Automatische Visualisierung von Schiffsrouten

Andreas Lorer[1] and Kevin Klugmann[1]

[1] Hochschule Ravensburg-Weingarten, Doggenriedstr. 88250 Weingarten
contact@andreaslorer.de, kevin.klugmann@gmail.com

Abstract

In dieser Arbeit soll aufgezeigt werden, wie eine automatisierte Visualisierung von Kreuzschifffahrtsrouten realisiert werden konnte. Anders als im Straßenverkehr bewegen sich Schiffe nicht in einem fest begrenzten Verkehrsnetzwerk, sondern größtenteils auf offenen Flächen. Daraus ergeben sich zahlreiche Probleme für eine maritime Routenfindung. In unserem Ansatz erfolgt das Routing Grid-basiert auf einem HTML5-Canvas. Mittels diesem Grid soll die Route über eine variable Anzahl an Hafenpunkten durch Verwendung des A*-Algorithmus gefunden werden. Es stellte sich heraus, dass eine voll automatische Kartenvisualisierung möglich ist, diese jedoch noch Potenzial für zukünftige Verbesserungen bietet.

Einleitung

Während es im Bereich des Straßenverkehrs relativ einfach ist, eine Route zu berechnen[1] und diese in einer interaktiven Karte zu visualisieren, gibt es zum jetzigen Zeitpunkt keine Möglichkeit, dies für Schiffsrouten zu tun. Abbildung 1 zeigt eine Visualisierung, wie sie momentan in einer Karte zum Einsatz kommt.

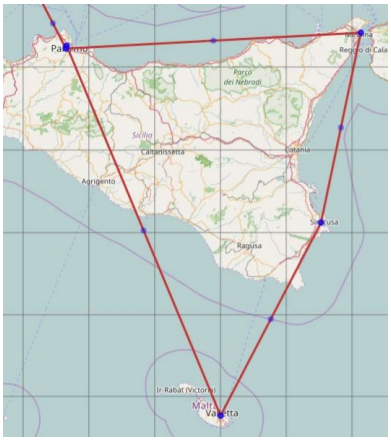

Abbildung 1: Beispiel einer Kartenvisualisierung

[1] Via Google-Maps, Bing-Maps oder weiteren Kartenanbieter

Die Grafik veranschaulicht bereits deutlich das zu lösende Problem: Die einzelnen Häfen sind nur durch gerade Linien verbunden, die nicht zwischen Wasser und Land unterscheiden. Dadurch führen Routen auch über Landmassen. Unser Ziel war es, eine ästhetische und möglichst realistische Darstellung der Fahrtrouten zu generieren. Dieses Visualisierungstool soll einem Buchungsportal für Kreuzfahrten zur Verfügung stehen. Bei der Zielgruppe handelt es sich um Personen, die eine Kreuzfahrt buchen möchten. Ihnen soll die ungefähre Fahrtroute optisch ansprechend präsentiert werden. Die Route hat also keinerlei Relevanz für Nautiker oder Kapitäne. Wassertiefe oder nicht befahrbare Abschnitte sind deshalb irrelevant und werden von uns nicht beachtet. Entscheidend ist in unserem Fall auch nicht die Berechnung des kürzesten Weges.

Momentan werden Routen für die Kreuzschifffahrt vorwiegend auf zwei Arten visualisiert.

Abbildung 2: Visualisierung über Pin's (links) oder händisch gezeichnet (rechts)

Wie in Abbildung 2 zu sehen ist, werden entweder nur einzelne Pin's auf die jeweiligen Häfen gesetzt oder es werden komplexe Vektorgrafiken händisch erzeugt. Letzteres ist sehr aufwändig und dementsprechend kostenintensiv. Einzig `http://www.cleancruising.com.au/` hatte eine visuell überzeugende Karte (Abbildung 3) mit einem ansprechendem Routing.

