeicher. Einen Nährungswert für den benötigten Speicherplatzbedarf gibt die nebenstehende Tabelle an.

Speicherbedarf	OTP Region
2 GB	kleine Stadt/ Region
4 GB	Großstadt
8 GB	Landkreis / Metropolregion
20 GB	Bundesland / Land
50 GB	Kontinent

Des Weiteren benötigt OTP die Java 8 Version. Sollte diese nicht bereits schon auf dem Rechner laufen, kann sie hier heruntergeladen werden. Hat man bereits das R-Paket `opentripplanner` installiert, kann man die installierte Java Version einfach mit der function `otp_check_jacva()` prüfen. Rückgabewert der Funktion ist ein einfaches `TRUE`, falls eine mit OTP kompatible Java Version installiert ist.

Beispieldaten dieses Artikels

Für diesen Artikel liegen zwei Datenbeispiele auf der OwnCloud der Stadt Jena bereit. Diese sind passwortgeschützt (VDST2021!) zugänglich und können heruntergeladen werden. Anhand der Beispieldaten wird in diesem Artikel die Funktionalität des OTP in `R` erklärt.

Beispiel 1 behandelt Jenaer Kindertageseinrichtungen, die von einem fiktiven Essensanbieter beliefert werden. Die Kinder in den verschiedenen Einrichtungen wollen zwischen 11:30 und

12:30 Uhr mit dem Essen beginnen. Das Ausliefern des Essens erfolgt über jeweils einen Essenscontainer pro Einrichtung. Dieser wird durch den Fahrdienst in die Einrichtung geschoben. Das Ausladen vor Ort dauert inkl. Parkplatzsuche und Rollcontainer in die Einrichtung bringen 10 Minuten. Frage ist, wie viele Touren benötigt der Essensanbieter, damit alle Kinder gut gesättigt den Mittagsschlaf spätestens 13 Uhr antreten können und kein Essen vor 11:30 Uhr ausgeliefert wird? Dieses Beispiel wird im Artikel beschrieben.

Beispiel 2 liefert neue Sitzkissen für Leipziger Kirchen aus. Die im Leipziger Westen gelegenen Kirchen sollen innerhalb von einer 8h Schicht mit diesen neuen Kissen für jeden Sitzplatz beliefert werden. Für das Entladen vor Ort werden im Schnitt 60 Minuten benötigt. Frage ist, wieviele Sitzkissen können maximal in einer 8h-Tour ausgeliefert werden? Hierfür liegt im Cloud-Verzeichnis der Programmablauf bereit und dient der eigenen Übung.

Ablauf der Routenplanung

Sind die Voraussetzungen geprüft und ist es dem Rechner möglich, aus dem Stadtnetz ins Internet zu kommunizieren, kann nun mit der Arbeit begonnen werden.

Im Folgenden wird der Arbeitsablauf in 7 Schritten für *Beispiel 1 - Essensversorgung in Jenaer Kitas* beschrieben.

Schritt 1: Download der Daten / Anlegen des Arbeitsverzeichnisses

Die notwendigen `R-Codes` und `R-Funktionen` liegen auf der Jenaer OwnCloud bereit. Am besten lädt man das gesamte Verzeichnis herunter und entpackt es in dem frei gewählten Arbeitsverzeichnis auf dem Rechner. Danach öffnet man die R-Projektdatei `OTP_VDST.Rproj`. Nun geht die Arbeitsumgebung von `R` in `R-Studio` auf und wir sehen im Reiter *Files* (in `R-Studio` im Teilfenster unten rechts) die gesamte Struktur des Verzeichnisses. Die Datei `VDST_Jena.r` führt vollständig kommentiert durch das *Beispiel 1 - Essensversorgung in Jenaer Kitas*. Sie öffnet sich mit Doppelklick auf den Namen.

