Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: 66043

Bearbeiter dieser Aufgabe:

Imanuel Fehse

07.04.2023

Inhaltsverzeichnis

Lösungsansatz

Das der Aufgabe zugrundeliegende Problem ist auf dem **„Traveling Salesman Problem“** (kurz **TSP**) basierend [Q1]. Hierbei soll der kürzeste Weg gefunden werden, der alle Punkte miteinander verbindet, wobei kein Punkt (bis auf den Startpunkt) doppelt „angefahren“ werden darf. Eine gängige Methode ein solches Problem zu lösen ist der **„Nearest Neighbour Algorithm“** (im Folgenden als **NNA** abgekürzt) [Q2]. Wie es der Name schon verrät, sucht dieser Algorithmus immer nach dem am nächsten liegenden Nachbarpunkt. Auf diese Weise wird ein möglichst kurzer Weg gefunden (jedoch nicht der kürzeste Weg), der schlussendlich nach einigen Wiederholungen dieses Suchprozesses nach dem am nächsten liegenden Nachbarpunkt alle Punkte miteinander verbindet.

Diese Lösungsmethode des NNA lässt sich allerdings nicht direkt auf Aufgabenstellung übertragen. Diese beinhaltet nämlich eine Bedingung (Winkel, der nicht überschritten werden darf), die beim Lösen des TSP durch den NNA nicht direkt berücksichtig werden kann. Aus diesem Grund wird das Problem in Teilprobleme zerlegt, die nacheinander gelöst werden und so die Aufgabe im Gesamten lösen.

Teilproblem 1 – Abbiegewinkel

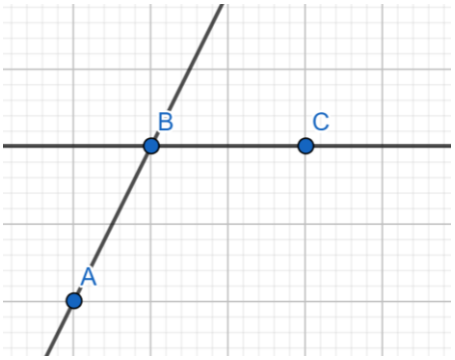
Die Aufgabenstellung gibt vor, dass der Pilot zwischen zwei Punkten nur geradeaus fliegen darf und nur an den Punkten abbiegen darf. Im Folgenden wird der Winkel, in dem der Pilot an diesen Punkten abbiegt als **Abbiegewinkel α** beschrieben. Um wie von der Aufgabenstellung gefordert einen Abbiegewinkel von 90° nicht zu überschreiten, muss dieser zunächst berechnet werden.

[Abb. 1] Positionierung des Abbiegewinkels α

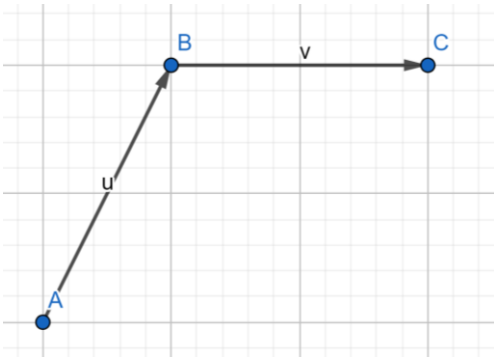
Hierfür ergeben sich zwei Möglichkeiten, die mit den vorhandenen Informationen arbeiten. Diese betrachten beide drei verschiedene Punkte, den letzten Punkt, an dem sich der Pilot befand, den aktuellen Punkt und den nächsten Punkt.

Methode 1: Betrachten der Verbindungen der Punkte als Geraden mit anschließender Schnittwinkelberechnung. Bei der Berechnung des Schnittwinkels muss die Laufrichtung, also von welchem Punkt zu welchem „geflogen“ wird, berücksichtig werden.

Methode 2: Berechnung der Vektoren, die die Punkt in korrekter Reihenfolge miteinander verbinden mit anschließender Schnittwinkelberechnung der Vektoren.



[Abb. 2] Methode 1



[Abb. 3] Methode 2

Um das erste Teilproblem zu lösen, bietet sich die zweite Methode eher an, da die erste mit einem höheren Aufwand verbunden ist. Bei der zweiten Methode werden die Koordinaten der Punkte nur zur Berechnung der Vektoren benötigt, da anschließend alle nötigen Informationen zur Richtung in den Vektoren enthalten sind.

Teilproblem 2 – Möglichst kurzer Weg

Beim TSP wird der NNA verwendet um das Problem des „kürzesten“ Weges (möglichst kurzer Weg) zu lösen. Dieser lässt sich jedoch aufgrund des Abbiegewinkels, welcher berücksichtigt werden muss, nicht direkt auf die Aufgabe anwenden. Der NNA kann jedoch trotzdem eingesetzt werden, um bei der Lösung des Problems zu helfen. Durch ihn werden die von einem Punkt ausgehenden Distanzen zu allen anderen Punkten berechnet, die der Pilot noch nicht „angeflogen“, also noch nicht besucht hat. Diese Punkte werden der Distanz nach Sortiert und anschließend darauf geprüft, ob beim anfliegen des Punktes vom aktuellen Punkt aus, ein Abbiegewinkel von 90° überschritten wird. Ist dies der Fall, wird der Punkt nicht weiter berücksichtigt (in die Berechnungen nachfolgender Punkte aber wieder mit einbezogen). Der Pilot fliegt zum nächstgelegenen Punkt, der diesen Abbiegewinkel nicht überschreitet.

Sind diese Probleme gelöst und es wurde eine Route/ein Streckenverlauf berechnet, der den Bedingungen entspricht, soll das Ergebnis auf zwei Wegen ausgegeben werden:

1. Als **Grafik**, die die Flugroute des Piloten veranschaulicht. Dies hilft dem Piloten dabei sich zu orientieren und bietet eine einfache Möglichkeit sich einen Überblick verschaffen zu können.
2. Als Abfolge der Standorte in einer **Textdatei**. Diese kann verwendet werden, um die Grafik an Stellen, wo sie undeutlich/schwer erkennbar ist (z.B. aufgrund einer Konzentration von Punkten), zu unterstützen und zu ergänzen. Hier lässt sich auch schneller erkennen, welcher Punkt als nächstes angeflogen werden muss. Zusätzlich enthält diese Textdatei auch die Länge der Route, sodass der Pilot berechnen kann, wie viel Treibstoff er benötigt und gegebenen Falls einen Tankstopp einplanen kann.

Umsetzung

Zunächst werden die Koordinaten der Außenstellen aus der Datei eingelesen, wozu der Dateiname in die Eingabeaufforderung eingegeben wird. Anschließend wird die Datei vom Programm eingelesen und in eine Liste konvertiert. Diese konvertiert Daten (im Folgenden als Liste, die Punkte enthält, betrachtet) werden an eine Klasse (Objekt) gegeben, welche diese verwaltet und den maßgeblichen Algorithmus zur Berechnung der Flugroute beinhaltet.

Aufbau der Klasse/des Objekts

Wird das Programm nach einer erfolgreichen Eingabe einer gültigen Datei gestartet, so wird zunächst ein Objekt erstellt, an welches die Daten aus der Datei übermittelt werden. Dieses Objekt verfügt über die maßgeblichen Funktionen des Algorithmus zur Berechnung der Flugroute. Alle weiteren Funktionen, die auch allgemein gelten und allgemein genutzt werden können, sind von diesem Objekt ausgelagert.

