Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: 66692

Bearbeiter dieser Aufgabe:
Sven Zimmermann

March 2023

Inhalt

1. Lösungsidee

Bei der Aufgabe „weniger krumme Touren“ sollte eine Route durch eine Menge an Punkten bestimmt werden. Für diese Route sollte folgendes gelten:

1. Sie muss alle Punkte enthalten.
2. Die Verbindungen zwischen zwei Punkten muss geradlinig sein.
3. Es sollen innerhalb der Route , abbiege-Winkel‘ von unter 90° (spitze Winkel) vermieden werden.
4. Die Strecke der Tour soll dabei möglichst kurz sein.

Festzustellen ist, dass es nicht immer möglich ist eine Route durch n Punkte zu finden ohne spitzen Winkel. Nimmt man beispielsweise die drei Eckpunkte eines gleichseitigen Dreiecks, so enthält die Route (2 Strecken) zwangsweise einen spitzen Winkel (60°).

1.1 Ansatz zum Generieren der Route

Der einfachste Ansatz zum Generieren der optimalen Lösung für das Problem ist das systematische Durchprobieren aller möglichen Routen. Da es jedoch bei n Punkten n! mögliche Routen gibt, wächst die Menge der zu Überprüfenden Routen mit der Menge an Punkten sehr schnell. Außerdem ist bei einem solchen Algorithmus erst nach dem Überprüfen aller möglichen Routen eine Aussage über die mögliche Lösung möglich (Zwischenergebnisse wären sinnlos). Daher ist ein solcher Algorithmus zur Lösung von allgemeinen Problemen dieser Art eher unpraktisch. Eine alternative stellt ein biologisch inspirierter Algorithmus dar, welcher die Verhaltensweisen von Waldameisen zur Kommunikation von Wegen zu Nahrungsquellen imitiert. Jene machen ihre Bewegungen von den Pheromonen anderer Ameisen und nahen Nahrungsquellen abhängig und gewichten selber Strecken mit Pheromonen, welche zu Nahrungsquellen führen. In der Natur entwickeln sich dann, durch die unterschiedliche Frequenz der, den Weg nutzenden, Ameisen (je kürzer die Strecke, desto schneller können Ameisen vom Nest zur Quelle laufen) und durch das langsame Verblassen der Pheromone, erhöhte Pheromon-Konzentrationen auf den kürzesten Strecken zur Nahrungsquelle:

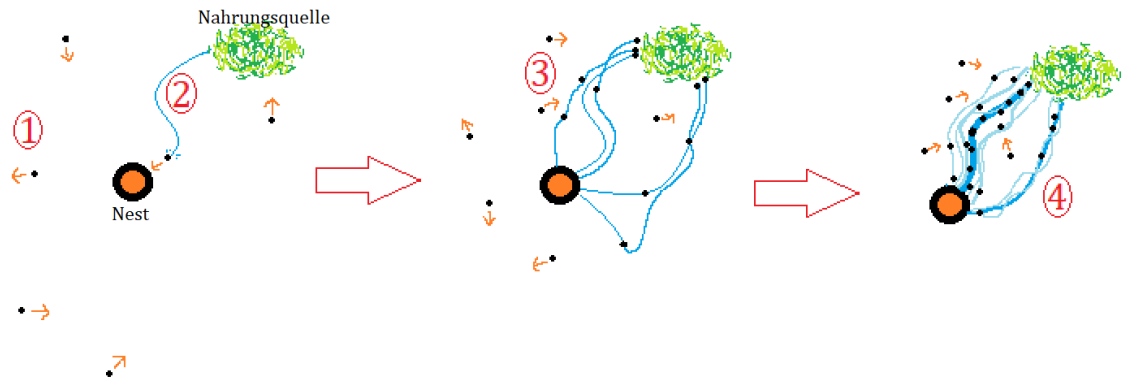


Abbildung 1.1: Schematische Darstellung des Verhaltens von Ameisen

1. Einzelnes Umherstreifen von Ameisen zur Suche nach Nahrung.
2. Falls eine Ameise eine Nahrungsquelle findet, stößt sie Pheromone auf dem Weg von der Nahrungsquelle zurück zum Nest aus, welche mit der Zeit jedoch verblassen.
3. Diesen Pheromonen folgen weitere Ameisen, sodass mehr und mehr Ameisen die Nahrungsquelle finden, welche selber die Pheromone auf der Strecke erneuern.

4. Durch das häufige Nutzen der Pheromon-Wege, entstehen große Konzentrationen, welche zur kürzeren Strecke tendieren, da jene durch eine höhere Ameisen-Frequenz eine höhere Pheromon-Konzentration ermöglichen.

Dieses Verhalten kann nun in einer abgewandelten Art von einem Programm imitiert werden. Ein solcher Algorithmus würde zunächst zufällige Routen generieren und dann die genutzten Strecken entsprechend der Qualität (z.B. Länge) der Routen bewerten und mit ‚Pheromonen' gewichten. Diese dienen dann bei der nächsten Generation als Wegweiser beim Generieren der Routen. Normalerweise werden diese sogenannten **ant colony optimization algorithms (ACO)** für das finden des kürzesten Weges genutzt, jedoch können sie – in leicht angepasster Form – auch hier verwendet werden.

Der Ablauf zum Generieren einer möglichst optimalen Route durch einen solchen Algorithmus wäre folgender:

1. Generiere eine beliebige Menge an Routen (=Umherstreifende einzelne Ameisen). Wähle die Teilstrecken der Route unter Beachtung der entsprechenden Gewichtungen (=Pheromone) und Länge der Strecke, sowie dem dadurch entstehenden Winkel.
2. Bewerte jede der generierten Routen und justiere die Gewichte indem die Gewichtungen der genutzten Strecken, je nach Bewertung der Route, anpasst werden.
3. Multipliziere alle Gewichte mit einer Konstante (0<x<1). (=Verfall der Pheromone)
4. Wiederhole Schritte 1-3 (jede Wiederholung = Generation), bis eine möglichst optimale Route gefunden wurde.

1.2 Generieren einer Route

Die Grundbausteine des Algorithmus sind die Routen selber, welche – optimalerweise – bei jeder Generation tendenziell besser werden. Beim Generieren der Routen müssen daher die Gewichtungen, welche sich aus den vorangegangenen Generationen ergeben haben, beachtet werden. Um eine, für die Verbesserungstendenz nötige, Variation zu erreichen, müssen die ‚Ameisen' beim Generieren einer neuen Route jedoch trotzdem Raum für Abweichungen haben, wobei sie, sich möglichst positiv auswirkende, Änderungen vornehmen sollen. Da jedoch eine Änderung vorweg nicht als positiv oder negativ klassifiziert werden kann, sollte die ‚einzelne Ameise' bei Änderungen möglichst eine kürzere Strecke oder einen stumpferen Winkel als neue Strecke wählen, da dadurch am ehesten positive Änderungen entstehen.