Für dieses Problem der automatischen Kartengenerierung wurde ein Algorithmus entwickelt, welcher in der Lage ist, zwischen Land und Wasser zu unterscheiden und damit eine

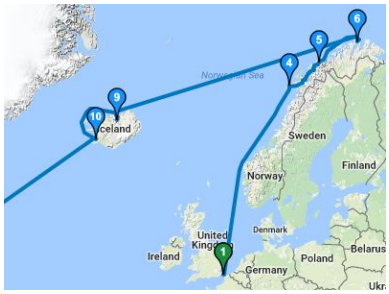

Abbildung 3: Kartenvisualisierung von cleancruise.com

visuell ansprechendere Visualisierung ermöglicht. Als Eingabe wird eine Liste an Hafen-Positionen benötigt, wovon eine Route als Ausgabe generiert wird. Eingabe und Ausgabe erfolgen dabei im `geojson-Format` [4].

Andere Bedingungen für einen Algorithmus sind oftmals seine lineare Laufzeit $O(n)$, die für Echtzeitsysteme benötigt werden. Unser Algorithmus ist allerdings nur für den Einsatz im Backend vorhergesehen. Dort werden die Routen für die Anzeige im Frontend bereits vorberechnet und zur Wiederverwendung abgespeichert. Eine lineare Laufzeit ist dadurch nicht von Nöten und war zu keiner Zeit eine Anforderung.

Relevante Publikationen

Es finden sich bisher nur wenige Publikationen, die sich mit dem Thema Schiffsrouten via Web-Plattform auseinandersetzen.

Die meisten Systeme, die Routenberechnungen für die Schifffahrt durchführen, benutzen eine graph-basierte Grid-Struktur oder eine Abwandlung davon. [1] entwickelte beispielsweise eine `Novel Grid Structur` (siehe Abbildung 4). In dieser Datenstruktur werden für jeden Knotenpunkt (Node) maritime Informationen abgespeichert und Gewichtungen für jede Kante (Edge) berechnet.

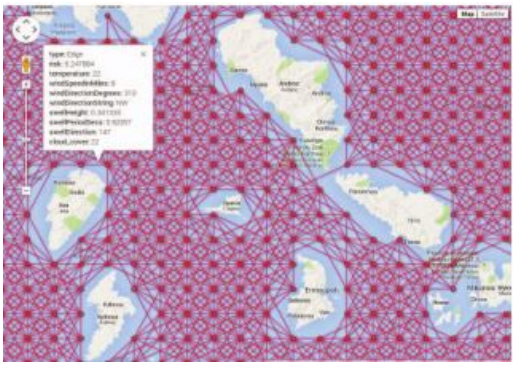

Abbildung 4: Novel Grid Structure (Abbildung von [1])

Die Motivation dahinter ist dabei meist ein Echtzeitsystem oder eine Echtzeitkalkulation der Route. Eine Grid-Struktur, die wie ein Graph aufgebaut ist, beschleunigt den Suchprozess des A*-Algorithmus erheblich[6]. Die Grid-Struktur wird dabei aus den statischen maritimen Informationen und zusätzlichen dynamischen Daten (meterologische Wetterdaten über Windstärke, Windrichtung etc.) für die Routenberechnung ergänzt [1, s. 2]. Für unseren Algorithmus wäre es zwar hilfreich gewesen, eine solche Datenstruktur zu haben, allerdings verfügten wir über keinerlei Zugriff auf solche Datenquellen, da diese unseren Recherchen nach nicht öffentlich (kostenfrei) verfügbar sind.

Die Pixel-Karte

In diesem Abschnitt möchten wir unsere Arbeit an dem Routing Algorithmus für Kreuzschifffahrten genauer erklären und den Problemlöseprozess aufzeigen. Unsere Lösung basiert zunächst auf der Generierung einer Pixel-Karte. Diese Karte dient uns als Grundlage für das Routing mittels A*-Algorithmus. Die Pixel-Karte aus Abbildung 5 wird durch Kartendaten von Natural Earth[2]auf einem HTML5-Canvas erzeugt.