Funktionen, über die das Objekt direkt verfügt (also in ihm direkt implementiert sind):

1. **Initialisierung**
   - [C11] implementiert als: *__init__()*
   - Initialisierung der Klasse/des Objekts
2. **Finden eines nächsten Punktes**
   - [C7] implementiert als: *next_point()*
   - Finden der möglichen Punkte, welche als nächstes angeflogen werden können (überschreiten nicht einen Abbiegewinkel von 90° und wurden noch nicht angeflogen)
   - Sortieren dieser Punkte nach Distanz, Auswahl des am nächsten liegenden, gültigen Punktes
3. **Finden einer Route**
   - [C8] implementiert als: *find_solution()*
   - Wiederholung der Suche nach einem nächsten Punkt, bis kein möglicher Punkt mehr gefunden wird
4. **Überprüfung der Route**
   - [C9] implementiert als: *check_results()*
   - Überprüfung der gefundenen Route, ob alle Punkte in ihr enthalten sind
   - ist die Route ungültig, so wird der gefundene Weg Schritt für Schritt „rückwärtsgegangen“ und nach einer anderen Route gesucht, bis eine gefunden wurde, eine alle Punkte beinhaltet
5. **Speichern der Route**
   - [C10] Implementiert als: *save_results()*
   - Speichern der gefundenen Lösung als Grafik und Textdatei

Sobald das Objekt erzeugt wurde, muss nur noch ein Startpunkt (im Programm der in der in der Eingabedatei erste Punkt, alle anderen werden als „mögliche Punkte“ gespeichert. Erklärung von „möglichen Punkten“ folgt) festgelegt und anschließend die Funktion *find_solution()* aufgerufen werden und es wird eine Flugroute berechnet und gespeichert.

Berechnung des Abbiegewinkels

Der Abbiegewinkel entspricht dem Schnittwinkel zweier Vektoren. Diese Vektoren verlaufen folgendermaßen

- Vektor a ($\vec{a}$) vom letzten Punkt zum aktuellen Punkt
- Vektor a ($\vec{b}$) vom aktuellen Punkt zum nächsten Punkt

Im Programmcode werden hierfür die abgekürzten, englischen Begriffe verwendet (lp = last point, cp = current point, np = next point).

Die **Vektoren** werden nach bekannten Rechenregeln, folgendermaßen berechnet:

$$\vec{v} = \overrightarrow{OP_2} - \overrightarrow{OP_1} = \begin{pmatrix} x_{P2} \\ y_{P2} \end{pmatrix} - \begin{pmatrix} x_{P1} \\ y_{P1} \end{pmatrix}$$

Des Weiteren benötigt man das **Skalarprodukt** der beiden Vektoren sowie den **Betrag** der Vektoren. Sie werden wie folgt berechnet:

$$|\vec{v}| = \sqrt{x^2 + y^2}$$

$$\overrightarrow{v_1} \circ \overrightarrow{v_2} = x_{v1} \cdot x_{v2} + y_{v1} \cdot y_{v2}$$

Anschließend kann man mit den errechneten Werten und der folgenden Formel den **Schnittwinkel** der beiden Vektoren berechnen, welcher dem Abbiegewinkel α entspricht:

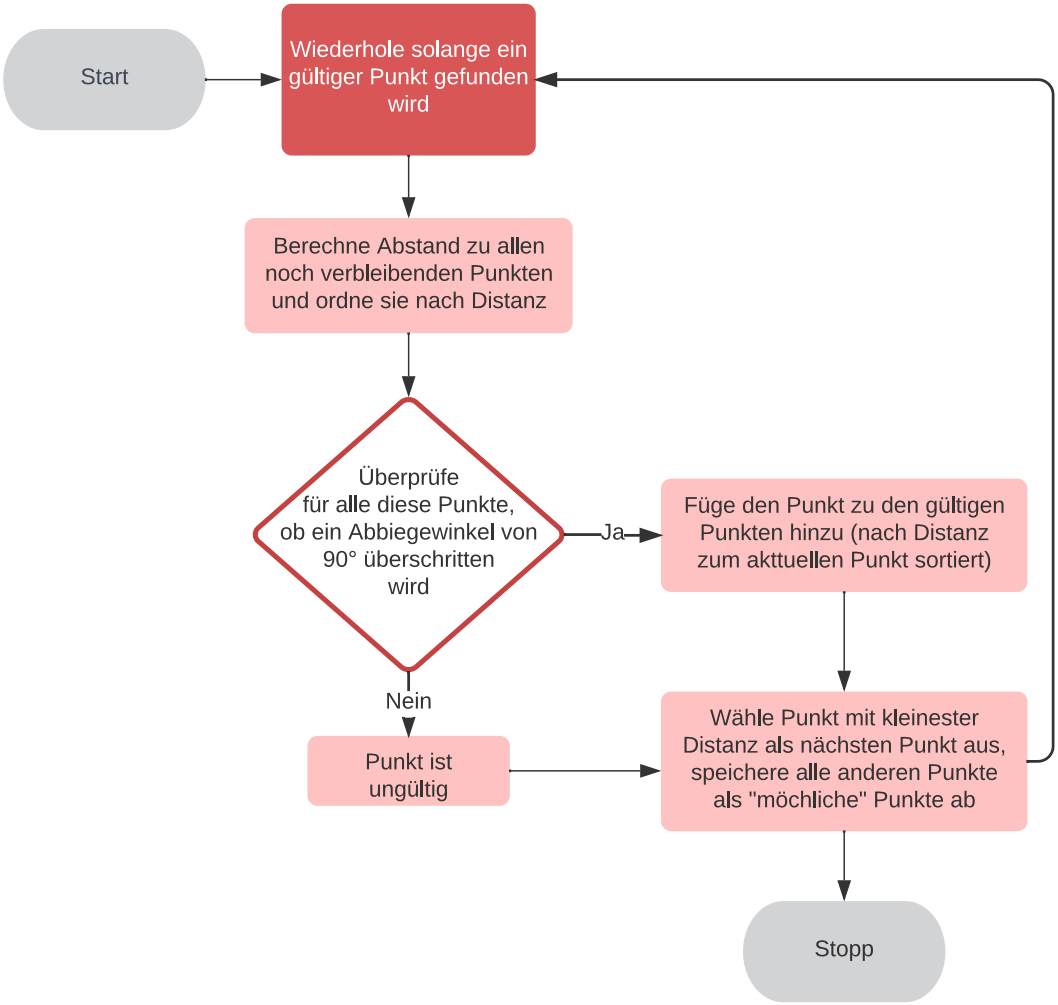
$$\alpha = \cos^{-1}\left(\frac{\vec{a} \circ \vec{b}}{|\vec{a}| \cdot |\vec{b}|}\right)$$

Diese Formeln werden im Code folgendermaßen implementiert:

- [C2] Berechnung der Vektoren
- [C6] Berechnung des Betrags eines Vektors
- [C5] Berechnung des Skalarprodukts zweier Vektoren
- [C3] Berechnung des Schnittwinkels zweier Vektoren

Berechnung der „ersten“ Route

Ist das Objekt erfolgreich initialisiert, so kann die Funktion *find_solution()* aufgerufen werden, welche die Berechnung einer Route startet. Diese Funktion versucht Mittels des erweiterten NNA das TSP zu lösen. Hierfür wird für jeden Punkt wie folgt vorgegangen:



[Abb. 4] Programmablaufplan der Berechnung der ersten Route

Endet dieser Vorgang, ist die Berechnung der ersten Route fertig. Das heißt, dass keine weiteren Punkte mehr gefunden werden, die als nächstes angeflogen werden können. Dies kann 2 Gründe haben:

1. Alle Punkte sind bereits in der Route enthalten, sodass kein weiterer Punkt mehr angeflogen werden muss.
2. Es gibt noch Punkte, die noch nicht angeflogen wurden. Jedoch können diese nicht angeflogen werden, da dabei ein Abbiegewinkel von 90° überschritten werden würde.