Das Zusammensetzen einer Route besteht insgesamt daraus, die verschiedenen Punkte in einer bestimmten Reihenfolge miteinander zu verbinden. Da der Startpunkt unerheblich ist, wird er zufällig aus allen Punkten gewählt. Davon ausgehend werden die nächsten Punkte der Route danach gewählt, wie weit sie entfernt sind, was für ein Winkel durch die Verbindung in der Route entstehen würde und wie stark die Strecke bisher gewichtet wurde:

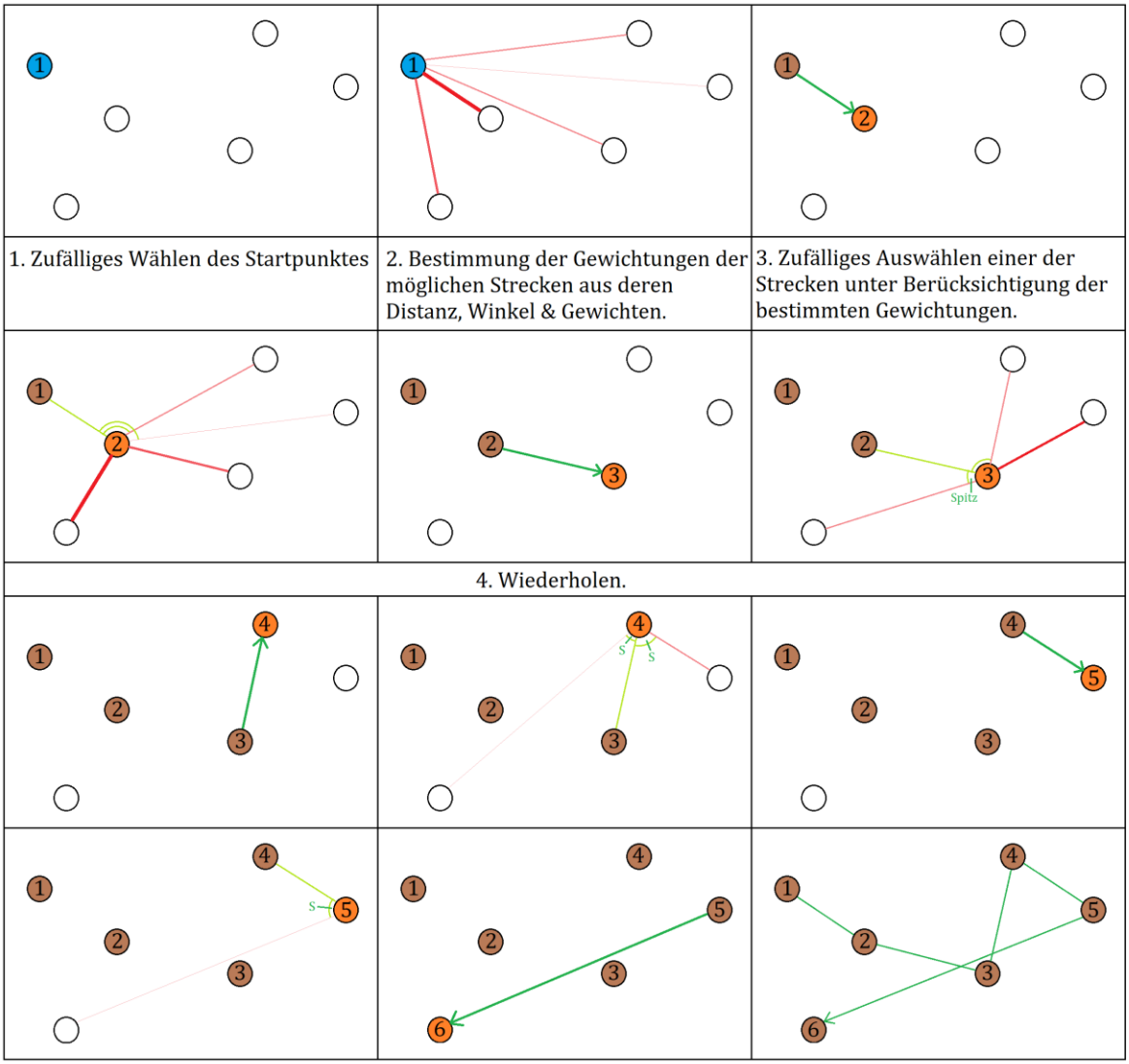


Abbildung 1.2: Exemplarischer Ablauf der Generation einer Route.

1.3 Bewerten der Routen

Wie sich zeigt kann das Wählen von verschiedenen Strecken am Ende dazu führen, dass nur noch Punkte übrig sind, deren Erreichen eine spitzwinklige Kurve benötigen (vgl. Abb. 1.2: 4->5). Daher ist es notwendig mehrere Routen zu generieren und die Gewichte entsprechend deren relativer Performance zu aktualisieren. Die Bewertungskriterien wären beim vorliegenden Problem:

1. Menge an spitzen Winkeln in der Route.
2. Länge der Route.

Wobei die Menge an spitzen Winkeln deutlich stärker in die Bewertung einfließen sollte, da die Aufgabe das Finden einer Route ohne spitzen Winkel ist, während die Länge nur eine sekundäre Rolle spielt.

1.4 Justieren der Gewichte

1.4.1 Gewichte aus Routen

Um eine Verbesserung in den generierten Routen über die Generationen zu erreichen, müssen Informationen über die Strecken zwischen jenen ausgetauscht werden. Dies geschieht in Form von Gewichten in Anlehnung an die Pheromone der Ameisen. Hierbei werden einige Strecken, die durch

häufiges Auftreten in den optimalen Routen der Generationen auffallen, stärker gewichtet und somit von nachfolgenden Generationen präferiert verwendet:

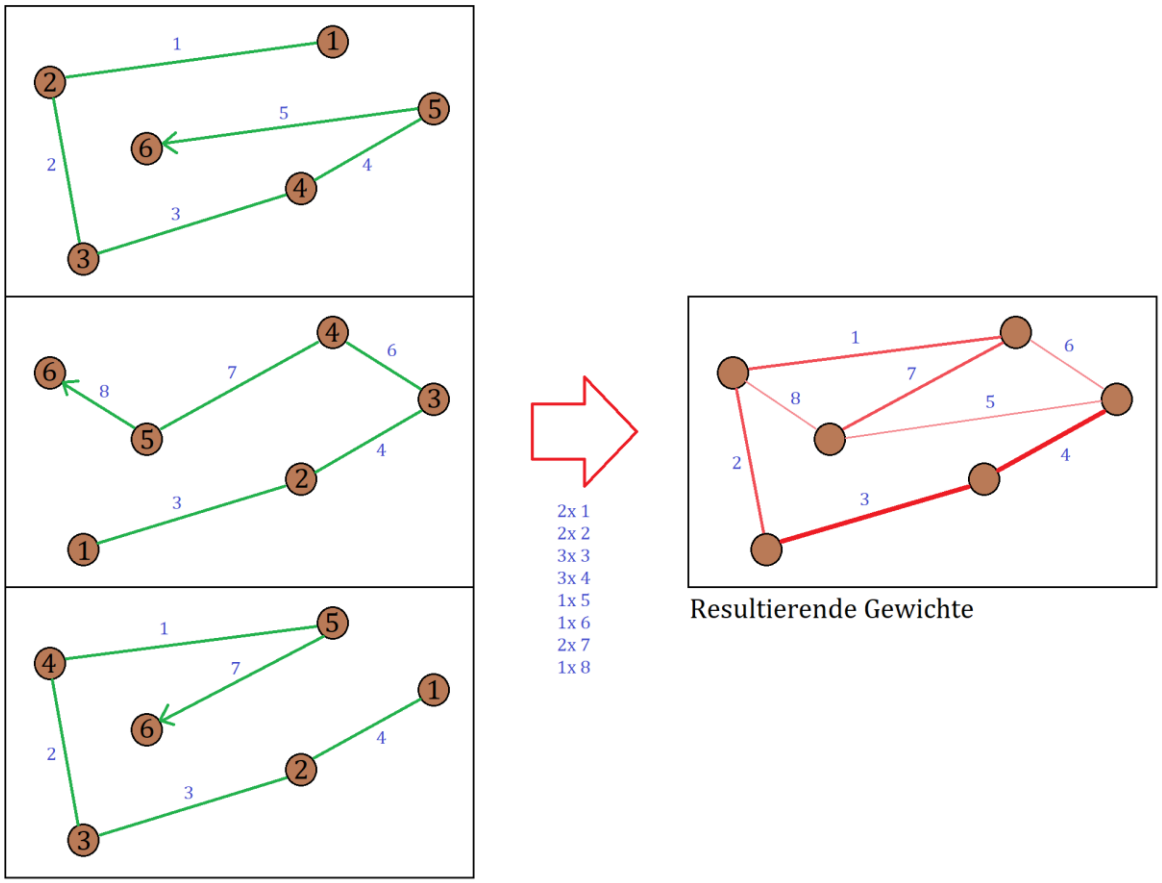


Abbildung 1.3: Berechnung der Gewichte aus den Routen einer Generation.

1.4.2 Verblassen der Gewichte

Um den Einfluss der früheren Generationen gegenüber dem Einfluss der späteren Generationen zu senken, werden die Gewichte nach jeder Generation abgeschwächt, sodass die Gewichte der früheren Generationen über die Zeit immer stärker ‚verblassen' (=Verblassen der Pheromone in der Natur).

2 Umsetzung

Um die Menge der möglichen konstanten Einstellungen für den Ablauf der Routen-Generierung (zum Beispiel: verblass-Geschwindigkeit der Gewichte, Gewichtungen der Faktoren beim Generieren der Routen etc.) kompakt umzusetzen, ist die Implementierung des Algorithmus als Klasse sinnvoll. Dabei Umfassen die zu initiierenden Einstellungen:

1. Gewichtungen der Einflussfaktoren Winkel, Distanz und vorherige Gewichte bei der Wahl des nächsten Punktes beim Generieren der Routen.
2. Gewichtung der vergleichsmäßigen Performance einer Route in der Generation für deren Bewertung und Einfluss in die Justierung der Gewichte.
3. Geschwindigkeit des Verblassens der Gewichte (durch einen Faktor 0<x<1: je kleiner desto schneller verblassen die Gewichte).

Die Gewichte werden umgesetzt als eine Liste an Wahrscheinlichkeitsverteilungen für jeden Punkt eine gewisse nächste Strecke zu wählen.

2.1 Generieren einer Route

Das Generieren einer Route besteht aus dem Wiederholten Wählen von Strecken von Punkt zu Punkt (vgl. Abb. 1.2). Jede Wahl soll dabei von den drei Faktoren Winkel, Distanz und Beliebtheit (=Gewichte) der vorherigen Generationen abhängig gemacht werden. Um Varianz in die Routen zu bringen ist die Endgültige Entscheidung der Streckte trotzdem zufällig, wobei die Tendenz der Faktoren zu beachten ist.

Hierfür existiert in der Python-Programmbibliothek NumPy die Funktion random.choice, welche zufällig ein Element aus einer Liste ziehen kann, wobei jedem Element der Liste eine Wahrscheinlichkeit zugeordnet werden kann. Die Elemente sind in dem Fall die möglichen Strecken und die Wahrscheinlichkeiten die zugehörigen normalisierten (Summe aller Wahrscheinlichkeiten = 1) Gewichtungen.

Eine mögliche Implementierung in Pseudocode sähe, wie folgt, aus:

```
Algorithmus: Generiere_Route:
Eingabe: Gewichte, Punkte
route <- leere Liste
mögliche_Punkte <- Punkte
Startpunkt <- wähle einen zufälligen Punkt aus Punkte
entferne Startpunkt aus mögliche_Punkte
füge route den Punkt Startpunkt hinzu
aktueller_Punkt <-Startpunkt
solange mögliche_Punkte nicht leer, tue:
        wahrscheinlichkeiten <- Gewichte an der Stelle von aktueller_Punkt
        setzte alle wahrscheinlichkeiten für die Punkte in route auf 0
        bestimme die entstehenden Winkel und Distanzen für jeden potentiellen Punkt, verändere wahrscheinlichkeiten entsprechend
        nächster_punkt <- wähle einen Punkt aus mögliche_Punkte entsprechend wahrscheinlichkeiten
        entferne nächster_punkt aus mögliche_Punkte
        füge route den Punkt nächster_punkt Startpunkt hinzu
        aktueller_Punkt <-nächster_punkt
gebe route zurück
```

2.2 Integration der Faktoren in die Wahrscheinlichkeit

Essentiell für die positive Varianz der Routen einer Generation (die abweichenden Routen, die eine Verbesserung zur bisher besten Route darstellen), ist die verhältnismäßige Integration der Faktoren Winkel, Distanz und Gewichte der vorherigen Generationen in die Generierung der Routen. Ist zum Beispiel der Einfluss der Gewichte zu hoch, so entsteht kaum Varianz. Ist zum anderen der Einfluss von Distanz zu hoch, so werden keine Routen generiert, welche längere Strecken nutzen um spitze Winkel zu vermeiden. Die Optimale Gewichtung der einzelnen Faktoren variiert von Fall zu Fall und sollte deswegen Variabel sein (Einstellungen).