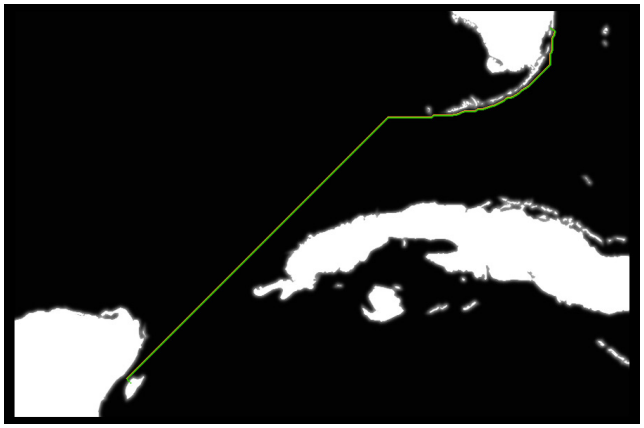

Abbildung 5: Generierte Pixel-karte für eine Routenberechnung

Der Maßstab der Karte beträgt dabei 1:10m (Millionen) und liegt im Datenformat `.shp` (für Shapefile) vor. Die Karte wurde mittels eines Tools (QGIS) in `geojson` umgewandelt. Höher aufgelöstes Kartenmaterial ist im Netz an verschiedenen Stellen zu finden[3], haben allerdings Dateigrößen von bis zu 500 MB. Da wir solch hohe Datenmengen client-seitig nicht verarbeiten konnten, ist für uns die Karte mit dem Maßstab 1:10m ein solider Kompromiss zwischen nötiger Genauigkeit entlang der Küstenlinie (Inseln & Fjorde) und Datengröße.

Nachdem die Pixel-Karte auf dem Canvas gezeichnet ist, dient uns die Canvas-Auflösung als Grid-Raster. Höhere

[2]Natural Earth 1:10m coastline `http://bit.ly/2kbyYIV`

[3]z.B. bei OpenStreetMap `http://openstreetmapdata.com/data/coast`

Auflösungen ermöglichen es dabei, mehr Details darzustellen. Das ist besonders für Fjorde und Inseln nötig, da diese bei einer zu niedrig aufgelösten Pixel-Karte nicht dargestellt werden können. Eine Auflösung von 512^2 oder 1024^2 wurde bei unseren Routenkalkulationen als ein solider Wert befunden und erzeugte gute Ergebnisse. Je höher die gewählte Auflösung, umso mehr Arbeitsspeicher wird allerdings auch benötigt. Da wir client-seitig eine Limitierung des verfügbaren Arbeitsspeichers durch den Browser haben, konnten höhere Kartenauflösungen nicht getestet werden. Sobald eine Portierung auf eine server-seitige Lösung via Node.js erfolgt, sollte dies allerdings möglich sein.

Pixel-Karte in Graustufen

Der HTML5 Canvas bietet die Möglichkeit, die Farbinformationen für jeden Pixel auszulesen. Ein schwarzer Pixel bedeutet dabei, dass es sich um Wasser handelt und ein weißer Pixel steht folglich für Landmasse, also ein Hindernis. Wird jedoch lediglich zwischen Land und Wasser unterschieden, führt dies zu einem Problem: Die Schiffe fahren genau entlang der Küstenlinie, da dies dem kürzesten Weg von A nach B entspricht (siehe Abbildung 6).

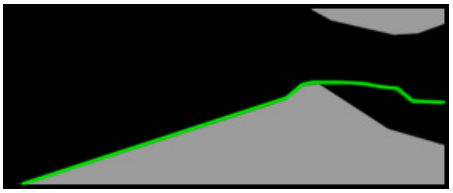

Abbildung 6: Pixel Karte vor dem Blur-Effekt.[4]

Da Schiffe allerdings nicht genau auf der Küstenlinie, sondern nur in Küstennähe fahren, würde dies den Vorstellungen von einer ästhetischen und möglichst realistischen Darstellung widersprechen.