Um dies herauszufinden, wird die Funktion *check_solution()* aufgerufen, welche überprüft ob alle Punkte angeflogen wurden. Ist dies der Fall, gibt die Funktion den booleschen Wert *True* zurück, falls nicht *False*. Ist der zurückgegebene Wert *True*, so bedeutet das, dass eine gültige Route gefunden wurde, welche alle Punkte enthält und abgespeichert wird. Falls *False* zurückgegeben wird, werden weitere Berechnungen durchgeführt, welche im Folgenden erläutert werden.

Fortführende Berechnung bei Misserfolg der „ersten“ Route

Gibt *check_solution()* nach der Berechnung der ersten Route *False* zurück, so bedeutet das, dass diese Route nicht alle Punkte miteinbezieht. Um dieses Problem zu lösen, muss eine andere Route berechnet werden, die alle Punkte miteinbezieht. Als Basis hierfür dienen die zuvor errechneten, möglichen Punkte (im Folgenden als „mögliche Punkte“ bezeichnet), also die Punkte, die von einem Punkt aus ausgehend ebenfalls möglich (also gültig) gewesen wären, jedoch aufgrund von einer höheren Distanz zum derzeitig aktuellen Punkt nicht verwendet wurden. Diese Punkte wurden in folgendem Format in einer Liste abgespeichert:

Route = [[derzeitig aktueller Punkt], [[nächster Punkt], [mögliche Punkte]]]

Nun soll eine andere Route gefunden werden, die alle Punkte miteinbezieht, wofür die schon berechnete Route mit allen jeweils möglichen Punkten verwendet wird. Dabei wird die erste Route von hinten nach vorne durchlaufen. Hierbei wird der an dieser Stelle der Route gewählte Punkt der Reihe nach durch die möglichen Punkte ersetzt. Mit jedem auswechseln des Punktes wird analog zu dem Verfahren aus der Berechnung der ersten Route versucht, von dieser Stelle aus die Route weiterzuführen, also eine Route zu berechnen, die alle Punkte miteinbezieht. Ist dieser Prozess erfolgreich, wird die Route abgespeichert und das Programm beendet. Ist dies nicht der Fall, wird der Prozess des Auswechselns des Punktes durch mögliche Punkte und das anschließende Berechnen einer weiterführenden Route fortgesetzt. Wird bei diesem Prozess eine Route gefunden, die zwar nicht alle Punkte miteinbezieht, jedoch mehr Punkte, so wird dieser Prozess von dieser Route ausgehend ausgeführt.

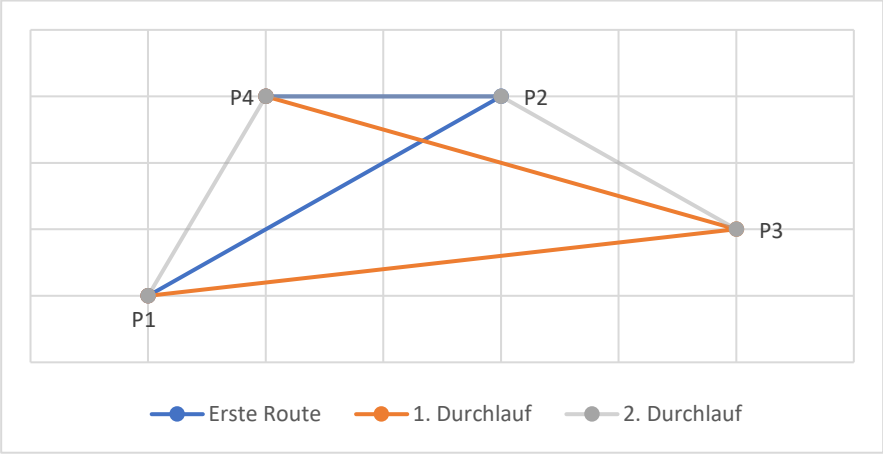
Veranschaulichung dieses Prozesses anhand der Punkte P1, P2, P3 und P4:

Hinweis: Diese Veranschaulichung zeigt in keiner Weise einen Rechenprozess, wie er wirklich stattfinden würde, da dem Programm widersprüchliche „Entscheidungen“ getroffen werden (z.B. wird ein Abbiegewinkel von 90° überschritten). Diese Veranschaulichung dient lediglich dem Verständnis der Berechnung einer gültigen Route, welche alle Punkte enthält, auf Basis der ersten Route.

Erste Route = [[P1, [P2, P3, P4]], [P2, [P3, P4]], [P4, []]] → Misserfolg

1. Durchlauf = [[P1, [P2, P3, P4]], [P3, [P4]], [P4, []]] → Misserfolg

2. Durchlauf = [[P1, [P2, P3, P4]], [P4, []], [P2, [P3]], [P3,[]]] → Erfolg



[Abb. 5] Veranschaulichung: Berechnung bei Misserfolg der „ersten“ Route

Laufzeitanalyse

Die hier durchgeführte Laufzeitanalyse ist die O-Notation. Das heißt, dass ausschließlich der worst-case für die Laufzeit des Programms betrachtet wird.
Bei der Berechnung ergeben sich folgende Laufzeiten für die einzelnen Funktionen. Hierbei rufen sich manche Funktionen gegenseitig auf, was ebenfalls schon in der Laufzeit der einzelnen Funktionen berücksichtigt ist.

Generelle Funktionen

- *read_file()*
  O(n)
- *get_time()*
  O(1)
- *get_path()*
  O(1)
- *get_vector()*
  O(1)
- *get_distance()*
  O(1)
- *get_vector_length()*
  O(1)
- *get_dot_product()*
  O(1)
- *get_cutting_angle()*
  O(1)
- *create_plot()*
  O(1)

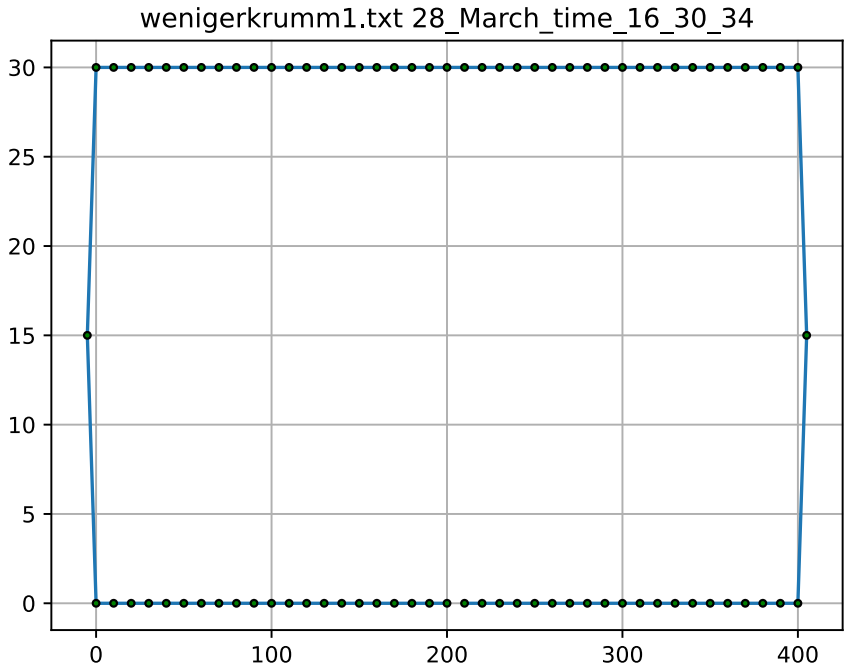
Klassen-/Objektfunktionen

- *__init__()*
  O(n)
- *next_point()*
  O(n)
- *find_solution()*
  $O(n^2)$
- *check_results()*
  O(1)
- *save_results()*
  O(n)

Insgesamt ergibt sich eine **Laufzeit von $O(n^2)$**. Hierbei ist jedoch anzumerken, dass die Wahrscheinlichkeit, dass es je zu diesem Fall kommt sehr gering ist. In den allermeisten Fällen findet das Programm schon nach wenigen Wiederholungen der in „Fortführende Berechnung nach Misserfolg der ersten Route“ [hier] beschriebenen Berechnung eine Route, die alle Punkte miteinbezieht.