Schritt 2: Vorbereitung der Arbeitsumgebung

Um das Beispiel nachvollziehen zu können, sollten bestimmte `R`-Pakete installiert werden. In der Datei sind diese angelegt aber auskommentiert:

```
##Installiere notwendige Pakete ggf. beim ersten Mal.
#install.packages("opentripplanner")
#remotes::install_version("opentripplanner", "0.2.3")
#install.packages("tidyverse")
#install.packages("mapview")
#install.packages("sf")
```

Danach können die Pakete in die bestehende `R`-Umgebung geladen werden, damit sie verwendet werden können. Das R-Paket opentripplanner sollte vorläufig in der Version 0.2.3 installiert werden.

```
## load up the packages we will need
library(opentripplanner)
library(tidyverse)
```

```
library(mapview)
library(sf)
options(scipen=999) # Keine wissenschaftliche Notation von
                    # großen Zahlen in R Ausgaben
```

Die nächsten Programmzeilen …

```
# Lade entsprechende Funktionen:
# ... allgemeine Funktionen
# Funktion zur Erstellung des lokalen OTP Servers
source("R-Codes/createOTPserver.r")
# Funktion zur Erstellung der (Unter-)Verzeichnisse in notwendiger Struktur
source("R-Codes/mkrR0_Verzeichnisse.r")
# ... zur Routenberechnung:
# Funktion zur Ermittlung der optimalen Route
source("R-Codes/mkrR1_optimaleroute.r")
# WrapperFunktion, die bis zu 10 optimalen Routen
source("R-Codes/mkrR2_OTProuteswrapper.R")
# Funktion zur Darstellung der ermittelten Routen
source("R-Codes/mkrR3_routeMaps.R")
# Funktion, die die Metadaten der berechneten Routen ausgibt
source("R-Codes/mkrR4_routeMetadata.R")
# Wrapper-Funktion, die die gesamte Routenanalyse durchführt
source("R-Codes/mkrR5_routeAnalyse")
```

… laden die Funktionen in die Arbeitsumgebung, die extra für diesen Artikel geschrieben wurden. Diese Funktionen erlauben es, den lokalen OTP-Server zu initialisieren, legen die Standardverzeichnisse an und stellen die R-Funktionen zur Verfügung, die zur Routenberechnung benötigt werden.

Schritt 3: Anlegen der Standardverzeichnisse

Nun können die Standardverzeichnisse für den lokalen OTP-Server angelegt werden. Diese benötigen die zur Verfügung gestellten Funktionen, damit die Ergebnisse korrekt abgespeichert werden können. In diesem Beispiel nennen wir den lokalen OTP-Server "*OTPJ*" und legen unter diesem das Verzeichnis "*Jena*" an.

```
# Erstelle notwendige Verzeichnisse im Arbeitsordner
city <- "Jena"
OTPname <- "OTPJ"
path<-mkrR0_Verzeichnisse(OTPname,city,setup = TRUE)
```

Exkurs 1: OSM Daten für die eigene Region

Um OTP für die eigene Region zu nutzen, muss zunächst die interessierende Region aus der OSM-Karte ausgeschnitten werden. Diese sollte dann im Format `*.pbf` (oder `*.osm`) vorliegen. Der lokale OTP-Server benötigt lediglich das Straßen- und Wegeverzeichnis von OSM. Für die Beispiele dieses Artikels liegen die notwendigen OSM-Dateien bereits im Verzeichnis "*OSMData*" vor. Die finale Datei (hier: `th_streetnet.pbf`) muss allerdings noch in das Verzeichnis des OTP-Servers unter *OTPname/city* (hier: "*OTPJ/Jena*") abgelegt werden.

Will man für eine andere Region das OSM-Straßennetz herunterladen, sollte man wie folgt vorgehen. Hier werden nun die Arbeitsschritte für das Thüringer Beispiel beschrieben, die aber einfach für die interessierende Region adaptiert werden können.