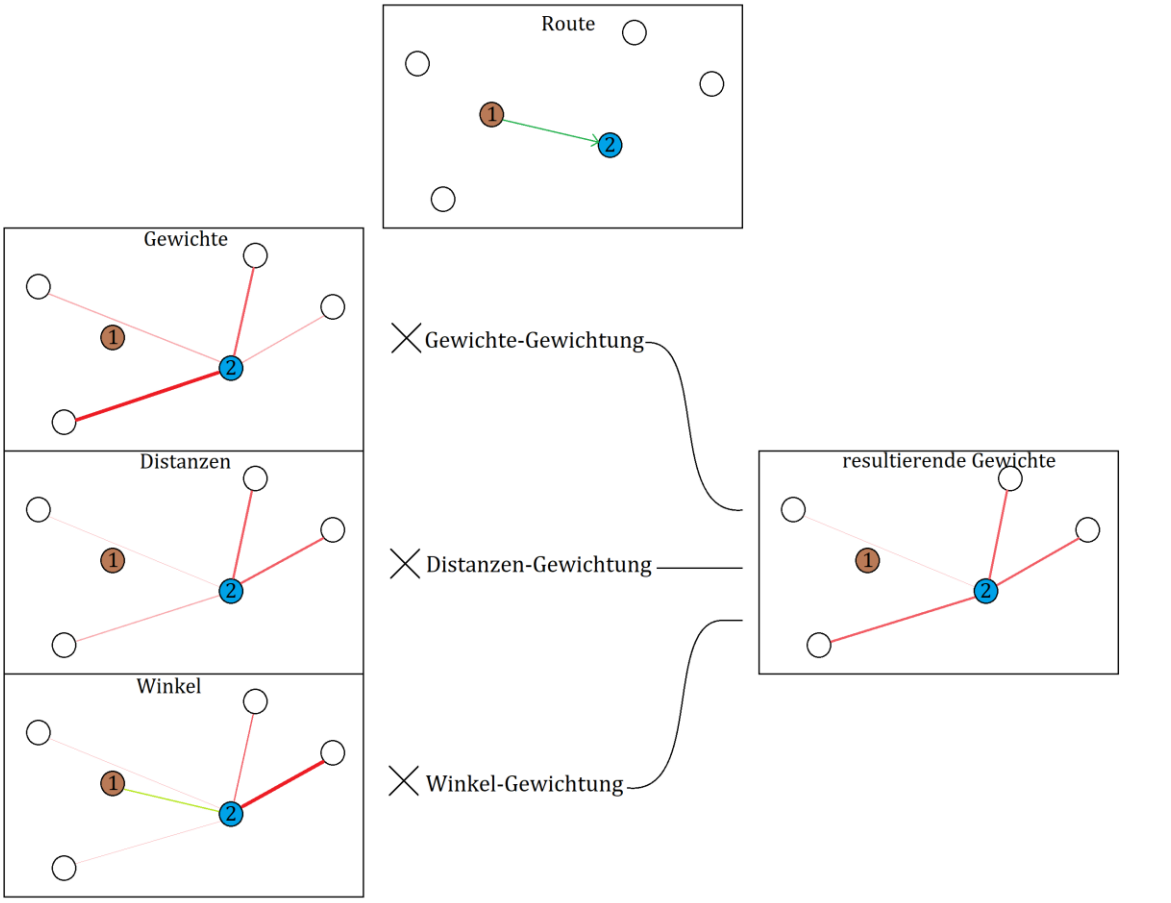


Abbildung 2.1: Beispielhafte Verrechnung von Faktoren

Um die Gewichtung und die Faktoren zu kombinieren und in die Wahrscheinlichkeit zu integrieren, erweist es sich am praktischsten die Faktoren mit ihren Gewichtungen zu potenzieren, da, wenn die Faktoren zwischen 0 und 1 normalisiert sind, eine Potenzierung zu einer entsprechend starken Polarisierung führt. Die Verrechnung der Faktoren miteinander erfolgt dann durch Multiplikation. Beispielsweise:

$$p_{Strecke} = (w_{Strecke})^{Gewichtung_{Gewiche}} * \left(\frac{1}{d_{Strecke}}\right)^{Gewichtung_{Distanz}} * \left(\frac{\alpha_{Strecke}}{360°}\right)^{Gewichtung_{Winkel}}$$

2.3 Bewerten einer Route

Um die Routen vergleichbar zu machen, ist es notwendig sie anhand konstanter Kriterien und Methoden zu bewerten. Die Kriterien sollten hierbei sein:

1. Routen mit weniger spitzen Winkeln sollten, unabhängig von ihrer Gesamtlänge, eine bessere Bewertung erhalten, als welche mit mehr.
2. Routen mit gleicher Menge an spitzen Winkeln, aber kürzerer Strecke, sollten besser Bewertet werden, als solche mit einer längeren Gesamtstrecke.
3. Die Bewertungen verschiedener Routen sollten einfach vergleichbar und möglichst einfach zu berechnen sein um keinen unnötigen Rechenaufwand zu erzeugen.

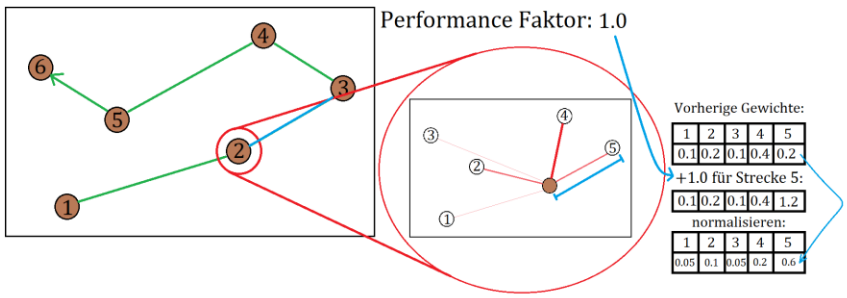
Um diese Kriterien umzusetzen ist das Produkt der Menge der spitzen Winkel (plus 1, da es auch 0 spitze Winkel geben kann) mit der Gesamtlänge der Route am praktischsten. Da eine Route umso besser ist desto kleiner ihre Gesamtlänge und ihre Menge an spitzen Winkeln, gilt:

Je besser eine Route ist desto niedriger ist ihre Bewertung.

2.4 Justieren der Gewichte

Die Justierung der Gewichte nach den Routen einer Generation soll abhängig von deren relativer Performance sein. So sollen Routen die gut waren (zum Beispiel: wenig spitze Winkel/kürzere Gesamtstrecke) stärker in die Gewichte für die nächsten Generationen einfließen, als Routen die schlechter waren. Hierfür werden alle Routen einer Generation zunächst nach ihren Bewertungen sortiert, die beste Bewertung ermittelt und jeder Route dadurch ein Score (zwischen 0 (schlecht) und (1) am besten) zugewiesen, indem die beste Bewertung durch ihre Bewertung (≥beste Bewertung) dividiert wird. Diese wird dann mit einem variablen Faktor potenziert wird um zusätzlich zwischen den guten und schlechten Routen zu polarisieren. Der resultierende Faktor wird auf alle Strecken der zugehörigen Route in den Gewichten aufaddiert und die Gewichte wieder normalisiert, sodass die Strecken der Route in den Gewichten leicht wahrscheinlicher sind als zuvor, während alle anderen Strecken leicht unwahrscheinlicher sind.