Beispiele

Weniger krumm 1



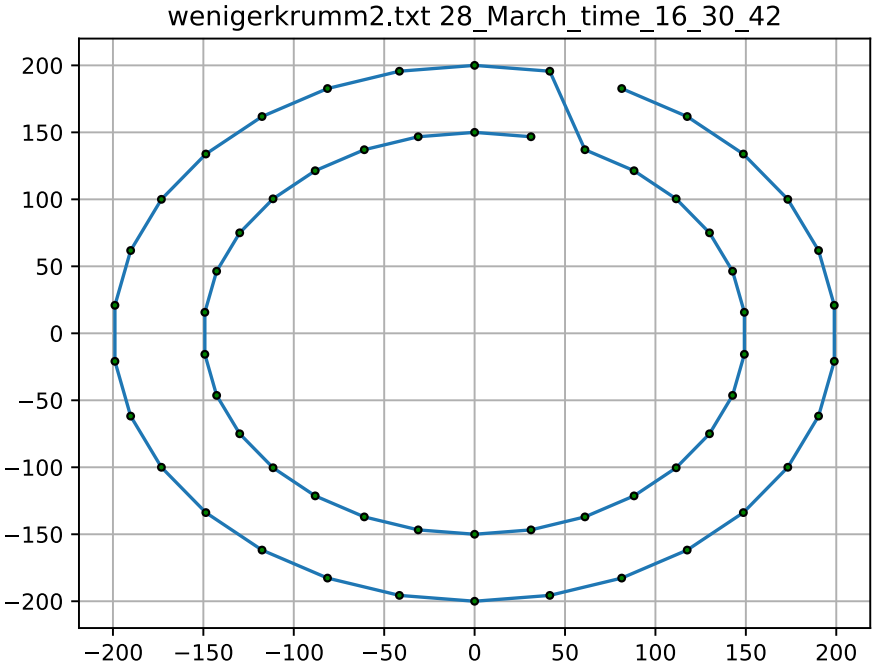
[B1] Grafik zu wenigerkrumm1.txt

Reihenfolge der Punkte

[[200.0, 0.0], [190.0, 0.0], [180.0, 0.0], [170.0, 0.0], [160.0, 0.0], [150.0, 0.0], [140.0, 0.0], [130.0, 0.0], [120.0, 0.0], [110.0, 0.0], [100.0, 0.0], [90.0, 0.0], [80.0, 0.0], [70.0, 0.0], [60.0, 0.0], [50.0, 0.0], [40.0, 0.0], [30.0, 0.0], [20.0, 0.0], [10.0, 0.0], [0.0, 0.0], [-5.0, 15.0], [0.0, 30.0], [10.0, 30.0], [20.0, 30.0], [30.0, 30.0], [40.0, 30.0], [50.0, 30.0], [60.0, 30.0], [70.0, 30.0], [80.0, 30.0], [90.0, 30.0], [100.0, 30.0], [110.0, 30.0], [120.0, 30.0], [130.0, 30.0], [140.0, 30.0], [150.0, 30.0], [160.0, 30.0], [170.0, 30.0], [180.0, 30.0], [190.0, 30.0], [200.0, 30.0], [210.0, 30.0], [220.0, 30.0], [230.0, 30.0], [240.0, 30.0], [250.0, 30.0], [260.0, 30.0], [270.0, 30.0], [280.0, 30.0], [290.0, 30.0], [300.0, 30.0], [310.0, 30.0], [320.0, 30.0], [330.0, 30.0], [340.0, 30.0], [350.0, 30.0], [360.0, 30.0], [370.0, 30.0], [380.0, 30.0], [390.0, 30.0], [400.0, 30.0], [405.0, 15.0], [400.0, 0.0], [390.0, 0.0], [380.0, 0.0], [370.0, 0.0], [360.0, 0.0], [350.0, 0.0], [340.0, 0.0], [330.0, 0.0], [320.0, 0.0], [310.0, 0.0], [300.0, 0.0], [290.0, 0.0], [280.0, 0.0], [270.0, 0.0], [260.0, 0.0], [250.0, 0.0], [240.0, 0.0], [230.0, 0.0], [220.0, 0.0], [210.0, 0.0]]

Length of route: 853.2455532033676km

Weniger krumm 2



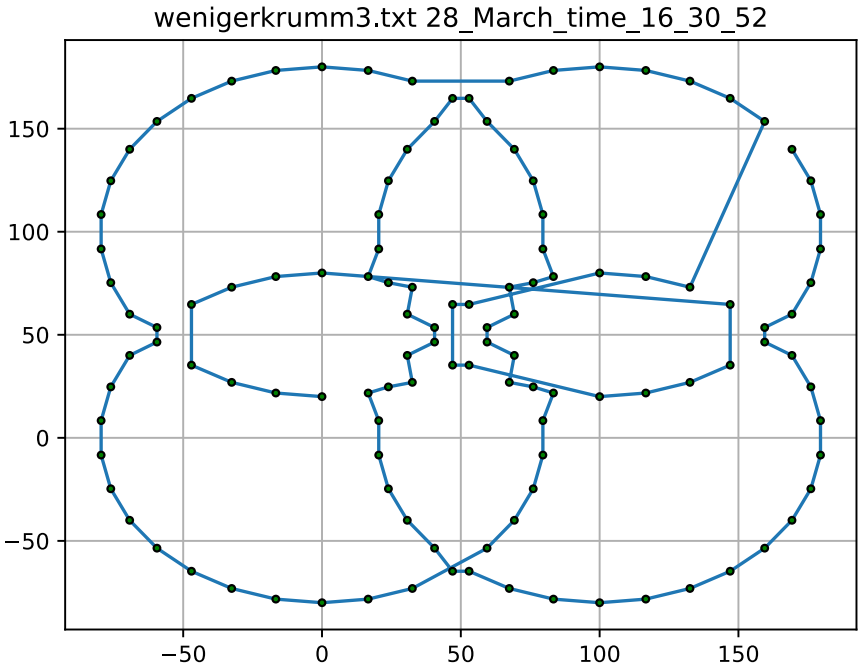
[B2] Grafik zu wenigerkrumm2.txt

Reihenfolge der Punkte

[[81.347329, 182.709092], [117.55705, 161.803399], [148.628965, 133.826121], [173.205081, 100.0], [190.211303, 61.803399], [198.904379, 20.905693], [198.904379, -20.905693], [190.211303, -61.803399], [173.205081, -100.0], [148.628965, -133.826121], [117.55705, -161.803399], [81.347329, -182.709092], [41.582338, -195.62952], [0.0, -200.0], [-41.582338, -195.62952], [-81.347329, -182.709092], [-117.55705, -161.803399], [-148.628965, -133.826121], [-173.205081, -100.0], [-190.211303, -61.803399], [-198.904379, -20.905693], [-198.904379, 20.905693], [-190.211303, 61.803399], [-173.205081, 100.0], [-148.628965, 133.826121], [-117.55705, 161.803399], [-81.347329, 182.709092], [-41.582338, 195.62952], [0.0, 200.0], [41.582338, 195.62952], [61.010496, 137.031819], [88.167788, 121.352549], [111.471724, 100.369591], [129.903811, 75.0], [142.658477, 46.352549], [149.178284, 15.679269], [149.178284, -15.679269], [142.658477, -46.352549], [129.903811, -75.0], [111.471724, -100.369591], [88.167788, -121.352549], [61.010496, -137.031819], [31.186754, -146.72214], [0.0, -150.0], [-31.186754, -146.72214], [-61.010496, -137.031819], [-88.167788, -121.352549], [-111.471724, -100.369591], [-129.903811, -75.0], [-142.658477, -46.352549], [-149.178284, -15.679269], [-149.178284, 15.679269], [-142.658477, 46.352549], [-129.903811, 75.0], [-111.471724, 100.369591], [-88.167788, 121.352549], [-61.010496, 137.031819], [-31.186754, 146.72214], [0.0, 150.0], [31.186754, 146.72214]]

