Softwareprojekt: Rekonstruktion metrischer Graphen

Terese Haimberger, Lea Helmers, Mahmoud Kassem, Daniel Theus, Moritz Walter

Inhaltsverzeichnis

1 Einleitung

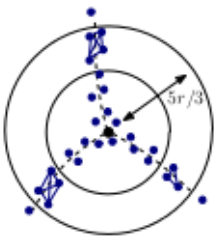
1.1 Motivation und Problemstellung

Im Rahmen des Softwareprojekts „Anwendungen effizienter Algorithmen“ haben wir uns damit befasst, einen Algorithmus umzusetzen, der aus einer Punktmenge den zugrundeliegenden Graphen sowie dessen Metrik rekonstruiert. Dadurch soll Struktur in große Mengen geometrischer Daten gebracht werden, was deren Analyse und Weiterverarbeitung erleichtert. Zu verarbeitende Daten können Netzwerke im weitesten Sinne sein, wie beispielsweise GPS-Daten, Strom- und Nachrichtennetze oder astrologische Daten. Häufig enthalten diese Daten Rauschen oder Ausreißer und sind zudem im Allgemeinen sehr umfangreich. Ziel des Algorithmus ist es, die Datenmenge kompakt durch ihre wichtigsten Verzweigungen darzustellen, wodurch sie auf ihre wichtigen Aspekte reduziert wird. So kann eine einfachere Visualisierung und die weitere Analyse und Verarbeitung der Daten ermöglicht werden.

1.2 Der Algorithmus

Als Grundlage für unsere Arbeit diente uns ein Paper [**chenEa2012**], welches einen Algorithmus für die Rekonstruktion metrischer Graphen beschreibt und dessen Richtigkeit beweist. Die Eingabe besteht dabei aus einem metrischen Raum (Y, d_y), der aus den Rohdaten konstruiert wird und dem ein metrischer Graph (X, d_x) zugrunde liegt, den es zu rekonstruieren gilt. Ziel des Algorithmus ist es, diesen zugrunde liegenden Graphen durch einen metrischen Graphen $(\hat{X}, d_{\hat{x}})$ anzunähern und dabei weitestgehend die Abstände von (X, d_x) in $(\hat{X}, d_{\hat{x}})$ beizubehalten. Zusätzlich zu (Y, d_y) wird auch ein Parameter r übergeben, der in einigen Schritten des Algorithmus eine wichtige Rolle spielt. Die Erstellung des Graphen $(\hat{X}, d_{\hat{x}})$ wird nun wie folgt durchgeführt:

1. **Kanten- und Knotenpunkte bestimmen**
   Zunächst wird festgestellt, welche der in (Y, d_y) enthaltenen Punkte in $(\hat{X}, d_{\hat{x}})$ zu einer Kante und welche zu einem Knoten gehören werden. Dafür wird um alle im Eingabegraphen enthaltenen Punkte ein Kreisring gelegt, wobei der innere Kreis des Kreisrings den Radius r und der äußere den Radius $\frac{5}{3}r$ hat. Für die Punktmenge in diesem Kreisring wird anschließend der Rips-Vietoris-Graph mit dem Parameter $\frac{4}{3}r$ erstellt. Das bedeutet, dass alle Punkte, die nicht mehr als $\frac{4}{3}r$ voneinander entfernt sind, zu einer Zusammenhangskomponente gefasst werden. Daraufhin wird der Grad des jeweiligen Knoten bestimmt, indem die Zusammenhangskomponenten des Rips-Vietoris-Graphen gezählt werden. Enthält der Kreisring um einen Knoten zwei Zusammenhangskomponenten, hat er Grad zwei und es handelt sich um einen Punkt, der im rekonstruierten Graphen $(\hat{X}, d_{\hat{x}})$ zu einer Kante gehören wird. Daher erhält er das Label *edge point*. Ist der Grad ungleich zwei, wird der Punkt zu einem Knoten gehören und wird daher als *preliminary branch point* gekennzeichnet.



   Diese Markierung erhalten diese Punkte lediglich vorläufig (*preliminary*), da im Anschluss alle Punkte innerhalb eines Abstands von $2r$ von einem *preliminary branch point* als *branch*

point gekennzeichnet werden. Hierbei werden auch die *preliminary branch points* zu *branch points*, womit sich die Punkte in (Y, d_y) nun in zwei Teilmengen $\mathbb{E}$ und $\mathbb{V}$ aufteilen lassen. In $\mathbb{E}$ sind dabei die *edge points* enthalten und in $\mathbb{V}$ die *branch points*. Der Vorgang wird mit Pseudocode in Algorithm 1 veranschaulicht.