Als **erstes** legt man das Unterverzeichnis "*OSMData*" an (ist schon im Download von der OWNCloud Jena enthalten). In dieses Verzeichnis lädt man - **zweitens** - die Gesamtdatei des interessierenden Bundeslandes von der Downloadseite der GEOFABRIK. In der Tabelle unter 'Sub Regions' am Ende der Internetseite einfach auf `[.osm.pbf]` klicken. Diese Datei muss nun - **drittens** - mit den OSM-Funktionen osmconvert und osmfilter bearbeitet werden. Diese beiden Funktionen sind bereits ebenfalls im Verzeichnis "*OSMData*" des Jena-Cloud Downloads enthalten. Für `osmconvert` gibt es je nach Betriebssystem (32-bit oder 64-Bit) eine andere Version. Nun ruft man - **viertens** - die Kommandozeile von Windows auf (`Windows-Taste+R` –> Eingabe `cmd`) und wechselt in das Verzeichnis "*OSMData*". Dazu muss man in der Regel das Laufwerk wechseln, um an das richtige Verzeichnis zu gelangen. Dies ist am einfachsten, indem man in der Kommandozeile den entsprechenden Laufwerksbuchstaben mit Doppelpunkt angibt und mit `Enter` die Zeile abschließt. Danach wechselt man in das Arbeitsverzeichnis mit dem `cd`-Befehl. Den Verzeichnispfad kann man direkt aus dem Windows Explorer kopieren und in die Kommandozeile einfügen. Nun müssen - **fünftens** - folgende (fettgedruckten) Befehle in der Kommandozeile ausgeführt werden, was anhand von Thüringen erklärt wird:

1. Konvertiere die OSM-Datei in ein Format, das nach OSM Attributen zu filtern ist:
```
osmconvert64 thueringen-latest.osm.pbf -o=thueringen.o5m
```
2. Extrahiere nun Wegenetz auf der konvertierten OSM-Datei:
```
osmfilter thueringen.o5m --keep="highway=" -o=th_streetnet.o5m
```
3. Schließlich muss wieder in das pbf-Format konvertiert werden.
```
osmconvert64 th_streetnet.o5m -o=th_streetnet.pbf
```

`osmconvert64`ist für ein 64-bit System. bei einem 32-bit System nimmt man dann `osmconvert`.

Nun muss - **sechstens** - die Enddatei (hier: `th_streetnet.pbf`) noch in das Verzeichnis des OTP-Servers (hier: "*OTPJ/Jena*") abgelegt werden.

Exkurs 2: Objektdaten erstellen

Für das *Beispiel 1 - Essensversorgung in Jenaer Kitas* liegen die Objektdaten im Verzeichnis "*Objektdaten*" des OwnCloud-Jena Verzeichnisses bereit. Diese `Rdata`-Datei muss nun in das unter Schritt 3 erstellte Standardverzeichnis "*R-Data*" gelegt werden. In diesem Fall heißt die Datei `KitaEssen_Jena_objdata.Rdata`.

Eine eigene Objektdatendatei sollte minimal folgende Spalten haben.

Spalte	Beschreibung der Spalte
Nr	ID des anzufahrenden Objektes (Verknüpfungskriterium)
Name	Name des anzufahrenden Objektes

Nkrit	Nkriterium, nach dem zusätzlich optimiert werden kann
geometry	Geokodierung des Objektes in WGS84

Die im deutschen Sprachraum üblichen Umlaute machen in R beim Einlesen oft Probleme, so dass die (zusätzliche) Spalte `Nr` als Zuspielkriterium empfohlen wird.

Wie die Datei erstellt wird, kann über die bereitgestellten R-Codes `KitaEssen_Jena_objdata.r` und `Kirchen_Leipzig_objdata.R` nachvollzogen werden. Das Jena-Beispiel benutzt eine ShapeDatei, um die Geokoordinaten zu den interessierenden Objekten (Kitas) zuzuspielen, während das Leipziger Beispiel die Geodaten als Longitude (`long`) und Latidude (`lat`) verwendet und im bereitgestellten R-Code in die notwendigen Geokoordinaten transformiert.

Schritt 4: Lade die Objektdatendatei in die Arbeitsumgebung

Um die Objektdatendatei in die Arbeitsumgebung zu laden, führe nun diese zwei R-Codezeilen aus:

```
# Lade die Geodaten der anzufahrenden Objekte
Objektdatendatei <- "KitaEssen_Jena_objdata.Rdata"
load(file = paste0("R-Data/",Objektdatendatei))
```

Schritt 5: Initialisiere den lokalen OTP-Server

Nun sind alle Voraussetzungen geschaffen, um den lokalen OTP-Server aufzubauen. Für große Gegenden muss entsprechend viel Arbeitsspeicher zur Verfügung stehen. In der R-Programmzeile wird der Arbeitsspeicher mit der Option `memory` zugewiesen.