Length of route: 2183.6622674320947km

Weniger krumm 3



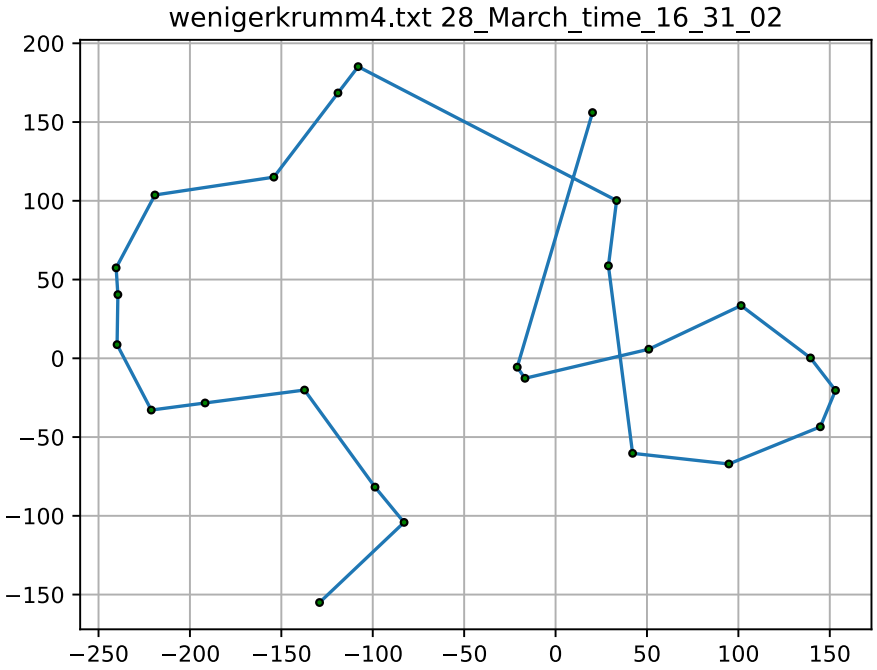
[B3] Grafik zu wenigerkrumm3.txt

Reihenfolge der Punkte

[[169.282032, 140.0], [176.084521, 124.72136], [179.561752, 108.362277], [179.561752, 91.637723], [176.084521, 75.27864], [169.282032, 60.0], [159.451586, 53.530449], [159.451586, 46.469551], [169.282032, 40.0], [176.084521, 24.72136], [179.561752, 8.362277], [179.561752, -8.362277], [176.084521, -24.72136], [169.282032, -40.0], [159.451586, -53.530449], [147.02282, -64.72136], [132.538931, -73.083637], [116.632935, -78.251808], [100.0, -80.0], [83.367065, -78.251808], [67.461069, -73.083637], [52.97718, -64.72136], [47.02282, -64.72136], [40.548414, -53.530449], [30.717968, -40.0], [23.915479, -24.72136], [20.438248, -8.362277], [20.438248, 8.362277], [16.632935, 21.748192], [23.915479, 24.72136], [32.538931, 26.916363], [30.717968, 40.0], [40.548414, 46.469551], [40.548414, 53.530449], [30.717968, 60.0], [32.538931, 73.083637], [23.915479, 75.27864], [16.632935, 78.251808], [20.438248, 91.637723], [20.438248, 108.362277], [23.915479, 124.72136], [30.717968, 140.0], [40.548414, 153.530449], [47.02282, 164.72136], [52.97718, 164.72136], [59.451586, 153.530449], [69.282032, 140.0], [76.084521, 124.72136], [79.561752, 108.362277], [79.561752, 91.637723], [83.367065, 78.251808], [76.084521, 75.27864], [67.461069, 73.083637], [69.282032, 60.0], [59.451586, 53.530449], [59.451586, 46.469551], [69.282032, 40.0], [67.461069, 26.916363], [76.084521, 24.72136], [83.367065, 21.748192], [79.561752, 8.362277], [79.561752, -8.362277], [76.084521, -24.72136], [69.282032, -40.0], [59.451586, -53.530449], [32.538931, -73.083637], [16.632935, -78.251808], [0.0, -80.0], [-16.632935, -78.251808], [-32.538931, -73.083637], [-47.02282, -64.72136], [-59.451586, -53.530449], [-69.282032, -40.0], [-76.084521, -24.72136], [-79.561752, -8.362277], [-79.561752, 8.362277], [-76.084521, 24.72136], [-69.282032, 40.0], [-59.451586, 46.469551], [-59.451586, 53.530449], [-69.282032, 60.0], [-76.084521, 75.27864], [-79.561752, 91.637723], [-79.561752, 108.362277], [-76.084521, 124.72136], [-69.282032, 140.0], [-59.451586, 153.530449], [-47.02282, 164.72136], [-32.538931, 173.083637], [-16.632935, 178.251808], [0.0, 180.0], [16.632935, 178.251808], [32.538931, 173.083637], [67.461069, 173.083637], [83.367065, 178.251808], [100.0, 180.0], [116.632935, 178.251808], [132.538931, 173.083637], [147.02282, 164.72136], [159.451586, 153.530449], [132.538931, 73.083637], [116.632935, 78.251808], [100.0, 80.0], [52.97718, 64.72136], [47.02282, 64.72136], [47.02282, 35.27864], [52.97718, 35.27864], [100.0, 20.0], [116.632935, 21.748192], [132.538931, 26.916363], [147.02282, 35.27864], [147.02282, 64.72136], [0.0, 80.0], [-16.632935, 78.251808], [-32.538931, 73.083637], [-47.02282, 64.72136], [-47.02282, 35.27864], [-32.538931, 26.916363], [-16.632935, 21.748192], [0.0, 20.0]]