Algorithm 1 Kanten- und Knotenpunkte bestimmen

```
for all y ∈ Y do
  R ← C_{5/3 r} \ C_r
  deg(y) ← # Zusammenhangskomponenten im Rips-Vietoris-Graphen_{4/3 r}(R)
  if deg(y) == 2 then
    y erhält das Label edge point
  else
    y erhält das Label branch point
  end if
end for
for all y ∈ Y do
  if y ist nicht weiter als 2r von einem preliminary branch point entfernt then
    y erhält das Label branch point
  end if
end for
```

2. **Struktur des Graphen rekonstruieren**
   Anschließend wird für die beiden Mengen $\mathbb{V}$ und $\mathbb{E}$ der Rips-Vietoris-Graph mit dem Parameter $2r$ erstellt. Die dabei entstehenden Zusammenhangskomponenten in $\mathbb{E}$ entsprechen nun den Kanten und jene in $\mathbb{V}$ den Knoten im Graphen $(\hat{X}, d_{\hat{x}})$. Um dessen Struktur zu vervollständigen müssen lediglich noch die Verbindungen zwischen Kanten und Knoten rekonstruiert werden. Dies geschieht, indem zwei Knoten aus $\mathbb{V}$ genau dann durch eine Kante $e \in \mathbb{E}$ verbunden werden, wenn sie in ihrer Zusammenhangskomponente Punkte haben, die zu Punkten der Zusammenhangskomponente von e einen kleineren Abstand als $2r$ haben.

3. **Metrik rekonstruieren**
   Schlussendlich müssen noch die Längen der Kanten bestimmt werden. Dafür wird jeder Kante in $\hat{X}$ als Länge der Durchmesser ihrer Zusammenhangskomponente, also der längste kürzeste Weg darin, zuzüglich $4r$ zugewiesen.

2 Organisation

2.1 Programmiersprache

Da allen Studenten im Team die objektorientierte Programmiersprache Java geläufig war und sie uns als durchaus geeignet für die Aufgabe schien, haben wir uns entschlossen, unser Programm darin zu schreiben.

2.2 Gruppenaufteilung

Für das Arbeiten an unserer Software haben wir uns in drei verschiedene Gruppen aufgeteilt. Die eine Gruppe hat sich mit der Vorverarbeitung der Rohdaten beschäftigt, im Wesentlichen also aus den Rohdaten den metrische Raum (Y, d_y) für die Eingabe des Algorithmus konstruiert. Dabei beschlossen wir, uns auf die Visualisierung von GPS-Daten und Schwarz-Weißbildern zu beschränken, sodass es vorrangig darum ging, diese beiden Datengrundlagen so aufzubereiten, dass die nächste Gruppe damit weiterarbeiten konnte. Die zweite Gruppe hat sich mit der Implementierung

des Algorithmus, genauer der Rekonstruktion von $(\hat{X}, d_{\hat{x}})$, in Java auseinandergesetzt, während die dritte für die Visualisierung des Graphen $(\hat{X}, d_{\hat{x}})$ zuständig war.

2.3 Kommunikation

Damit wir stets alle über die Arbeit unserer Teammitglieder informiert waren, haben wir unsere Ergebnisse auf der Hosting-Plattform *GitHub*[1] gespeichert und aktualisiert. Um anstehende Aufgaben für alle zugänglich zu dokumentieren haben wir zusätzlich die für Projektmanagement bestimmte Anwendung *Trello* [2] genutzt, bei der man ein virtuelles Notizbrett erstellen kann. Hier konnte jeder neue Aufgaben an eine virtuelle Pinnwand hängen und sich dafür als Bearbeiter eintragen. Zusätzlich konnten wir dort den Stand der Arbeit an diesen Aufgaben dokumentieren, wodurch stets jeder darüber informiert war, woran die anderen gerade arbeiteten.
Darüber hinaus haben wir per E-Mail kommuniziert und uns einmal pro Woche in der Universität getroffen, um die wichtigsten Ergebnisse der vergangenen und die Vorhaben für die kommende Woche zu besprechen und festzulegen.