Um dieses Problem zu lösen und eine glaubhaftere Schiffsroute zu generieren, musste die berechnete Route von der Küstenlinie versetzt werden. Dazu haben wir die Pixel-Karte in zwei Schritten einer Manipulation unterzogen, welche in Abbildung 7 zu sehen ist. Erstens wurde die Pixel-Karte 3 mal gezeichnet und dabei die Pinselgröße des Canvas jedes mal verringert (auf 6, 3, 0.1). Bevor dann in der letzten Iteration das Land gezeichnet wurde, erfolgte noch ein Gaussian Blur Filter[9]. Dieser bewirkt, dass die sehr harten Kanten, die beim Zeichnen entstehen, geglättet werden und ein homogener Übergang erreicht wird.

Nach der Anwendung des Blur Filters wurde zuletzt die Landmasse eingezeichnet, damit diese ihre scharfe Kante behält und nicht durch den Blur in Unschärfe verschwimmt.

[4]Das Land wurde ausgegraut, um die Route besser sichtbar zu machen

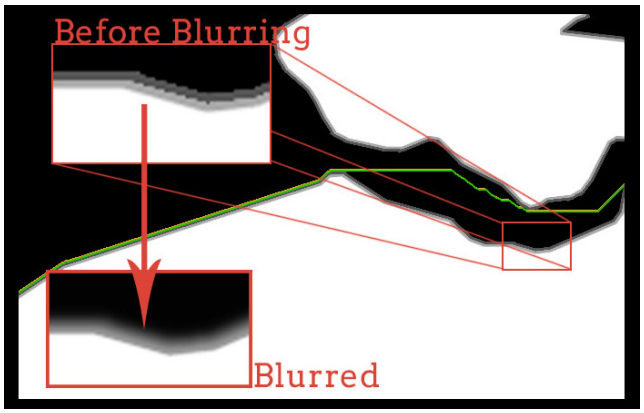

Abbildung 7: Anwendung eines Gaussian-Blur Filters

A*-Kantengewichtung

Durch dieses Verfahren erhielten wir eine Pixel-Karte, die nicht nur Schwarz und Weiß, sondern auch Graustufen-Werte zwischen 0-255 beinhaltet. Der von uns verwendete A*-Algorithmus[3] ermöglicht es, eine Kantengewichtung für die Routenberechnung mit einfließen zu lassen. Eine Kantengewichtung führt dazu, das gewisse Wege bevorzugt werden können. In unserem Fall hat die Farbe Schwarz die geringste Gewichtung und ist damit der absolut bevorzugte Weg, den der A*-Algorithmus wählen kann. Die Graustufen-Werte haben nun eine logarithmische Gewichtung die durch den RGB-Wert der Farbe errechnet wird.

$$weight = \lceil rgbValue * \ln(rgbValue)/100 \rceil$$

Damit der A*-Algorithmus das Land nicht als Abkürzung verwendet, haben RGB-Werte über 250[5]eine sehr viel höhere Gewichtung. Dadurch soll erreicht werden, dass das Land nur als aller letzte Möglichkeit überquert werden darf. Zum Beispiel dann, wenn die GPS Koordinate eines Hafens, innerhalb eines Kanals oder Flusses liegt, der im Kartenmaterial nicht enthalten ist. Deshalb erfolgt die Errechnung für RGB-Werte über 250 mittels der Gewichtung $weight = rgbValue^3$. Ein RGB Wert von 255 würde also eine Gewichtung von $255^3 = 16581375$ bedeuten. Wohingegen ein RGB Wert von beispielsweise 120 nur $weight = \lceil 120 * ln(120)/100 \rceil = \lceil 5.774 \rceil = 6$ ergeben würde. Die Gewichtung der Grauwerte zwischen 1-249, werden folglich auf einen Wertebereich von 1-14 abgebildet.

Wie in Abbildung 8 zu sehen ist, beeinflusst nun die Gewichtung der RGB Werte die Wahl der eingeschlagenen Route. Dadurch wird die Route weg von der Küstenlinie „geschoben“ und visualisiert nun zunehmend das reale Verhalten einer Schiffsroute.

[5]Werte die schon sehr nahe an der Farbe Weiß liegen.