Length of route: 2102.942874830064km

Weniger krumm 4



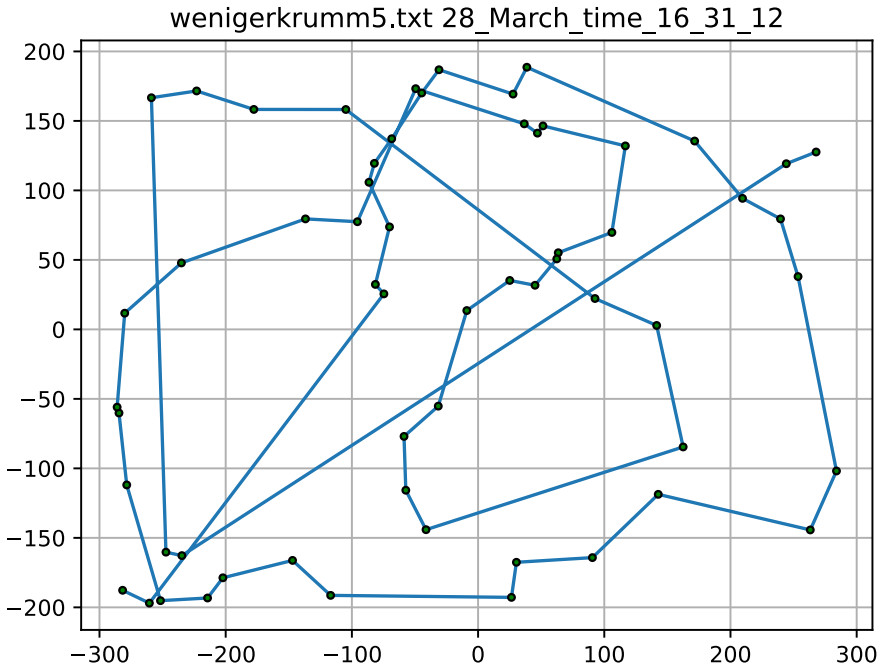
[B4] Grafik zu wenigerkrumm4.txt

Reihenfolge der Punkte

[[20.212169, 156.013261], [-20.971208, -5.637107], [-16.72313, -12.689542], [51.00814, 5.769601], [101.498782, 33.484198], [139.446709, 0.233238], [153.130159, -20.36091], [144.832862, -43.476284], [94.789917, -67.087689], [42.137753, -60.319863], [28.913721, 58.69988], [33.379688, 100.161238], [-107.988514, 185.173669], [-119.026308, 168.453598], [-154.088455, 115.022553], [-219.148505, 103.685337], [-240.369194, 57.426131], [-239.414022, 40.427118], [-239.848226, 8.671399], [-221.149792, -32.862538], [-191.716829, -28.360492], [-137.317503, -20.146939], [-98.760442, -81.770618], [-82.864121, -104.1736], [-129.104485, -155.04164]]

Length of route: 1386.3260614804515km

Weniger krumm 5



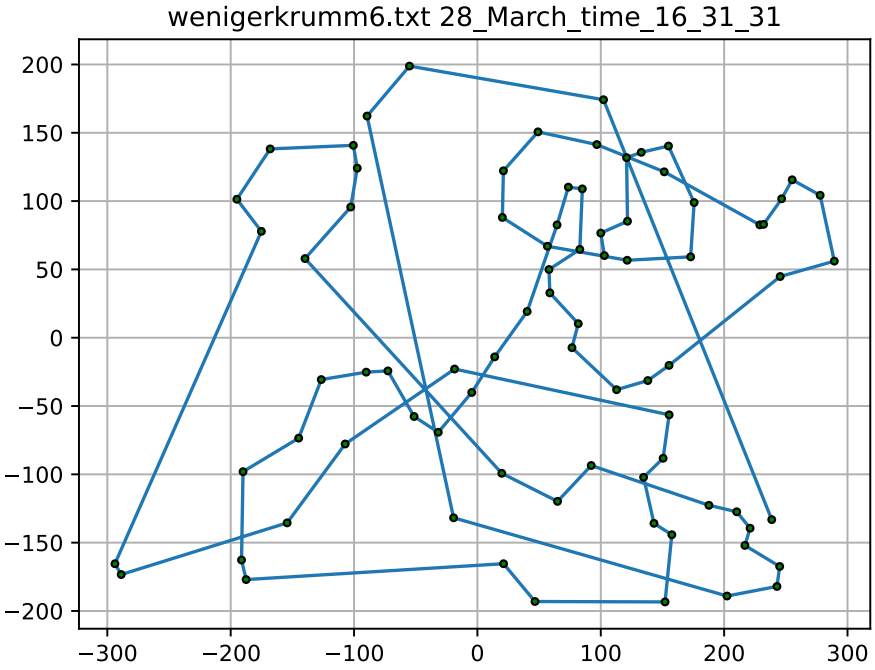
[B5] Grafik zu wenigerkrumm5.txt

Reihenfolge der Punkte

[[-281.67899, -187.717923], [-260.477802, -196.955535], [-74.639411, 25.542881], [-81.384378, 32.368323], [-70.183535, 73.738342], [-86.45758, 105.836348], [-82.17351, 119.465553], [-68.446198, 137.178953], [-44.669924, 170.088013], [-30.991436, 186.807059], [27.706327, 169.284192], [38.65473, 188.608557], [171.595574, 135.520994], [209.544977, 94.267052], [239.63955, 79.491132], [253.534863, 38.014987], [283.989938, -101.866465], [263.236651, -144.293091], [142.765554, -118.682439], [90.584569, -164.218416], [30.366828, -167.573232], [26.451074, -192.813352], [-116.831788, -191.380552], [-147.023475, -166.22013], [-202.218443, -178.735864], [-214.362324, -193.26519], [-251.656688, -195.189953], [-278.409792, -111.914073], [-284.547616, -60.154961], [-286.024059, -55.955204], [-280.008136, 11.657786], [-235.099412, 47.810306], [-136.787038, 79.501703], [-95.621797, 77.468533], [-49.447381, 173.210759], [36.599805, 147.88535], [47.040512, 141.206562], [51.417675, 146.417721], [116.702667, 132.021991], [106.03343, 69.754891], [63.541591, 55.140221], [62.366656, 50.713913], [45.123674, 31.740242], [25.098172, 35.205544], [-8.936916, 13.543851], [-31.548604, -55.223912], [-58.684205, -76.988884], [-57.266232, -115.737582], [-41.263039, -144.118212], [162.493244, -84.574019], [141.513053, 2.821137], [92.639946, 22.21603], [-104.781549, 158.212048], [-177.685937, 158.265884], [-223.039999, 171.558368], [-258.868593, 166.669198], [-247.341131, -160.277639], [-234.711279, -162.774591], [244.228552, 119.192512], [267.845908, 127.627482]]