3 Arbeit in den einzelnen Gruppen

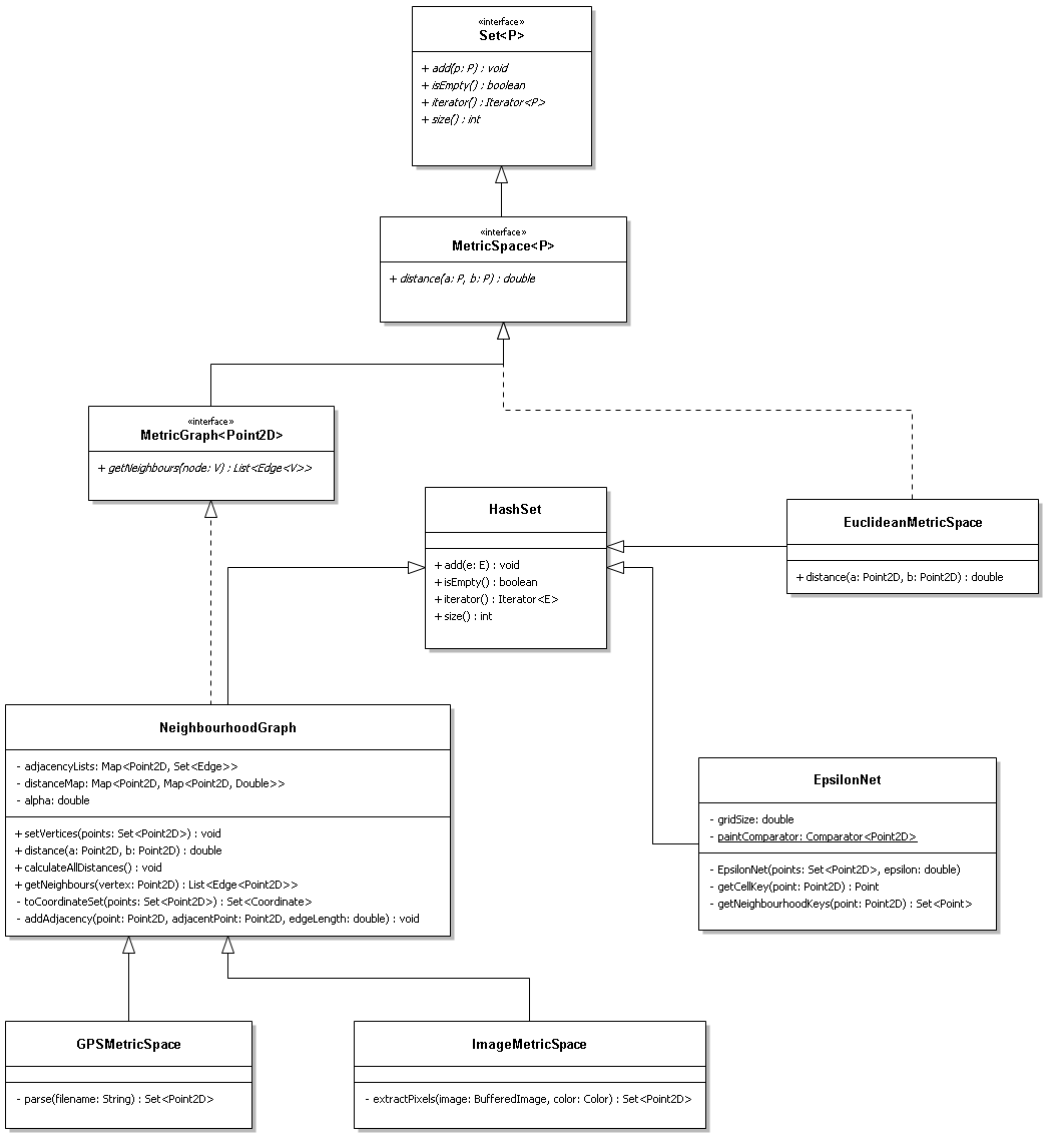
Im Folgenden soll nun ein Überblick darüber gegeben werden, wie die drei verschiedenen Gruppen gearbeitet haben. Die Gruppe, die sich mit der Vorverarbeitung beschäftigt hat, bestand aus zwei Studierenden, jene, die für die Graphenrekonstruktion verantwortlich war, aus drei und für die Visualisierung des rekonstruierten Graphen war ein Student verantwortlich.