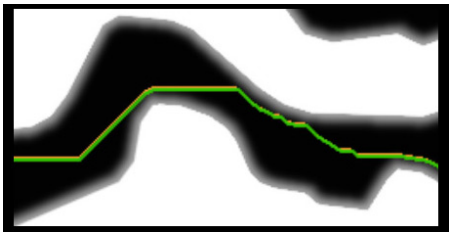

Abbildung 8: Ergebnis der Routenberechnung mit Blur Filter Pixel-Karte

Verbesserung der Routing Genauigkeit

Zuerst erfolgten unsere Berechnungen mithilfe einer einzigen Pixel-Karte. Dies funktionierte sehr gut, solange die einzelnen Häfen einer Route relativ nahe zusammen lagen.

Für weitläufigere Routen war die Kartenauflösung jedoch zu grob, da sehr weit weg gezoomt werden musste, um alle Hafenpunkte auf der Karte zu erfassen. Um dieses Problem zu lösen, berechneten wir schließlich jeden Abschnitt einer Route einzeln. Bei einer Route über die Punkte A, B und C werden also die Routen AB und BC separat berechnet und dann für jeden Abschnitt eine eigene Pixel-Karte erzeugt.

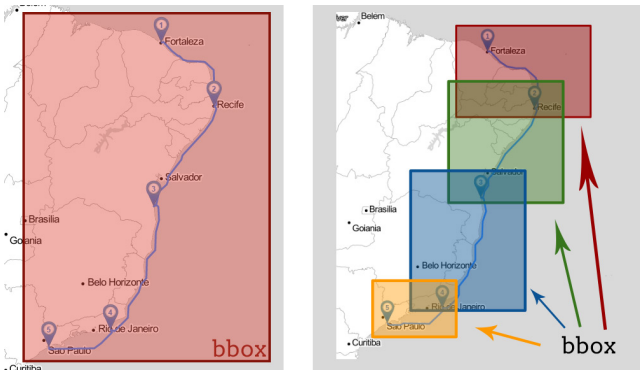

Abbildung 9: Unterteilung der Route in einzelne Abschnitte (rechts), anstatt wie zuvor, in einen einzelnen Abschnitt (links)

Um dies zu ermöglichen, wird für jeden Abschnitt eine `bbox` (Bounding Box) für den jeweiligen Routenabschnitt berechnet (siehe Abbildung 9). Diese bbox wird dann nochmals ein wenig vergrößert, um einen Puffer zu haben und dient dann dem A*-Algorithmus für die Wegfindung. Dadurch wird erreicht, dass Routenabschnitte, die weiter von einander entfernt liegen, eine andere Zoomstufe besitzen als Routen, die näher zusammen verlaufen. Abbildung 10 zeigt zwei Pixel-Karten, die für eine Fahrtroute in Norwegen berechnet wurden. Wie zu sehen ist, weisen sie ganz unterschiedliche Zoom-Stufen auf, was eine Routenführung durch die Fjorde erst möglich macht.

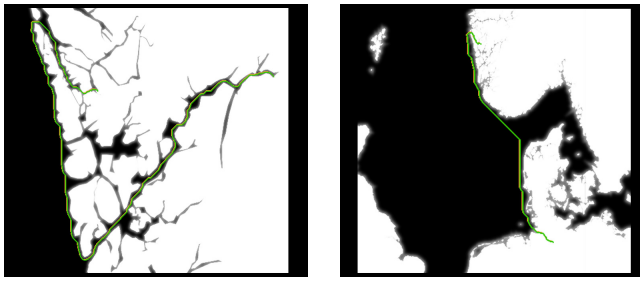

Abbildung 10: Zwei Beispiele für eine Pixel-Karte die für den jeweiligen Abschnitt unterschiedliche Zoom-Stufen aufweisen.

Karten Rasterisierung

Da das Kartenmaterial die ganze Erde umfasst, mussten Optimierungen gefunden werden, damit die Berechnungszeiten für unsere Route in einem akzeptablen Rahmen bleiben.

In einem ersten Ansatz verfolgten wir das Ziel, die Polygone unseres Kartenmaterials auf die Größe der bbox zuzuschneiden. Dies führte zu allerlei Problemen, die nicht gelöst werden konnten. So hatte beispielsweise die Library zum Zuschneiden an sich schon einen für uns kritischen Fehler[6], sodass eine andere Lösung gefunden werden musste.