Die Übergaben für `OTPname`, `city` und `path` wurden in Schritt 3 definiert und stehen somit bereits in der R-Arbeitsumgebung zur Verfügung.

```
## Lokalen OTP Server erstellen
otpcon <- createOTPserver(OTPname,city,memory = 8192,path)
```

Die Ausführung klappt nur, wenn der Rechner, von dem der Befehl ausgeführt wird, unbeschränkt ins Internet gehen darf. Dies kann manchmal bei kommunalen Netzwerken seitens der jeweiligen Netzadministratoren und der lokalen IT-Richtlinie nur eingeschränkt zugelassen sein. Hier sind vor Ort entsprechende Arbeitsvoraussetzungen zu prüfen.

Folgende Fehlermeldung:

```
 Fehler in otp_checks(otp = otp, dir = dir, router = router, graph = TRUE) :
  Assertion on 'paste0(dir, "/graphs/", router, "/Graph.obj")' failed: File d
oes not exist: 'C:/Jena/OTP_VDST _Test/OTPJ/graphs/Jena/Graph.obj'
```

tritt auf, wenn die unter Exkurs 1 erstellte OSM-Datei nicht im korrekten Verzeichnis liegt.

Schritt 6: Travel-Time-Matrix-berechnen

Ist nach ca. 5-10 Minuten der lokale OTP-Server erstellt, kann mit dem Herzstück der Analyse begonnen werden. Um die optimale Route zu ermitteln, müssen erst einmal die Teilrouten zwischen allen Objekten, die angefahren werden sollen, ermittelt werden. Dies ist mit folgenden

Codezeilen einfach zu ermitteln. Der Code orientiert sich an der englischen Einführung in das R-Paket opentripplanner.

```
# Start und Zielorte festlegen
toPlace   = objdata[rep(seq(1, nrow(objdata)), times = nrow(objdata)),]
fromPlace = objdata[rep(seq(1, nrow(objdata)), each  = nrow(objdata)),]
names(toPlace)
#  travel time matrix  - von überall nach überall
routesALL <- otp_plan(otpcon = otpcon,
                      fromPlace = fromPlace,
                      toPlace = toPlace,
                      fromID = as.character(fromPlace$Name),
                      toID = as.character(toPlace$Name),
                      get_geometry = TRUE)
```

Der Datensatz `routesALL` der erstellten Routen zwischen allen Objekten ermöglicht allerdings nur die Optimierung nach (minimaler) Wegelänge und (minimaler) Zeitdauer. Um auch noch nach dem Maximum von *Nkriterium* (Essen pro Kita, Sitzkissen pro Kirche und Ähnlichem) zu optimieren, muss dieses nun noch zugespielt werden. Dies erledigen diese Codezeilen:

```
# Nkriterium zuspielen - hier auszuliefernde Essen für die Kita-Kinder
# st_set_geometry(NULL) löscht die Geometrie aus objdata
dfFROM <- objdata %>%
  select(fromPlace = Name, Nkrit1 = Nkrit) %>%
  st_set_geometry(NULL)
dfTO   <- objdata %>%
  select(toPlace   = Name, Nkrit2 = Nkrit) %>%
  st_set_geometry(NULL)

# Spiele dies dem Datensatz für die Routenberechnung zu
routesALLk <- routesALL  %>%
  left_join(dfFROM, by = "fromPlace") %>%
  left_join(dfTO  , by = "toPlace") %>%
  select(fromPlace,Nkrit1,toPlace,Nkrit2,duration,distance,startTime,endTime)
```

Schritt 7: Berechnung der optimalen Routen

Schritt 7 erklärt ausführlich, was die in Schritt 8 vorgestellte R-Funktion macht. In der alltäglichen Verwendung dieses Arbeitsablaufes kann deswegen gleich zu Schritt 8 gesprungen werden.

Bleibt man in der Struktur der Standardverzeichnisse, kann nun mit wenigen Definitionen eine optimale Route berechnet werden.