3.1 Preprocessing

Die Kernaufgabe dieser Gruppe bestand in der Erfassung, sowie der Vorverarbeitung der Eingabedaten in eine für den Algorithmus passende Form.
Als Eingabe sind Schwarz-Weiß-Bilder, sowie GPS-Trace-Route-Dateien gedacht. Aus diesen müssen die Koordinaten der einzelnen Punkte extrahiert werden. Im Falle der GPS-Dateien wurde dazu ein Filter erstellt, welcher die angegebene Datei durchsucht und die Koordinaten (Längen- und Breitengrade) extrahiert. Sollte ein Bild angegeben werden, wird jeder Bildpunkt geprüft, wobei von jedem schwarzen die Pixelkoordinaten erfasst werden.
Da eventuell eine sehr große Punktmenge das Resultat sein kann, gibt es die Möglichkeit zur Reduzierung in Form der EpsilonNet-Klasse, welche durch Angabe eines Parameters (*epsilon*) redundante Punkte entfernt. Dabei wird ein Algorithmus verwendet, der unter Verwendung eines Gitters ein metrisches Epsilon-Netz in linearer Zeit berechnet.[3]
Die Punkte werden in einem Set platziert und sind so zur weiteren Verarbeitung zugänglich.
Anschließend ist es das Ziel, diese Punkte in einen metrischen Raum (*metric space*) umzuformen, d. h. die Punktmenge so zu strukturieren, dass wir möglichst schnell und einfach die paarweisen Abstände zwischen den Punkten berechnen können.
Dazu wird zuerst die Delaunay-Triangulierung der Punktmenge berechnet (mithilfe der DelaunayTriangulationBuilder-Klasse der JTS Topology Suite). Anhand dieser Triangulierung wird ein Nachbarschaftsgraph erstellt. Der Graph besteht aus allen Knoten der Triangulierung und allen Kanten, deren Radius einen bestimmten Wert (*alpha*) nicht überschreitet. (Der einzige Unterschied zum Alpha-Komplex ist, dass der Nachbarschaftsgraph keine Dreiecke enthält.) Als Datenstruktur werden dabei Adjazenzlisten verwendet.

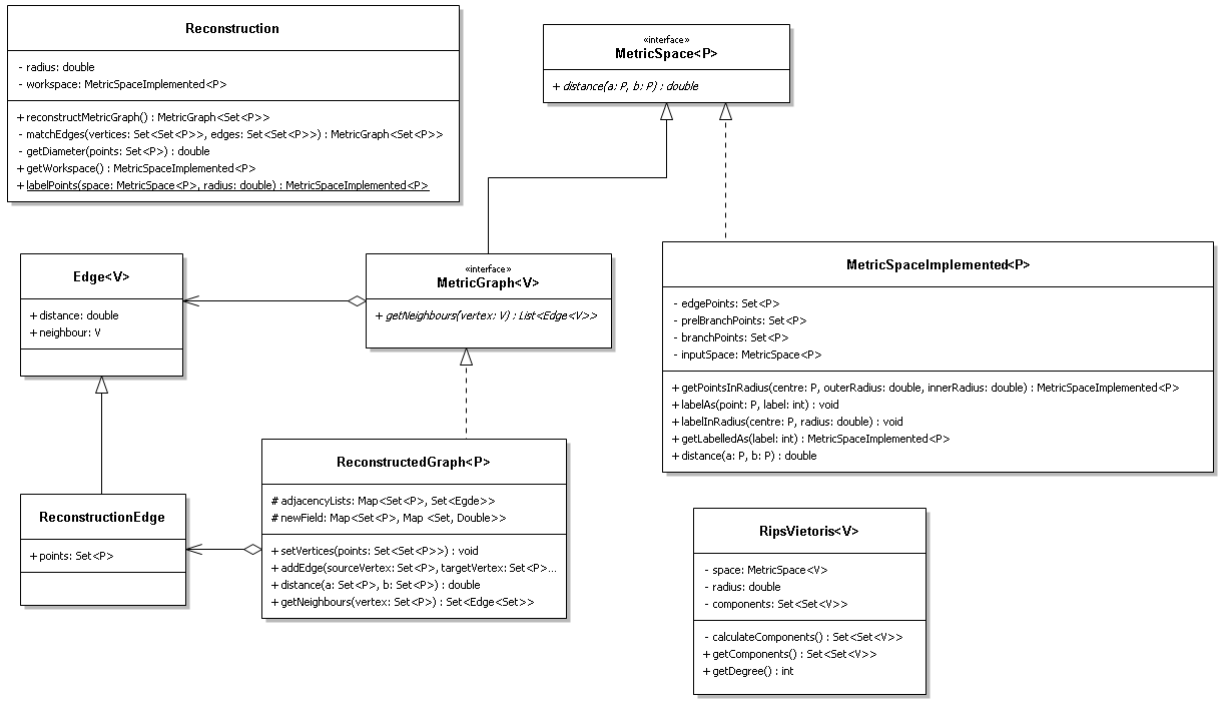
[1] `github.com`
[2] `trello.com`
[3] Sariel Har-Peled und Benjamin Raichel. Net and Prune: A Linear Time Algorithm for Euclidean Distance Problems. `http://web.engr.illinois.edu/~raichel2/aggregate.pdf`, 8. November 2012. (Der Algorithmus wird im Beweis zum Lemma 2.2 beschrieben.)