Der zweite Ansatz bestand darin, nur diejenigen `geojson-Features` auszuwählen, die für die jeweilige Routenberechnungen relevant sind (Abbildung 11 rot stellt die Auswahl des Features dar).

Abbildung 11: Auswahl der relevanten geojson-Features via bbox

Wie zu sehen ist, würde dies für eine Route in Großbritannien ganz passabel funktionieren. Abbildung 12 zeigt allerdings, dass es Probleme gibt, sobald eine Route zum Beispiel nicht nur in Großbritannien, sondern auch in Bereiche benachbarter Staaten reichen würde.

Da unser Kartenmaterial aus der Küstenlinie besteht, ist das Feature für Europa gleichzeitig auch mit Asien, Afrika

[6]Stand 14.02.2017: `https://github.com/w8r/martinez/issues/11`

Abbildung 12: Problematik wenn Auswahl nur über Features erfolgt

usw. verbunden, sodass es nicht möglich ist, einzelne Bereiche wie beispielsweise „nur Deutschland und Dänemark“ zu selektieren. Dies hatte zur Folge, dass immer noch sehr lange Rechenzeiten benötigt wurden, da der `Overhead` nach wie vor enorm war. Abhängig von der Länge und der Position der Route wurden Rechenzeiten von bis zu einer Stunde benötigt.

Infolgedessen haben wir das Kartenmaterial in einzelne Teile aufgeteilt.[7]

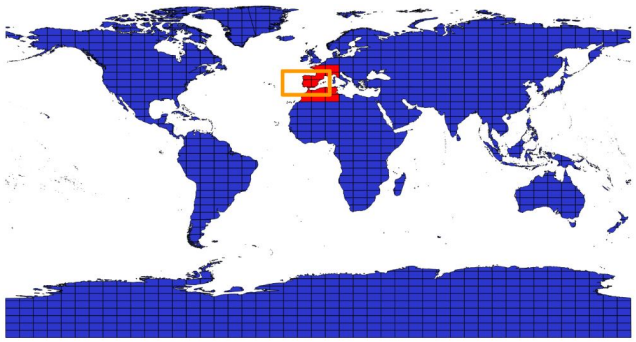

Abbildung 13: Zerteilung der Karte in 2025 Tiles

Das ermöglicht es uns, nur noch diejenigen Features auszuwählen, die für unsere Route benötigt werden. Dadurch konnte das Berechnen der Pixel-Karte erheblich beschleunigt werden. Durchschnittlich haben wir Routen mit 6 Hafenpunkten, bei welchen die Berechnungszeit im Schnitt bei 37sec liegt.

Ergebnis

Das Ergebnis unserer Arbeit ist ein Tool, welches mittels Webtechnologien erstellt wurde und dessen Algorithmus eine Routenberechnung durchführt. Unser Tool ist in Abbildung 14 zu sehen und im Web unter[8]abzurufen.

[7]Verwendet wurde dabei QGIS[7], zusammen mit dem `Grid-Splitter` Plugin.