Ablauf für eine Route



Nachteil, wenn dieses Verfahren für jede Route einzeln angewendet würde:

vorherige Gewichte: [0.1 | 0.2 | 0.1 | 0.4 | 0.2]
1. Route (Performance: 1.0, Nutzt Strecke 5)
neue Gewichte : [0.1 | 0.2 | 0.1 | 0.4 | 1.2]
normalisiert : [0.05|0.1 |0.05|0.2 | 0.6] => 5 am wahrscheinlichsten
2. Route (Performance: 0.8, Nutzt Strecke 3)
neue Gewichte : [0.05|0.1 |0.85|0.2 | 0.6]
normalisiert : [0.03|0.05|0.47|0.12|0.33] => 3 am wahrscheinlichsten
3. Route (Performance: 0.6, Nutzt Strecke 1)
neue Gewichte : [0.63|0.05|0.47|0.12|0.33]
normalisiert : [0.4|0.03|0.29|0.07|0.21] => 1 am wahrscheinlichsten

=> Repräsentiert nicht alle Routen, sondern nur die als letztes hinzugefügte.

Ablauf mehrerer Routen

1. Alle Faktoren addieren:
vorherige Gewichte: [0.1 | 0.2 | 0.1 | 0.4 | 0.2]
nach der 1. Route : [0.1 | 0.2 | 0.1 | 0.4 | 1.2] (+1.0 bei 5)
nach der 2. Route : [0.1 | 0.2 | 0.9 | 0.4 | 1.2] (+0.8 bei 3)
nach der 3. Route : [0.7 | 0.2 | 0.9 | 0.4 | 1.2] (+0.6 bei 1)

2. Normalisieren:
neue Gewichte : [0.21|0.06|0.26|0.12|0.35] => bessere Repräsentation der Routen

Abbildung 2.2: Exemplarische Justierung der Gewichte.

2.5 Übernehmen der besten Routen

Da für die Justierung der Gewichte nur die Performance der Routen innerhalb der Generation verglichen wird, ist es möglich, dass durch eine schlechte Generation die Gewichte verschlechtert werden, was zu einer Abwärtsspirale in der Entwicklung der Generationen führt. Daher ist es sinnvoll immer die besten Routen einer Generation als Vergleichsobjekt in die nächste Generation zu übernehmen. Ein weitere Vorteil, neben der Vermeidung der Verschlechterung, ist dabei der, dass, im Falle einer Verschlechterung (generierte Routen der neuen Generation sind deutlich schlechter als beste übernommene Routen), die übernommenen besten Routen der vorherigen Generation mehrmals in die Justierung der Gewichte einfließen und somit die Wahrscheinlichkeit einer Verbesserung in der nächsten Generation erhöhen.

2.6 Optimierung

Einer der aufwendigsten Prozesse ist die permanente Berechnung von Winkel und Distanzen beim Generieren der Routen, welche das Bestimmen von Werten trigonometrischer Funktionen oder das Bestimmen von Wurzeln erfordert. Da diese Berechnungen häufig redundant sind, ist es naheliegend, die berechneten Werte zu speichern und bei Bedarf abzurufen. Für die Distanzen ergibt sich eine zweidimensionale Matrix, mit jeweils Start- und Endpunkt als Eingabe und der Distanz als Ausgabe. In Python kann dies durch eine zweidimensionale Liste ermöglicht werden, wobei ein Punkt als Index der ersten Dimension und der zweite Punkt als Index der zweiten Dimension dient. Für die Winkel ist ähnliches Möglich, wobei drei Dimensionen für die drei Punkte zur Berechnung eines Winkels nötig sind.

Durch die Berechnung aller Distanzen und Winkel im Vorhinein ist das Generieren vieler Routen deutlich schneller und somit das Durchlaufen vieler Generationen praktischer.

3 Beispiele

Ausgaben meines Programms zu den Beispieleingaben der BWInfo-Webside (https://bwinf.de/bundeswettbewerb/41/2/):

wenigerkrumm1.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 847.4341649025257km
Route: [[400.0, 30.0], [390.0, 30.0], [380.0, 30.0], [370.0, 30.0], [360.0, 30.0], [350.0, 30.0], [340.0, 30.0], [330.0, 30.0], [320.0, 30.0], [310.0, 30.0], [300.0, 30.0], [290.0, 30.0], [280.0, 30.0], [270.0, 30.0], [260.0, 30.0], [250.0, 30.0], [240.0, 30.0], [230.0, 30.0], [220.0, 30.0], [210.0, 30.0], [200.0, 30.0], [190.0, 30.0], [180.0, 30.0], [170.0, 30.0], [160.0, 30.0], [150.0, 30.0], [140.0, 30.0], [130.0, 30.0], [120.0, 30.0], [110.0, 30.0], [100.0, 30.0], [90.0, 30.0], [80.0, 30.0], [70.0, 30.0], [60.0, 30.0], [50.0, 30.0], [40.0, 30.0], [30.0, 30.0], [20.0, 30.0], [10.0, 30.0], [0.0, 30.0], [-5.0, 15.0], [0.0, 0.0], [10.0, 0.0], [20.0, 0.0], [30.0, 0.0], [40.0, 0.0], [50.0, 0.0], [60.0, 0.0], [70.0, 0.0], [80.0, 0.0], [90.0, 0.0], [100.0, 0.0], [110.0, 0.0], [120.0, 0.0], [130.0, 0.0], [140.0, 0.0], [150.0, 0.0], [160.0, 0.0], [170.0, 0.0], [180.0, 0.0], [190.0, 0.0], [200.0, 0.0], [210.0, 0.0], [220.0, 0.0], [230.0, 0.0], [240.0, 0.0], [250.0, 0.0], [260.0, 0.0], [270.0, 0.0], [280.0, 0.0], [290.0, 0.0], [300.0, 0.0], [310.0, 0.0], [320.0, 0.0], [330.0, 0.0], [340.0, 0.0], [350.0, 0.0], [360.0, 0.0], [370.0, 0.0], [380.0, 0.0], [390.0, 0.0], [400.0, 0.0], [405.0, 15.0]]

wenigerkrumm2.txt:
Output:
Anzahl spitzer Winkel: 0

Länge der Route: 2183.662266704193km
Route: [[-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], [-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], [-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], [-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]]

wenigerkrumm3.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 1856.3256060041856km
Route: [[0.0, 80.0], [-16.632935, 78.251808], [-32.538931, 73.083637], [-47.02282, 64.72136], [-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], [47.02282, 164.72136], [52.97718, 164.72136], [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], [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], [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], [-47.02282, 35.27864], [-32.538931, 26.916363], [-16.632935, 21.748192], [0.0, 20.0], [16.632935, 21.748192], [23.915479, 24.72136], [32.538931, 26.916363], [30.717968, 40.0], [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], [59.451586, 153.530449], [69.282032, 140.0], [76.084521, 124.72136], [79.561752, 108.362277], [79.561752, 91.637723], [76.084521, 75.27864], [67.461069, 73.083637], [52.97718, 64.72136], [47.02282, 64.72136], [40.548414, 53.530449], [40.548414, 46.469551], [47.02282, 35.27864],