Dazu müssen einfach die folgenden Angaben gemacht werden, die die Art der optimalen Route definieren:

```
Optimierung   <- "Zeit"     # Optimierungskriterium, String,
                             # "Zeit" - minimale Zeit,
                             # "Weg" - minimal zu fahrende Strecke,
                             # "Nkriterium" - maximal auszuliefernde/
                             #     einzusammelnde Objekte nach NKriterium
```

```
tvorOrt       <- 10          # Zeit in Minuten vor Ort (zum Be-/ Entladen)
Routenzeit    <- 60          # Zeit in Minuten, die eine Route maximal
                             # dauern darf.
                             # Die Route wird inklusive der Route berechnet,
                             # die als erste echt länger ist als:
                             # Abbruchroute*Routenzeit
Abbruchroute  <- 0.95        # Abbruchkriterium für weitere Route
                             # default: wenn Routenzeit zu 95% ausgeschöpft,
                             # breche ab und beginne neue Route
```

Des Weiteren sollte man für die Ausgabe angeben, wie das *Nkriterium* übersetzt heißt. In dem Beispiel des Artikels kümmern wir uns um Kita-Essen. Zudem sollte man noch angeben, ob alle Objekte in der Objektdatei **objdata** angefahren werden sollen. Dies steuert man über **Nkrit**.

```
LabNkriterium <- "Kita-Essen"    # Label für die Optimierung nach max(Nkrit)
Nkrit         <- 1               # Einrichtungen, die weniger als die angege-
                                 # bene Anzahl im NKriterium haben, werden
                                 # nicht angefahren.
```

Als letztes sollte man mit den Übergaben noch definieren, wie die Ergebnisdateien heißen sollen. Die Karten liegen dann im Standardverzeichnis "*Maps*" und die Daten im Unterverzeichnis "*Maps/Routedata*" (vgl. Schritt 3)

```
Datum         <- "20210201"          # Zeitpunkt für Routenberechnung -
                                     # wird für Dateinamen der Datensätze
                                     # verwendet
Label         <- paste0(tvorOrt,"s") # Label, das im Dateinamen auftaucht um
                                     # "sprechenden" Dateinamen zu haben,
                                     # lehnt  sich an die Angabe der Zeit
                                     # vor Ort an (tvorOrt)

# Dateiname der Datei, mit allen Daten zu den gefundenen optimalen Routen
Routendatendatei <-
                  paste0(Datum,"_optRoutes_",Optimierung,"_",Label,".Rdata")
Objektdatendatei <- Objektdatendatei # Datei der anzufahrenden Objekte mit
                                     # Geokoordinaten, siehe Exkurs 2 wie
                                     # das erforderliche Format ist.
```

Sind diese Definitionen alle erledigt, kann nun die Analyse der optimalen Route nach den definierten Kriterien beginnen.

Die Ermittlung der optimalen Route erfolgt in einem Optimierungs-Prozess: Dazu werden zunächst aus der in Schritt 6 erstellten Travel-Time-Matrix die Objekte entfernt, die nicht angefahren werden sollen (Schritt 7a). Danach werden iterativ die eigentlichen Routen erstellt. Sobald eine Route erstellt ist, werden die bereits angefahrenen Objekte aus der Travel-Time-Matrix entfernt und aus der verkleinerten Datei die nächste Route berechnet. Dies wird so lange

wiederholt, bis alle Objekte angefahren wurden (Schritt 7b). Die Schritte 7c-7d sind verschiedene Ansichten der ermittelten Routen.

Beachte, dass die optimalen Routen lediglich zwischen den anzufahrenden Objekten berechnet werden. Die An- bzw. Abfahrt von Standort des Dienstleisters findet (noch) keine Beachtung.

Schritt 7a: Ausgangs-Travel-Time-Matrix definieren

Aus der in Schritt 6 errechneten Travel-Time-Matrix müssen die Routen entfernt werden, die nicht berücksichtigt werden sollen. Dazu wurde in den Definitionen die Angabe und **Nkrit** gesetzt. Dies ergibt den Datensatz, aus dem die erste Route berechnet wird.

```
# a) Datensatz für Route 1:
# mindestens Nkrit auszuliefernde Essen
routesALL1 <- routesALLk %>% filter(Nkrit1 >= Nkrit & Nkrit2 >= Nkrit)
```