Length of route: 4369.761697375777km

Weniger krumm 6



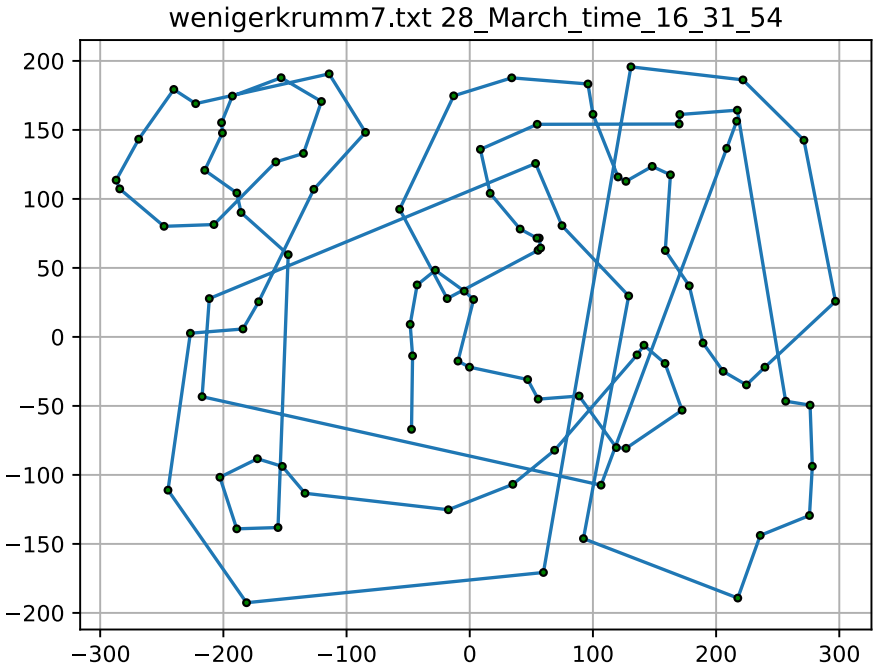
[B6] Grafik zu wenigerkrumm6.txt

Reihenfolge der Punkte

[[102.909291, 60.107868], [100.006932, 76.579303], [121.661135, 85.267672], [120.906436, 131.79881], [132.794476, 135.681392], [154.870684, 140.32766], [175.677917, 98.929343], [172.836936, 59.184233], [121.392544, 56.694081], [56.716498, 66.959624], [20.21829, 88.031173], [21.067444, 122.164599], [49.091876, 150.678826], [96.781707, 141.370805], [151.432196, 121.427337], [228.929427, 82.624982], [231.944823, 82.961057], [246.621634, 101.705861], [255.134924, 115.594915], [277.821597, 104.262606], [289.298882, 56.051342], [245.415055, 44.794067], [155.341949, -20.252779], [138.136997, -31.348808], [112.833346, -38.057607], [76.647124, -7.289705], [81.740403, 10.276251], [58.71662, 32.83593], [58.019653, 49.937906], [83.005905, 64.646774], [85.04383, 108.946389], [73.689751, 110.224271], [64.559003, 82.567627], [40.327635, 19.216022], [14.005617, -14.015334], [-4.590656, -40.067226], [-31.745416, -69.20796], [-51.343758, -57.654823], [-72.565291, -24.28182], [-90.16019, -25.200829], [-126.569816, -30.645224], [-144.887799, -73.49541], [-189.988471, -98.043874], [-191.216327, -162.689024], [-187.485329, -177.031237], [21.176627, -165.421555], [46.674278, -193.090008], [152.102728, -193.381252], [157.588994, -144.200765], [143.114152, -135.866707], [134.692592, -102.152826], [150.526118, -88.230057], [155.405344, -56.437901], [-18.507391, -22.90527], [-107.196865, -77.792599], [-154.225945, -135.522059], [-288.744132, -173.349893], [-293.833463, -165.440105], [-175.118239, 77.842636], [-194.986965, 101.363745], [-167.994506, 138.195365], [-100.569041, 140.808607], [-97.391662, 124.120512], [-102.699992, 95.632069], [-139.74158, 57.93668], [19.765322, -99.2364], [64.943433, -119.784474], [92.25582, -93.514104], [187.669263, -122.655771], [210.186432, -127.403563], [221.028639, -139.435079], [216.82592, -152.024123], [245.020791, -167.448848], [242.810288, -182.054289], [202.34698, -189.069699], [-19.310012, -131.810965], [-89.453831, 162.237392], [-55.091518, 198.826966], [102.223372, 174.201904], [238.583388, -133.143524]]

Length of route: 4814.159567276395km

Weniger krumm 7



[B7] Grafik zu wenigerkrumm7.txt

Reihenfolge der Punkte

[[-47.266557, -66.984045], [-46.403062, -13.755804], [-48.354421, 9.091412], [-42.704691, 37.679514], [-27.911955, 48.326745], [-4.434919, 33.164884], [3.152113, 27.10389], [-9.580869, -17.516639], [-0.200936, -21.927663], [47.011363, -30.88718], [55.550895, -45.089968], [88.818853, -42.834512], [118.989764, -80.203583], [126.904044, -80.733297], [172.389228, -53.13327], [158.552316, -19.254407], [141.433472, -6.023095], [135.781192, -13.05344], [68.910854, -82.123346], [34.959132, -106.842499], [-17.356579, -125.254131], [-133.730932, -113.306155], [-152.130365, -93.844349], [-172.378071, -88.298187], [-202.828627, -101.70005], [-189.135201, -139.078513], [-155.651746, -138.151811], [-147.363185, 59.608175], [-185.649161, 90.144456], [-189.062172, 104.285631], [-215.113949, 120.740679], [-200.771246, 147.741054], [-201.485143, 155.27483], [-192.681053, 174.522947], [-153.13014, 187.817274], [-120.386562, 170.589454], [-134.985023, 132.944989], [-157.423365, 126.800331], [-207.665172, 81.410371], [-248.169463, 80.132237], [-284.129027, 107.252583], [-287.058297, 113.599823], [-268.739142, 143.276483], [-240.249363, 179.334919], [-222.492322, 169.033315], [-114.146166, 190.615321], [-84.6269, 148.216494], [-126.568914, 106.964962], [-171.354954, 25.463068], [-184.0927, 5.737284], [-226.787625, 2.658862], [-244.959501, -111.046573], [-181.208895, -192.622935], [59.8272, -170.713714], [130.854855, 195.695082], [221.808162, 186.241012], [271.301094, 142.524086], [296.911892, 25.811569], [239.616628, -21.94416], [224.599361, -34.798485], [205.717887, -24.976511], [189.387028, -4.465225], [178.19836, 37.031984], [158.742184, 62.618834], [162.923138, 117.465744], [148.108328, 123.558283], [126.799911, 112.72728], [120.375033, 115.889661], [100.043893, 161.295125], [95.947213, 183.278211], [34.079032, 187.731112], [-13.030259, 174.698005], [-56.914543, 92.501249], [-18.316063, 27.75586], [55.434792, 62.72916], [57.555364, 64.417343], [56.389778, 71.618509], [54.551865, 71.567133], [40.897139, 78.152317], [16.573231, 104.020979], [8.64366, 135.90743], [54.766523, 154.053847], [169.990437, 154.260412], [170.514252, 161.16985], [217.218893, 164.294928], [256.475967, -46.591418], [276.276517, -49.448662], [278.105364, -93.771765], [275.793495, -129.415477], [235.827007, -143.838844], [217.599278, -189.258062], [92.29804, -146.169487], [129.024315, 29.701695], [74.8875, 80.586458], [53.436521, 125.683201], [-211.429137, 27.770425], [-217.28247, -43.316616], [106.599423, -107.433987], [208.592696, 136.61846], [216.691, 156.31437]]

Length of route: 6082.268462460331km

Wichtige Teile des Codes

```python
# read file and return data as list split by line and removed empty lines
def read_file(filename):
    f = open(filename, "r")
    raw_data = f.read().split("\n")
    # remove empty lines
    raw_data.remove("")

    # convert data to usable format
    data = []
    for coordinate in raw_data:
        xy = coordinate.split(" ")
        x = xy[0]
        y = xy[1]
        data.append([float(x), float(y)])
    return data
```

[C1] Einlesen und Konvertieren der Datei

```python
# calculate and return vector from point1 to point2
def get_vector(point1, point2):
    x1 = point1[0]
    y1 = point1[1]
    x2 = point2[0]
    y2 = point2[1]
    x = x2 - x1
    y = y2 - y1
    return [x, y]
```

[C2] Berechnung des Vektors von Punkt 1 zu Punkt 2

```python
# return cutting angle of two vectors
def get_cutting_angle(vector_a, vector_b):
    # check if vectors are equal, if they are, the cutting angle is 0°
    if vector_a != vector_b:
        # get dot product
        dot_product = get_dot_product(vector_a, vector_b)
        # get length of vectors
        length_a = get_vector_length(vector_a)
        length_b = get_vector_length(vector_b)

        # calculate cutting angle
        cutting_angle = math.acos(dot_product / (length_a * length_b))
        return math.degrees(cutting_angle)

    # if vectors are equal, return 0°
    else:
        return 0
```

[C3] Schnittwinkelberechnung der Vektoren a und b

```
# create and save plot with results
def create_plot(done, not_done, path, time):
    # define path which plot will be saved to
    folder = path + "/renders/results_" + time
    # create plot
    plt.grid()
    plt.title(file + " " + time)

    # convert done points to usable format (line) plot can use
    done_converted = []
    for point in done:
        done_converted.append(point[0])

    # convert done points to format (line) plot can use
    x_done = []
    y_done = []
    for point in done_converted:
        x_done.append(point[0])
        y_done.append(point[1])
    # add to plot
    plt.plot(x_done, y_done, marker="o", markersize=3,
             markeredgecolor="black", markerfacecolor="green")

    # add not done points to plot
    for point in not_done:
        plt.plot(point[0], point[1], marker="o", markersize=5,
                 markeredgecolor="black", markerfacecolor="red")

    # create folder if it doesn't exist
    if not os.path.exists(folder):
        os.makedirs(folder)
    # save plot and show it
    plt.savefig(folder + "/plot_" + time + ".png", dpi=600)
    plt.savefig(folder + "/plot_" + time + ".svg")
    plt.show()
```