[52.97718, 35.27864], [67.461069, 26.916363], [76.084521, 24.72136], [83.367065, 21.748192], [100.0, 20.0], [116.632935, 21.748192], [132.538931, 26.916363], [147.02282, 35.27864], [147.02282, 64.72136], [132.538931, 73.083637], [116.632935, 78.251808], [100.0, 80.0], [83.367065, 78.251808], [69.282032, 60.0], [59.451586, 53.530449], [59.451586, 46.469551], [69.282032, 40.0], [79.561752, 8.362277], [79.561752, -8.362277], [76.084521, -24.72136], [69.282032, -40.0], [59.451586, -53.530449], [40.548414, -53.530449], [30.717968, -40.0], [23.915479, -24.72136], [20.438248, -8.362277], [20.438248, 8.362277]]

wenigerkrumm4.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 1205.068555245931km
Route: [[42.137753, -60.319863], [94.789917, -67.087689], [144.832862, -43.476284], [153.130159, -20.36091], [139.446709, 0.233238], [101.498782, 33.484198], [51.00814, 5.769601], [-16.72313, -12.689542], [-20.971208, -5.637107], [28.913721, 58.69988], [33.379688, 100.161238], [20.212169, 156.013261], [-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]]

wenigerkrumm5.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 3568.612621430512km
Route: [[-234.711279, -162.774591], [-247.341131, -160.277639], [-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], [-70.183535, 73.738342], [36.599805, 147.88535], [47.040512, 141.206562], [51.417675, 146.417721], [38.65473, 188.608557], [-30.991436, 186.807059], [-49.447381, 173.210759], [-68.446198, 137.178953], [-82.17351, 119.465553], [-86.45758, 105.836348], [-81.384378, 32.368323], [-74.639411, 25.542881], [-8.936916, 13.543851], [25.098172, 35.205544], [45.123674, 31.740242], [92.639946, 22.21603], [141.513053, 2.821137], [162.493244, -84.574019], [142.765554, -118.682439], [90.584569, -164.218416], [30.366828, -167.573232], [-41.263039, -144.118212], [-57.266232, -115.737582], [-58.684205, -76.988884], [-31.548604, -55.223912], [62.366656, 50.713913], [63.541591, 55.140221], [106.03343, 69.754891], [209.544977, 94.267052], [239.63955, 79.491132], [253.534863, 38.014987], [283.989938, -101.866465], [263.236651, -144.293091], [26.451074, -192.813352], [-116.831788, -191.380552], [-147.023475, -166.22013], [-202.218443, -178.735864], [-214.362324, -193.26519], [-251.656688, -195.189953], [-260.477802, -196.955535], [-281.67899, -187.717923], [-258.868593, 166.669198], [-223.039999, 171.558368], [-177.685937, 158.265884], [-104.781549, 158.212048], [-44.669924, 170.088013], [27.706327, 169.284192], [116.702667, 132.021991], [171.595574, 135.520994], [244.228552, 119.192512], [267.845908, 127.627482]]

wenigerkrumm6.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 4120.190524622865km
Route: [[238.583388, -133.143524], [221.028639, -139.435079], [210.186432, -127.403563],

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

wenigerkrumm7.txt:
Output:
Anzahl spitzer Winkel: 0
Länge der Route: 5202.514190633359km
Route: [[178.19836, 37.031984], [158.742184, 62.618834], [120.375033, 115.889661], [100.043893, 161.295125], [95.947213, 183.278211], [130.854855, 195.695082], [217.218893, 164.294928], [216.691, 156.31437], [208.592696, 136.61846], [162.923138, 117.465744], [148.108328, 123.558283], [126.799911, 112.72728], [74.8875, 80.586458], [54.551865, 71.567133], [40.897139, 78.152317], [16.573231, 104.020979], [8.64366, 135.90743], [-13.030259, 174.698005], [-114.146166, 190.615321], [-153.13014, 187.817274], [-192.681053, 174.522947], [-222.492322, 169.033315], [-240.249363, 179.334919], [-268.739142, 143.276483], [-287.058297, 113.599823], [-284.129027, 107.252583], [-248.169463, 80.132237], [-207.665172, 81.410371], [-185.649161, 90.144456], [-189.062172, 104.285631], [-200.771246, 147.741054], [-201.485143, 155.27483], [-120.386562, 170.589454], [-84.6269, 148.216494], [55.434792, 62.72916], [57.555364, 64.417343], [56.389778, 71.618509], [53.436521, 125.683201], [54.766523, 154.053847], [34.079032, 187.731112], [-134.985023, 132.944989], [-157.423365, 126.800331], [-215.113949, 120.740679], [-211.429137, 27.770425], [-226.787625, 2.658862], [-217.28247, -43.316616], [-244.959501, -111.046573], [-181.208895, -192.622935], [59.8272, -170.713714], [92.29804, -146.169487], [106.599423, -107.433987], [118.989764, -80.203583], [126.904044, -80.733297], [172.389228, -53.13327], [205.717887, -24.976511], [239.616628, -21.94416], [256.475967, -46.591418], [276.276517, -49.448662], [278.105364, -93.771765], [275.793495, -129.415477], [235.827007, -143.838844], [217.599278, -189.258062], [-17.356579, -125.254131], [-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], [88.818853, -42.834512], [135.781192, -13.05344], [141.433472, -6.023095], [158.552316, -19.254407], [189.387028, -4.465225], [224.599361, -34.798485], [296.911892, 25.811569], [271.301094, 142.524086], [221.808162, 186.241012], [170.514252, 161.16985], [169.990437, 154.260412], [129.024315, 29.701695], [-133.730932, -

113.306155], [-152.130365, -93.844349], [-172.378071, -88.298187], [-202.828627, -101.70005], [-189.135201, -139.078513], [-155.651746, -138.151811], [34.959132, -106.842499], [68.910854, -82.123346], [55.550895, -45.089968], [47.011363, -30.88718], [-0.200936, -21.927663], [-9.580869, -17.516639], [-18.316063, 27.75586], [-56.914543, 92.501249], [-126.568914, 106.964962], [-147.363185, 59.608175], [-171.354954, 25.463068], [-184.0927, 5.737284]]