[8]Ship-Routing Tool: `https://andi-lo.github.io/ship-cruising/dist/`

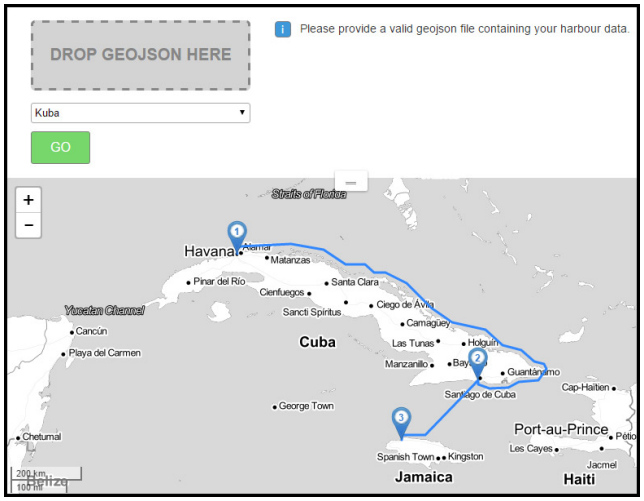

Abbildung 14: User Interface unseres Tools

Es besitzt ein interaktives Interface, das ein Dropdown-Menü zum Auswählen von vordefinierten Schiffsrouten und eine Drag & Drop Funktionalität beinhaltet. Mittels Drag & Drop können valide[9]geojson-Dateien zuerst hochgeladen und danach mittels „Go-Button“ direkt die Route berechnet werden. Eine Route wird dabei als geojson-Objekt vom Typ `FeatureCollection<Point>` (siehe Anhang: Listing 1) spezifiziert. Nachdem die Route berechnet wurde, wird sie in die Karte eingezeichnet und weitere Interaktionen wie z.B. Pinching oder Zoom sind möglich. Nach der Berechnung der Route erfolgen weitere Optimierungsschritte. Zuerst werden ähnlich verlaufende Routenabschnitte zusammengefügt, dann erfolgt eine Simplifizierung der Route. Das bedeutet, dass die Anzahl an berechneten Routen-Punkten verringert wird. Zuletzt erfolgt noch das Glätten (smoothing) der Route mittels Catmull-Rom Interpolation.

Fazit

Es gibt mehrere Möglichkeiten, um sowohl unser Tool als auch das visuelle Ergebnis weiter zu verbessern.

Schnellerer Suchalgorithmus

Für Anwendungen, die den A*-Algorithmus auf einer gridbasierten Struktur verwenden, gibt es verschiedene Optimierungsmöglichkeiten. Sowohl eine Änderung der Heuristic, als auch „Map Preprocessing“.

Ein Beispiel: Wird ein Schritt nach Norden und dann nach Osten gegangen, dann ist dies meistens dasselbe, wie wenn zuerst nach Osten und dann nach Norden gegangen wird. Graph-Algorithmen kennen allerdings weder „Norden“ noch „Osten“ und müssen deshalb beide Fälle ausprobieren.[6, vgl.] Dieses Verhalten nennt man „Path Symmetries“.

[9]Eine Validierung der geojson-Datei auf Korrektheit erfolgt dabei nicht

„Two grid paths are symmetric if they share the same start and end point and one can be derived from the other by swapping the order of the constituent vectors.“[5, p. 1]

In [5] wird der „RSR-Algorithmus“ (Rectangular Symmetry Reduction) als Pathfinding Algorithmus beschrieben, der diese Symmetrien aufheben soll.

„RSR can be described as a pre-processing algorithm which identifies symmetries by decomposing the grid map into empty rectangular regions.“[5, p. 2]

„[...] We observed that in most cases RSR can consistently speed up A search by a factor of between 2-3 times (Baldur's Gate), 2-5 times (Adaptive Depth) and 5-9 times (Rooms). In some cases the gain can be much higher: up to 30 times.“*[5, p. 4]

Die wohl umfangreichste Javascript Pathfinding Library[10]hat allerdings bis jetzt noch keine fertige RSR Implementierung. RSR ist in seiner ursprünglichen Form allerdings auch nur für einheitlich gewichtete Graphen verwendbar und es müsste eine Möglichkeit gefunden werden, RSR auf gewichtete Graphen zu übertragen. Dies wäre allerdings ein Forschungsfeld für sich.

Änderung der Heuristik

Der Sinn von einer Heuristik für den A*-Algorithmus ist es, eine Abschätzung für die Länge des kürzesten Pfads zu geben. Je besser die geschätzte Distanz dem kürztesten Pfad entspricht, umso schneller ist der A*-Algorithmus. Momentan wird eine Heuristik eingesetzt, die diagonale Bewegungen erlaubt. Die Heuristik könnte aber auf verschiedene Arten verbessert werden. In [2] wird ein Verfahren vorgestellt, bei dem die Grid-Map vor dem Pathfinding analysiert wird, um eine bessere Heuristik zu generieren.