Die Berechnung der Abstände erfolgt dann mithilfe des Floyd-Warshall-Algorithmus zur Bestimmung des kürzesten Pfades. Da diese Berechnung lange dauert, berechnen wir alle Abstände einmal und speichern sie in einer zweidimensionalen Tabelle (realisiert durch verschachtelte HashMap-Objekte). Bei der Rekonstruktion des Graphen können wir dann einfach auf die schon berechneten Abstände zugreifen.
Nun liegen alle benötigten Daten für den eigentlichen Algorithmus vor und werden in Form eines MetricSpace-Objektes übergeben.



3.2 Rekonstruktion

Die Aufgabe der Rekonstruktion ist es, mithilfe des Algorithmus und der aus der Vorverarbeitung gelieferten Daten den gesuchten Graphen aus der Punktmenge zu rekonstruieren. Dazu haben wir den Algorithmus selbst in der Klasse Reconstruction implementiert. Der Algorithmus benötigte jedoch ebenfalls den Rips-Vietoris-Graphen über bestimmte Punktmengen; dieser wurde in der Klasse RipsVietoris implementiert. Da die Daten der Vorverarbeitung nur die Punkte sowie eine Distanzfunktion über diese Punkte gemäß der Schnittstelle MetricSpace umfasste, war es notwendig diesen MetricSpace zu erweitern, was durch die Klasse MetricSpaceImplemented geschieht. Dadurch, dass die Rekonstruktionsgruppe aus 3 Mitgliedern bestand, hat jeder die Programmierung einer der Klassen übernommen. Die Aufteilung war wie folgt:



1. MetricSpaceImplemented: Lea Helmers
2. Reconstruction: Moritz Walter
3. RipsVietoris: Mahmoud Kassem

Da die Klassen eng zusammenarbeiten, fand meist ein reger Austausch in den Gruppentreffen und via E-Mail statt. Getestet wurden die Klassen später zunächst soweit möglich einzeln und dann als Modul. Dabei wurden unter anderem auch Testeingaben und eine vorläufige Anzeige für metrische Räume entwickelt. Als Endergebnis dieses Moduls gibt es den fertig rekonstruierten Graphen, abrufbar durch reconstructMetricGraph(), welcher dann durch die Visualisierung angezeigt werden kann.

3.3 Visualisierung

Die Visualisierung dient drei Zwecke:

1. **Testen:** Sie erlaubt es uns auf einen Blick zu sehen, ob unsere Implementierung das liefert, was wir erwarten.
2. **Parameterauswahl:** Sie ermöglicht es dem Benutzer, unterschiedliche Parameter auszuprobieren und zu sehen, wie sich dadurch die Zwischen- und Endergebnisse ändern.
3. **Vergleich der Ein- und Ausgabe:** Am Ende sieht der Benutzer eine graphische Darstellung des metrischen Raumes, zusammen mit dem rekonstruierten metrischen Graphen.