4 Quellcode: Implementierung in Python

Nachfolgend ist das Hauptprogramm meiner Implementierung in Python 3.10:

```
import sidefunc as sf
import numpy as np
from numpy.random import choice as np_choice
import random

class DrohnenKolonie():
    """
    Die Klasse DrohnenKolonie dient dem Generieren von einer möglichst optimalen Route durch eine Sammlung von Punkten via. ant-colony-optimization

    Variable Einstellungen:
    generieren_von_routen_einflussfaktoren(Winkel, Distanz, Gewichte, zusätzliche Polarisierung): Gewichtung verschiedener Faktoren bei dem Wählen des nächsten Punktes beim Generieren einer Route
    gewichtung_bewertung: Polarization bei der Bewertung der Routen
    menge_der_gespeicherten_routen: Menge der gespeicherten Top-Routen der vorherigen Generation (werden in nächste Generation als Vergleich übernommen) -> verhindert Verschlechterung
    verblassgeschwindigkeit_gewichte: Faktor
    """
    def __init__(self, punkte, generieren_von_routen_einflussfaktoren=[5, 2, 3, 3], gewichtung_bewertung=2, menge_der_gespeicherten_routen=3, verblassgeschwindigkeit_gewichte=0.85):
        """
        Initialisierung und Instanziierung der Einstellung dieses Algorithmus
        """
        self.punkte = punkte
        #Optimierung durch präventive Berechnung aller Winkel und Distanzen die sich in den Punkten ergeben können und Speicherung in einer Matrix
        self.distanzen_matrix = [[sf.calculate_distance(p1, p2) for p2 in punkte] for p1 in punkte]
        self.winkel_matrix = [[[sf.calculate_angle(p1, p2, p3) for p3 in punkte] for p2 in punkte] for p1 in punkte]

        #Initialisierung der konstanten Einstellungen
        self.verblassgeschwindigkeit_gewichte = verblassgeschwindigkeit_gewichte

        self.einflussfaktor_winkel = generieren_von_routen_einflussfaktoren[0]
        self.einflussfaktor_distanz = generieren_von_routen_einflussfaktoren[1]
        self.einflussfaktor_gewichte = generieren_von_routen_einflussfaktoren[2]
        self.zusaetzliche_polarisierung = generieren_von_routen_einflussfaktoren[3]

        self.polarization_inder_bewertung_von_routen = gewichtung_bewertung
        self.menge_der_gespeicherten_besten_routen = menge_der_gespeicherten_routen


    def generiere_route(self, anzahl_generationen=150, routen_pro_generation=25):
        """
        generiere_route generiert mithilfe eines evolutionär-inspierierten Algorithmus eine möglichst kurze Route durch eine Menge an Punkten
        mit möglichst keinen spitzen Winkeln, wobei Generationen mit jeweils n Routen generiert werden, die jeweils je nach Performance Gewichte anpassen/justieren, welche den nachfolgenden Generationen als Wegweiser dienen.
        """
        #Instanziierung der Gewichte.
        self.strecken_gewichtungen = np.ones((len(punkte), len(punkte)))
        beste_route = None
```

```
        vorherige_beste_routen = None
        for generation in range(anzahl_generationen):
                #Generieren aller Routen einer Generation.
                routen_einer_generation = self.generiere_routen_einer_generation(routen_pro_generation,
                vorherige_beste_routen)
                #Anpassen der Gewichte je nach Performance der Route in der Generation.
                self.anpassen_der_gewichte(routen_einer_generation)
                #Ermittlung der besten Routen der neuen Generation.
                vorherige_beste_routen = sorted(routen_einer_generation, key=lambda x:
                x[1])[:self.menge_der_gespeicherten_besten_routen]
                beste_route_aktuelle_generation = min(routen_einer_generation, key=lambda x: x[1])
                #Eventuelle Aktualisierung der insgesamt besten Route (falls beste Route der Generation besser ist als die
                #insgesamt beste bisher gefundene Route).
                if beste_route == None:
                        beste_route = beste_route_aktuelle_generation
                elif beste_route_aktuelle_generation[1] < beste_route[1]:
                        beste_route = beste_route_aktuelle_generation
                #Verblassen der Gewichte, sodass der Einfluss der älteren Justierungen geringer für das Generieren neuer
                #Routen ist, als die Justierungen neuerer Generationen.
                self.verblassen_der_gewichte()
        #Schließlich wird die beste gefundene Route zurückgegeben.
        return beste_route[2], sf.count_distances(beste_route[0], self.distanzen_matrix), [self.punkte[p] for p in
        beste_route[0]]

def generiere_routen_einer_generation(self, routen_pro_generation, vorherige_beste_routen):
        """
        generiere_routen_einer_generation generiert routen_pro_generation viele Routen unter Beachtung der
        entsprechenden Gewichte
        """
        routen_einer_generation = []
        for nte_route in range(routen_pro_generation):
                #Für jede Route wird ein zufälliger Startpunkt gewählt von welchem aus die Route generiert und dann
                #gespeichert wird.
                zufaelliger_startpunkt = random.randint(0, len(self.punkte)-1)
                route = self.generiere_einzelne_route_mit_startpunkt(zufaelliger_startpunkt)
                        routen_einer_generation.append((route, self.generiere_punktzahl_der_route(route),
                        sf.count_acute_angles(route, self.winkel_matrix)))
        #Anschließend werden die vorherigen besten Routen der Generation, als Vergleich, hinzugefügt.
        if vorherige_beste_routen != None:
                for route in vorherige_beste_routen:
                        routen_einer_generation.append((route[0], route[1], route[2]))
        return routen_einer_generation

def generiere_einzelne_route_mit_startpunkt(self, startpunkt):
        """
        generiere_einzelne_route_mit_startpunkt generiert eine Route vom Startpunkt startpunkt unter Beachtung der
        entsprechenden Gewichte
        """
        #Initialisierung der variablen zur speicherung der neuen Route und der bereits besuchten Punkte
        route = [startpunkt]
        bereits_besucht = set()
        bereits_besucht.add(startpunkt)
        vorheriger_punkt = startpunkt
        for n in range(len(self.punkte)-1):
                #Solange nicht alle Punkte in der Route erreicht wurden: Füge weiteren Punkt der Route hinzu...
                if len(route) > 1:
                        naechster_punkt =
                        self.waehle_naechsten_punkt(self.strecken_gewichtungen[vorheriger_punkt], bereits_besucht,
                        route[-1], route[-2])
                else:
                        #Falls die Route noch weniger als zwei Punkt enthält, können keine Winkel berechnet werden.