[C4] Erstellen der Grafik, die den Verlauf der Flugroute zeigt, welche anschließend abgespeichert wird. Abspeichern der Reihenfolge, in der die Punkte abgeflogen werden.

```
# get and return dot product of two vectors
def get_dot_product(vector_a, vector_b):
    return vector_a[0] * vector_b[0] + vector_a[1] * vector_b[1]
```

[C5] Berechnung des Skalarprodukts zweier Vektoren

```
# return length of vector
def get_vector_length(vector):
    return math.sqrt(vector[0] ** 2 + vector[1] ** 2)
```

[C6] Berechnung des Betrags eines Vektors

```
def next_point(self):
    cp = self.cp.copy()
    lp = self.lp.copy()

    # calculate distances to all not done points
    distances = []
    for point in self.not_done.copy():
        distances.append([point, get_distance(cp, point)])
    # order from small distance to high distance
    distances.sort(key=lambda x: x[1])

    # check which points are valid
    valid_points = []
    for point in distances:
        # get vectors
        # if lp == [] --> first point is being checked (no lp available)
        if lp:
            vector_lp_cp = get_vector(lp, cp)
            vector_cp_point = get_vector(cp, point[0])
        else:
            vector_lp_cp = get_vector(cp, point[0])
            vector_cp_point = get_vector(cp, point[0])

        # get cutting angle
        cutting_angle = get_cutting_angle(vector_lp_cp, vector_cp_point)

        # check if cutting angle is smaller than or 90°
        if cutting_angle <= 90:
            valid_points.append(point)
        else:
            pass

    # check if there are valid points
    if not valid_points:
        return False

    else:
        # save current point and every possible point to done
        # add every valid point besides the first one to other_points
        other_points = []
        for point in valid_points[1:]:
            other_points.append(point[0])
        self.done.append([valid_points[0][0], other_points])
        self.not_done.remove(valid_points[0][0])

        # move cp to lp and nearest possible point to cp
        self.lp = cp.copy()
        self.cp = valid_points[0][0].copy()

        return True
```

[C7] Berechnung der Punkte, die am nächsten liegen und gültig sind. Anschließend wird der am nächsten liegende ausgewählt und die anderen abgespeichert.

```
def find_solution(self):
    # search for next point till no valid point is found
    while self.next_point():
        pass

    # check if all points are done (solution was found)
    if self.check_results():
        self.save_results()
    else:
        """
        loop backwards through calculated path and search for other
solutions by using other points (saved in
        self.done). For every of this actions calculate next points to
search for an other path. Repeat this
        process for this path and all following.
        """
        self.cor_path = self.done.copy() # save path to find other paths
        # reset values
        self.done = []
        self.not_done = self.all_points.copy()

        # loop backwards through calculated path
        for point in reversed(self.cor_path):
            # get index of point in self.cor_path
            index = self.cor_path.index(point)
            # copy cor_path to cur_path and remove all points after c_point
            self.cur_path = self.cor_path.copy()[:index]

            # loop through all other points
            for other_point in point[1]:
                leftover = point[1].copy()
                leftover.remove(other_point)
                self.cur_path1 = self.cur_path.copy()
                self.cur_path1.append([other_point, [leftover]])
                self.cp = self.cur_path1[-1][0].copy()
                self.lp = self.cur_path1[-2][0].copy()

                # add every point included in cur_path to done
                # reset values
                self.done = []
                self.not_done = self.all_points.copy()
                for p_in_cur_path in self.cur_path1.copy():
                    self.done.append(p_in_cur_path)
                    self.not_done.remove(p_in_cur_path[0])

                # search for next point till no valid point is found
                while self.next_point():
                    pass
                # check if all points are done (solution was found)
                if self.check_results():
                    self.save_results()
                    exit()
                else:
                    pass
```

[C8] Berechnung eines ersten Weges. Beinhaltet dieser schon die Lösung, so ist das Programm fertig. Falls nicht fährt das Programm mit folgendem Ablauf fort:
Die bisher gefundene Lösung wird von hinten her durchgangen. Neben dem nächstliegenden gültigen Punkten sind auch alle anderen Punkte immer abgespeichert worden. Nun wird der ausgewählte Punkt der Reihe nach mit den anderen gültigen Punkten ausgetauscht und geprüft,

ob nun eine Lösung gefunden werden kann. Dieser Prozess wird so lange wiederholt, bis eine Lösung gefunden wurde.

```
def check_results(self):
    # check if all point are done
    if not self.not_done:
        print("\n\nALL POINTS ARE DONE\n")
        print("Saving results...")
        return True
    else:
        return False
```

[C9] Überprüft, ob eine Lösung gefunden wurde

```
# save results
def save_results(self):
    create_plot(self.done.copy(), self.not_done.copy(), self.path,
self.time)

    # save done to file
    location = self.path + "/renders/results_" + self.time + "/results_" +
self.time + ".txt"
    f = open(location, "a")
    data = []
    for point in self.done.copy():
        data.append(point[0])

    length_route = get_length_route(data)
    f.write(str(data) + "\nLength of route: " + str(length_route) + "km")
    print("Results saved to", location)
```

[C10] Speicher die gefundene Lösung ab

```
# initialize class
def __init__(self, filename):
    # general
    self.path = get_path()  # path of executed file
    self.time = get_time()  # current time

    # coordinates
    self.all_points = []    # all coordinates
    self.done = []          # coordinates that are already done
    self.not_done = []      # coordinates that are not done yet
    self.stp = []           # start point
    self.lp = []            # last point
    self.cp = []            # current point
    self.cor_path = []      # base path; source to search for other paths
    self.cur_path = []      # path to store current path (used to find
other paths)
    self.cur_path1 = []     # path to store current path (used to find
other paths) points added

    # set to values
    self.all_points = read_file(filename)   # read coordinates from file
and save them to all_points
    self.not_done = self.all_points.copy()  # copy all_points to not_done
```

[C11] Initialisierung der Klasse

Quellen

Textquellen

- **[Q1]** Wikipedia: Traveling Salesman Problem
  https://en.wikipedia.org/wiki/Travelling_salesman_problem
  28.03.2023 | 13:30 Uhr
- **[Q2]** Wikipedia: Nearest Neighbour Algorithm
  https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm
  28.03.2023 | 13:40 Uhr

Bildquellen

- **[Abb. 1]** Darstellung: Wo befindet sich der Abbiegewinkel
  https://bwinf.de/fileadmin/bundeswettbewerb/41/aufgaben412.pdf
  05.04.2023 | 19:00 Uhr
- **[Abb. 2]** Darstellung: Methode 1 zur Berechnung des Abbiegewinkels
  Imanuel Fehse
  28.03.2023 | 14:30 Uhr
- **[Abb. 3]** Darstellung: Methode 2 zur Berechnung des Abbiegewinkels
  Imanuel Fehse
  28.03.2023 | 14:30 Uhr
- **[Abb. 4]** Programmablaufplan: Berechnung der „ersten“ Route
  Imanuel Fehse
  07.04.2023 | 14:50 Uhr
- **[Abb. 5]** Veranschaulichung: Berechnungen bei Misserfolg der „ersten“ Route
  Imanuel Fehse
  07.04.2023 | 16:30 Uhr
- **[B1]** Auswertung: wenigerkrumm1.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B2]** Auswertung: wenigerkrumm2.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B3]** Auswertung: wenigerkrumm3.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B4]** Auswertung: wenigerkrumm4.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B5]** Auswertung: wenigerkrumm5.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B6]** Auswertung: wenigerkrumm6.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr
- **[B7]** Auswertung: wenigerkrumm7.txt
  Imanuel Fehse
  28.03.2023 | 18:30 Uhr