Die graphische Benutzeroberfläche zur Visualisierung haben wir in zwei Fenster aufgeteilt: das Vorverarbeitungsfenster dient zum Testen der Vorverarbeitung und zur Auswahl der Parameter *epsilon* und *alpha* und das Rekonstruktionsfenster dient zum Testen unserer Implementierung des Algorithmus, zur Auswahl des Parameters *radius* und zum Vergleich der Ein- und Ausgabe.
Damit der Benutzer bei der Parameterauswahl nicht nach jeder Änderung der Parameter lange warten muss, bis er die Auswirkung sieht, haben wir Vorschau-Schaltflächen eingebaut. Im Vorverarbeitungsfenster wird beim Vorschau ein Nachbarschaftsgraph ohne Abstände berechnet. Wichtig

bei der Auswahl von *epsilon* und *alpha* sind nämlich nicht die Abstände zwischen den Knoten, sondern die Struktur des Graphen. Im Rekonstruktionsfenster werden beim Vorschau nur die Kanten- und Knotenpunkte bestimmt (die Kantenpunkte werden blau und die Knotenpunkte rot markiert). Zur Benutzerfreundlichkeit gibt es in beiden Fenstern eine Statusanzeige. Falls eine Berechnung lange dauert, kann der Benutzer dort den Fortschritt verfolgen. An dieser Stelle werden auch die Kantenlängen bzw. die Koordinaten angezeigt, wenn der Benutzer mit der Maus über Kanten oder Knoten des rekonstruierten metrischen Graphen fährt.

4 Testen

Das Testen erwieß sich anfangs als schwierig, denn benötigte Daten für ein Modul kamen meist aus einem anderen Modul und es war nicht auszuschließen, dass diese Daten bereits Fehler enthielten. Desweiteren wurde anfangs das Ergebnis eines Moduls direkt an das nächste weitergeleitet ohne eine zwischenzeitliche Veranschaulichung, so dass die Daten auf Fehler gesichtet werden mussten. Aus diesem Grund entwickelten wir in den einzelnen Gruppen Hilfsprogramme zur Visualisierung des entsprechenden Zwischenergebnisses sowie einige Sets von festen Test-Eingabedaten. Es war jedoch etwas zweischneidig, da anfangs diese Visualisierungsprogramme ebenfalls Fehler enthielten, so dass zum Beispiel bei der Anzeige des Ergebnisses der Graph an verschiedenen Achsen gespiegelt wurde. Nach beheben dieser Schwierigkeiten war es möglich die meisten Funktionen erstmals rudimentär zu testen bis dann schließlich einer aus der Gruppe das komplette Modul im Zusammenspiel nochmals getestet hat.
Nachdem dies geschehen war, führten wir die Module zusammen zu einem Gesammtprogramm. Dabei sind unter anderem die Teilvisualisierungen mit in die Endvisualisierung eingeflossen. Nun wurde nochmals das komplette Programm als solchens getestet, wobei hier kaum noch Schwierigkeiten auftraten, da die meisten Fehler durch die vorhergehenden Tests beseitigt wurden.

5 Schwierigkeiten und Verbesserungsvorschläge

Eine grundsätzliche Schwierigkeit war das Testen fertiger Module. Aufgrund der Datenfluss-ähnlichen Struktur des Gesamtprogramms kann ein kleiner Fehler in der Vorverarbeitung massiven Einfluss auf das Endergebnis nehmen. Da dieses Ergebnis graphischer Natur ist, war wirkliches Testen aber erst nach Erstellung einer graphischen Ausgabe (Prototyp) möglich. Im Falle des Epsilon-Netzes ließ sich so beispielsweise zuverlässiges Aussortieren von nicht benötigten Punkten erst spät testen, da anhand der blanken Koordinaten die Auswirkungen sich nicht gut genug erkennen ließen und die Menge als ganzes nur schwer zu erfassen war.
Ideen für weitere Features bzw. Änderungen an Algorithmen gab es einige, allerdings machte der Zeitmangel die Umsetzung zu Nichte. So waren als Verbesserungen z. B. eine Einbeziehung des Zeitstempels von GPS Daten zur Verbesserung des Nachbarschaftsgraphen, die Abstandsberechnung mithilfe des Algorithmus von Dijkstra statt Floyd-Warshall um die Laufzeit zu senken, eine automatisierte Möglichkeit zur optimalen Bestimmung der Parameter oder auch die Berechnung von Orthodromen um den Abstand in km anzugeben, angedacht.

6 Zusammenfassung