Schritt 7b: Benötigte Routen berechnen

Nun wird dieser Datensatz in den Programmablauf der Funktion zur Berechnung der optimalen Routen gesteckt. Dazu werden einige der eben gemachten Definitionen übergeben. Je nach den gemachten Angaben Art der Optimierung (**Optimierung**), zu Aufenthalt (**tvorOrt**) und maximaler Dauer der einzelnen Routen (**Routenzeit**), werden mit diesen Codezeilen unterschiedlich viele Routen berechnet.

```
# b) Berechne bis zu 10 Routen
mkrR2_optroutwrapper(routesALL1,Label = Label,Datum = Datum,Optimierung = Opt
imierung,Aufenthalt =  tvorOrt,Routenzeit=Routenzeit,Abbruchroute = Abbruchro
ute)
```

Sobald eine Route berechnet wurde, erscheint dies in der Ausgabe. Alle Daten der berechneten Routen und die Travel-Time-Matrix werden in dem Standardverzeichnis "*Maps/Routedata*" unter dem definierten Namen von **Routendatei** abgelegt.

Schritt 7c: Anzeige der Routen auf einer Karte

Nun kann man sich die Routen einfach auf einer Karte anzeigen lassen. Dabei werden die angefahrenen Objekte als gelbe Punkte mit ausgegeben. Zudem wird die Karte in das Verzeichnis "*Maps*" als intereaktive Karte in **html**-Format und statisch als **png**-Datei gespeichert.

```
# c) Zeige auf Karte an:
#    Aufruf zur Erstellung der Karte.
mkrR3_routemaps(
      Routendatendatei = Routendatendatei,
      Objektdatendatei = Objektdatendatei,
      NKriterium = LabNkriterium ,
      Grafikzeigen = TRUE#,  # TRUE wenn Grafik auf Bildschirm
                             # angezeigt werden soll.
                             # Unabhängig davon wird sie immer
                             # abgespeichert im Unterordner "Maps"
      #maptypes = c("OpenStreetMap.DE", "OpenTopoMap")
                )
```

Die verschiedenen Optionen für das Kartenbild (Option `maptypes`) sind auf Leaflet Extras zusammengestellt.

Essensversorgung in Jenaer Kitas: bezüglich Zeitdauer optimierte Routen

Essensversorgung in Jenaer Kitas: bezüglich Zeitdauer optimierte Routen

Schritt 7d: Metadaten der einzelnen Routen

Die Metadaten zeigen pro berechneter Route die Dauer, die angefahrenen Objekte, den zurückzulegenden Weg sowie die ausgefahrenen Dinge nach *Nkriterium* (hier: Kita-Essen) an.

```
# d) Daten der einzelnen Routen
mkrR4_routeMetadaten(Routendatendatei = Routendatendatei,
                     NKriterium = LabNkriterium)
```

Die Ausgabe ist in diesem Beispiel folgende:

```
Einrichtungen  16.00
Strecke        22.24
Kita-Essen    845.00

  Nr      Einrichtungen Fahrzeit Strecke `Kita-Essen`
  <fct>           <dbl> <fct>      <dbl>        <dbl>
1 Route 1             4 0h 57min    4.25          219
2 Route 2             4 0h 58min    4.54          244
3 Route 3             4 1h 1min     4.28          239
4 Route 4             4 1h 8min     9.17          143
```

Schritt 7e: Daten der einzelnen Routen

Will man nun wissen, wie die Einrichtungen heißen und in welcher Reihenfolge sie angefahren werden (sollten), lädt man die Routendatei, die in Schritt 7b erstellt wurde und lässt sich die einzelnen Datensätze der Routen anzeigen. Das Beispiel zeigt die berechnete Route 1.

```
# e) Ergbenisbeschau
load(paste0("R-Data/Routedata/",Routendatendatei))
route1$Tourdaten
```

Die Ausgabe für Route 1 ist demzufolge:

```
Simple feature collection with 3 features and 6 fields
geometry type:  LINESTRING
dimension:      XYZ
bbox:           xmin: 11.57731 ymin: 50.92775 xmax: 11.59955 ymax: 50.93717
z_range:        zmin: 0 zmax: 0
geographic CRS: WGS 84
       fromPlace Nkrit1          toPlace Nkrit2 duration distance
1  Pi mal Daumen     14     Glühwürmchen     55      421 1886.951
2   Glühwürmchen     55   Janusz Korczak     92      231  972.796
3 Janusz Korczak     92 Fuchs und Elster     58      387 1387.979
```