```

```
                naechster_punkt =
                self.waehle_naechsten_punkt(self.strecken_gewichtungen[vorheriger_punkt], bereits_besucht,
                route[-1])
            #Füge den neuen Punkt der Route hinzu und entferne ihn als zukünftige Option
            route.append(naechster_punkt)
            bereits_besucht.add(naechster_punkt)
            vorheriger_punkt = naechster_punkt
        #Gebe die fertige Route zurück
        return route

def waehle_naechsten_punkt(self, strecken_gewichtungen, bereits_besucht, vorheriger_punkt1, vorheriger_punkt2=None):
        """
        waehle_naechsten_punkt wählt aus den möglichen (noch nicht erreichten) Punkten einen zufälligen aus
        unter Beachtung der entsprechenden Gewichte, Winkel und Distanzen.
        """
        #Übernehme die gespeicherten Gewichte für die Strecken vom derzeitigen Punkt
        gewichtungen_fuer_moegliche_naechste_punkte = np.copy(strecken_gewichtungen)
        #Entfernen der bereits benutzten Optionen
        gewichtungen_fuer_moegliche_naechste_punkte[list(bereits_besucht)] = -0.1
        #Jeder Option eine Wahrscheinlichkeit geben und strecken_gewichtungen anpassen
        gewichtungen_fuer_moegliche_naechste_punkte += 1.1
        gewichtungen_fuer_moegliche_naechste_punkte = gewichtungen_fuer_moegliche_naechste_punkte **
        self.einflussfaktor_gewichte
        gewichtungen_fuer_moegliche_naechste_punkte -= 1
        gewichtungen_fuer_moegliche_naechste_punkte /= max(gewichtungen_fuer_moegliche_naechste_punkte)
        #Wahrscheiblichkeiten anpassen durch Winkel & Distanzen
        for moeglicher_naechster_punkt_index in range(len(gewichtungen_fuer_moegliche_naechste_punkte)):
            if gewichtungen_fuer_moegliche_naechste_punkte[moeglicher_naechster_punkt_index] > 0: #nur < 0,
            #wenn bereits in Route enthalten
                if vorheriger_punkt2 != None:
                    gewichtungen_fuer_moegliche_naechste_punkte[moeglicher_naechster_punkt_ind
                    ex] *=
                    (self.winkel_matrix[vorheriger_punkt2][vorheriger_punkt1][moeglicher_naechster_
                    punkt_index]/360.0) ** self.einflussfaktor_winkel *
                    (1.0/self.distanzen_matrix[vorheriger_punkt1][moeglicher_naechster_punkt_index])
                    ** self.einflussfaktor_distanz
                else:
                    #Falls die Route noch weniger als zwei Punkt enthält, können keine Winkel
                    #berechnet werden.
                    gewichtungen_fuer_moegliche_naechste_punkte[moeglicher_naechster_punkt_ind
                    ex] *=
                    (1.0/self.distanzen_matrix[vorheriger_punkt1][moeglicher_naechster_punkt_index])
                    ** self.einflussfaktor_distanz
        angepasste_gewichtungen = gewichtungen_fuer_moegliche_naechste_punkte ** self.zusaetzliche_polarisierung
        #Wahrscheinlichkeiten normalisieren (Summe aller Wahrscheinlichkeiten = 1)
        normalisierte_angepasste_gewichtungen = angepasste_gewichtungen / sf.sum(angepasste_gewichtungen)
        #Fehlerbehebung
        if np.isnan(normalisierte_angepasste_gewichtungen).any():
            normalisierte_angepasste_gewichtungen =
            np.ones(len(normalisierte_angepasste_gewichtungen))/len(normalisierte_angepasste_gewichtungen)
        #Zufällige Wahl des nächsten Punktes unter Beachtung der berechneten Gewichtungen
        naechster_punkt = np_choice(range(len(self.punkte)), 1, p=normalisierte_angepasste_gewichtungen)[0]
        #Rückgabe des gewählten Punktes
        return naechster_punkt

def generiere_punktzahl_der_route(self, route):
        """
        generiere_punktzahl_der_route berechnet die Punktzahl einer Route aus der Menge ihrer spitzen Winkel und ihrer
        Länge,
        wobei eine Route mit weniger spitzen Winkeln immer besser ist als eine kürzere Route.
        """
        return (sf.count_acute_angles(route, self.winkel_matrix) + 1) * sf.count_distances(route, self.distanzen_matrix)
```

```
def anpassen_der_gewichte(self, generation):
        """
        anpassen_der_gewichte justiert die Gewichte entsprechend der verhältnismäßigen Performance der Routen einer
        Generation.
        """
        #Sortieren der Routen der Generation entsprechend deren Punktzahl
        sortierte_routen = sorted(generation, key=lambda x: x[1])
        for routen_index in range(len(sortierte_routen)-1, -1, -1):
                #Berechnung der verhältnismäßigen Performance einer Route (im Vergleich zu besten Route)
                verhaeltnismaeszige_performance_faktor = (sortierte_routen[0][1]/sortierte_routen[routen_index][1]) **
                self.polarization_inder_bewertung_von_routen
                if routen_index == 0 and sortierte_routen[1][2] > sortierte_routen[0][2]:
                        #Falls eine Route mit Abstand die beste war: Vergebe Bonus-Punkte
                        verhaeltnismaeszige_performance_faktor *= 2.5
                if verhaeltnismaeszige_performance_faktor < 1e-4:
                        #Falls eine Route zu schlecht war: ziehe sie nicht in die Bewertung mit ein
                        continue
                #Justiere die entsprechenden Gewichte der Route gemäß ihrer Performance
                for index_punkt_in_route in range(1, len(sortierte_routen[routen_index][0])):
                        #Ignoriere dabei Strecken die für einen spitzen Winkel sorgten (zusätzliche 'Bestrafung' spitzer
                        #Winkel).
                        if index_punkt_in_route < 2 or
                        self.winkel_matrix[sortierte_routen[routen_index][0][index_punkt_in_route-
                        2]][sortierte_routen[routen_index][0][index_punkt_in_route-
                        1]][sortierte_routen[routen_index][0][index_punkt_in_route]] >= 90:
                                self.strecken_gewichtungen[sortierte_routen[routen_index][0][index_punkt_in_rout
                                e-1]][sortierte_routen[routen_index][0][index_punkt_in_route]] +=
                                verhaeltnismaeszige_performance_faktor
        #Normalisieren aller Gewichte
        for i in range(len(self.strecken_gewichtungen)):
                self.strecken_gewichtungen[i] /= sf.sum(self.strecken_gewichtungen[i])

def verblassen_der_gewichte(self):
        """
        verblassen_der_gewichte multipliziert alle Gewichte mit einem Faktor (<1), sodass 'ältere' Justierungen mit den
        Generationen einen geringeren Einfluss haben.
        """
        for gewichte_eines_punktes in self.strecken_gewichtungen:
                gewichte_eines_punktes *= self.verblassgeschwindigkeit_gewichte
```