In „A New Method of Ship Routing on Raster Grids, with Turn Penalties and Collision Avoidance“[8] wird die Heuristik dem Verhalten eines Schiffes angenähert. So sind Kursänderungen immer mit einem Verlust an Geschwindigkeit oder einer Verlängerung der Route verbunden. Dieses Verhalten wird nun auch durch die Heuristik wiedergespiegelt und Kursänderungen sind mit „Kosten“ verbunden, sodass sie, wenn möglich, zu vermeiden sind. Dieses Verhalten könnte sich visuell auch positiv auf unseren Algorithmus auswirken, da momentan der von A* gefundene initiale Weg sehr kantig verläuft.

[10]Pathfinding.js:`https://github.com/qiao/PathFinding.js`

Quad-Tree Preprocessing

Es wäre darüber hinaus auch möglich, die Pixel-Karte in quadratische Regionen zu unterteilen. Große, offene Flächen könnten damit durch nur wenige Quadrate repräsentiert werden und unregelmäßige Kanten entlang der Küste würden durch viele kleine Quadrate entstehen. Eine Implementierung ist zum Beispiel hier zu sehen: `https://www.youtube.com/watch?v=95aHGzzNCY8`. Weitere Optimierungsmöglichkeiten für den A*-Algorithmus sind unter [6] zu finden.

Vorberechnung der Pixel-Karte

Um ein noch besseres Ergebnis zu erreichen, wäre es außerdem denkbar, einige Vorberechnungen zu tätigen. Anstatt die Pixel-Karte für jede Route neu zu berechnen und zu zeichnen, wäre es auch möglich, diese vorab einmalig zu errechnen. Die Auflösung müsste allerdings entsprechend hoch sein. Um die Effizienz zu steigern, könnte die gespeicherte Karte wiederum in Teile zerlegt werden. Dadurch könnte man bei einer Berechnung auf eine bereits vorliegende Karte zurückgreifen und sich hierbei die jeweiligen Berechnungskosten ersparen.

Letztlich könnte es auch zielführend sein, das Kartenmaterial von den diversen Kartenanbieter zu verwenden. Diese sind bereits für alle Zoomstufen vorhanden und sehr präzise. Was ihnen allerdings fehlen würde, wäre der „Blur-Effekt“, den unsere Karte verwendet, um den Routenverlauf von der Küstenlinie zu verschieben.

Literatur

[1] A. MAKRYGIORGOS, I. A. VETSIKAS, S. P. Accelerating Multi-objective Ship Routing Using a Novel Grid Structure and a Simple Heuristic. Tech. rep., Institute of Informatics and Telecommunications NCSR ”Demokritos”, 2015.

[2] GOLDBERG, A., AND HARRELSON, C. Computing the shortest path: A* search meets graph theory. Tech. rep., Vancouver, Canada, July 2004.

[3] GRINSTEAD, B. Javascript-astar, 2014. `https://github.com/bgrins/javascript-astar`.

[4] H. BUTLER, M. DALY, A. D. S. G. S. H. T. S. The GeoJSON Format. RFC, IETF Trust, August 2016.

[5] HARABOR, D. Fast Pathfinding via Symmetry Breaking. Tech. rep., NICTA and The Australian National University, 2012. `http://users.cecs.anu.edu.au/~dharabor/data/papers/harabor-aigamedev12.pdf`.

[6] PATEL, A. Grid pathfinding optimizations, March 2014. `http://www.redblobgames.com/pathfinding/grids/algorithms.html`.

[7] QGIS. Qgis, 1991. `http://qgis.org/`.

[8] SZLAPCZYNSKI, R. A New Method of Ship Routing on Raster Grids, with Turn Penalties and Collision Avoidance. 27 – 42.

[9] VIGOUR.IO. ctx-blur, 2016. `https://github.com/vigour-io/ctx-blur`.

Anhang

Beispiel einer Route mit 3 Wegpunkten (Häfen) im `geojson-Format`

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [
          -82.36,
          23.14
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [
          -75.87,
          19.98
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [
          -77.93,
          18.46
        ]
      }
    }
  ]
}
```

Listing 1: Beispiel Route: Kuba