Schritt 8: Routenanalyse von Schritt 7 in einer R-Funktion

Statt der Schritte 7a - 7e kann auch die Funktion `mkrR5_RoutenAnalyse` gestartet werden. Damit wird die eigentliche Analyse noch schlanker und man kann schneller verschiedene Optimierungsverfahren gegeneinander testen. Ein typischer Codeaufruf ist der folgende. Dabei ist zu beachten, dass hier keine Angabe des Namens der `Routendatendatei` und des `Label`s gemacht werden muss. Der Funktionsaufruf generiert die sprechenden Namen selbst. Während des Funktionsdurchlaufs wird in der `R`-Konsole ausgegeben, was bereits erstellt wurde. Am Ende werden auch die Metadaten der Routenberechnung angezeigt und auf Wunsch mit der Option `showroutdata` die Daten der berechneten Routen in der Konsole angezeigt. Standardmäßig ist dies allerdings der Übersichtlichkeit wegen ausgeschaltet.

Beachte, dass in dem Beispiel für Schritt 8 die Routenzeit von 60 auf 90 Minuten herauf erhöht wurde.

```
mkrR5_routeAnalyse(
  Optimierung    = "Zeit",       # Optimierungskriterium, String
  tvorOrt        = 10,           # Zeit in Minuten vor Ort (zum Be-/ Entladen)
  Routenzeit     = 90,           # Zeit in Minuten, die eine Route
                                 # maximal dauern darf.
  Abbruchroute   = 0.95,         # Abbruchkriterium für weitere Route
  LabNkriterium = "Kita-Essen", # Label für die Optimierung nach max(Nkrit)
  Nkrit          = 1,            # Einrichtungen, die weniger als die angegebe
-
                                 # ne Anzahl im NKriterium haben,
  Datum          = "20210201",   # Zeitpunkt für Routenberechnung - wird für
                                 # Dateinamen der Datensätze verwendet
  Label          = NULL,         # Label, das im Dateinamen auftaucht, um
                                 # "sprechenden" Dateinamen zu haben, lehnt
                                 # sich an die Angabe `tvorOrt` an
  Routendatendatei = NULL,       # Dateiname der Datei, mit allen Daten
                                 # zu den gefundenen optimalen Routen
  Objektdatendatei = Objektdatendatei,  #Datei der anzufahrenden Objekte mit
                                        # Geokoordinaten
  showroutdata = FALSE           # TRUE, wenn Daten der einzelnen berechneten
                                 # Routen angezeigt werden sollen
)
```

Zusammenfassung

Der Artikel hat in 7 Schritten gezeigt, wie das Salesmanproblem mit dem `R`-Paket `opentripplanner` gelöst werden kann. Der vorgeschlagene `R`-Code ist dabei so gestaltet, dass lediglich 10 Angaben individuell gesetzt werden müssen. Interessant ist der Lösungsansatz auch deswegen, weil er einen kostenlosen Routenplaner zur Verfügung stellt, der auf kleinräumiger Ebene Routen in verschiedenen Modalitäten anbieten kann. Dies ist insbesondere dann der Fall, wenn die GTFS-Daten für den lokalen ÖPNV mit eingebunden werden. Leider ist die Oberfläche des lokalen OTP-Servers englisch. Hier wäre eine Weiterentwicklung aus dem KOSIS-Verbund wünschenswert, die eine deutsche Navigation entwickelt.

Abkürzungsverzeichnis

Abkürzung	Bedeutung
GB	Giga-Byte
GTFS	General Transit Feed Specification
KOSIS	(Verbund) Kommunales Statistisches Informationssystem
OSM	OpenStreetMap
OTP	OpenTripPlanner
ÖPNV	Öffentlicher Personen Nahverkehr
R	R-Statistiksoftware (Open Source)
VDSt	Verband deutscher Städtestatistiker

Literaturverzeichnis

Morgan et al., (2019). OpenTripPlanner for R. Journal of Open Source Software, 4(44), 1926, https://doi.org/10.21105/joss.01926

OpenTripPlanner: https://docs.opentripplanner.org/en/latest/

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.