Implementierung des Hill-Climber-Algorithmus für eine Anwendung in der Logistik

0. Link zum Git Repository https://github.com/ilyinariana/hillClimber-new

1. Ziele

Das Projekt zielte darauf ab, den Hill-Climber-Algorithmus zu implementieren, um mithilfe der Entfernung zwischen 80 Städten eine gute Lösung für das Problem des Handlungsreisen zu finden. Der bestehende Code liest die Distanzen ein und ermittelt die kürzeste Route, indem er über alle möglichen Wege iteriert. Dabei variiert die Reihenfolge der Städte durch Vertauschen.

2. Methoden

Das Problem des Handlungsreisenden wurde in diesem Projekt mithilfe des Hill-Climber-Algorithmus erfolgreich gelöst. In diesem Algorithmus geht es darum, die kürzeste Route zu finden, die eine bestimmte Anzahl von Städten genau einmal besucht und zum Ausgangspunkt zurückkehrt [1]. Die Herangehensweise basiert darauf, eine lokale optimale Lösung durch wiederholte Verbesserungen zu finden, indem zwei Städte ausgetauscht und die daraus resultierenden Routenlängen verglichen werden.

1. Datei durch “import panda” importieren: Im ersten Schritt werden die in Excel-Formaten vorgegebenen Daten durch die Panda-Bibliothek verwendet.
2. Der Algorithmus startet mit einer initialen Lösung, die den Besuch aller Städte in einer vorgegebenen Reihenfolge vorsieht. In diesem Fall wird eine Sequenz von Städten von 1 bis 79 verwendet.
3. In diesem Schritt werden zwei benachbarte Städte von der Route vertauscht, um eine potenziell kürzere Route zu finden.
4. Berechnung der Gesamtdistanz der neuen Route
   Wichtige Formeln:
   - Berechnung der Gesamtdistanz einer Route:

     $$\text{Gesamtdistanz} = \sum_{i=1}^{n-1} d(\text{City}[i], \text{City}[i+1])$$

   Wobei d(City [i], City[i + 1]) die Distanz zwischen den Städten city[i] und city[i + 1] ist.

   - Vertauschen von zwei Städten in der Route
     newJourney[i], newJourney[j] = newJourney[j], newJourney[i]

   - Distanzberechnung zwischen zwei Städten:
     d(cityA, cityB) = distanceTable.loc[cityA - 1][cityB]

5. Die neue Route wird angenommen, wenn sie kürzer ist als die bisher beste gefundene Route.

6. Schleife “for j in range(len(initial_journey) - 1)”: Diese Schleife läuft für jedes i mit j im Bereich von 80 bis len(initial_journey) - 1. Dies gewährleistet, dass alle Städtepaare nur einmal ausgetauscht werden.

3. Ergebnisse

Die wichtigsten Ergebnisse des Projekts sind wie folgt:

- Hill-Climber-Algorithmus: Der umgesetzte Hill-Climber-Algorithmus startete mit einer ursprünglichen Route und iterierte über potenzielle Nachbarlösungen durch Vertauschen von Städten. Eine verbesserte Route wurde vom Algorithmus gefunden, aber nur durch lokale Optimierungen. Dies kann zu einer nicht optimalen Lösung führen.
- Simulated Annealing Algorithmus: Bei jeder Iteration werden zwei zufällige Städte ausgewählt und vertauscht, um eine neue Reise zu erstellen. Der Hill-Climber-Algorithmus wurde an einen Simulated Annealing Algorithmus angepasst, um ein besten gefundenen Lösungen zu erreichen. Dadurch wurden auch schlechtere Lösungen wahrscheinlicher akzeptiert, was es ermöglichte, aus lokalen zu entgehen.
- Vergleich der Algorithmen: Der Hill-Climber-Algorithmus ist schneller, weil er nur lokale Verbesserungen durchführt. Aufgrund der probabilistischen Akzeptanzregel braucht der Simulated Annealing Algorithmus längere Zeit. Jedoch zeigt Simulated Annealing tendenziell eine höhere Qualität an Lösungen, da es eine eingehendere Suche im Lösungsraum ermöglicht.

4. Diskussion

Es wurde erwartet, dass der Algorithmus durch einen systematischen Austausch der Positionen der Städte in der Rundreise eine bessere Lösung als die zufällige Ausgangslösung finden würde. Dies entspricht dem üblichen Verhalten des Hill-Climber-Algorithmus, der durch lokale Verbesserungen zu einer lokalen Optimumslösung führt [2].
Die Ergebnisse, die ermittelt wurden, entsprechen den Erwartungen. Im Gegensatz zur Ausgangslösung hat der Algorithmus eine verbesserte Rundreise mit einer verringerten Gesamtdistanz gefunden. Dies entspricht den Erwartungen des Algorithmus von Hill Climber, der tendenziell lokale Optimierung vornimmt [3].
Einige offene Fragen könnten sein; Wie robust ist der Algorithmus in Bezug auf unterschiedliche Startoptionen und -parameter? Und wie beeinflussen Veränderungen in der Distanzmatrix die gefundenen Lösungen?
Zur Beurteilung der Robustheit des Algorithmus werden Sensitivitätsanalysen durchgeführt. Außerdem sollten unterschiedliche Versionen des Algorithmus von Hill Climber oder anderen Methoden zur Optimierung versucht werden.

Literaturverzeichnis

[1]Smith, J. (2015). Optimization algorithms for the Travelling Salesman Problem. *Journal of Computational Optimization, 23*(4),

[2],[3]Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). Cambridge, MA: MIT Press.