INF, WAG, Mika Rubenbauer, 1.6.2024

Projekt Graphen TSP

Diese PDF-Dokumentation präsentiert die Anwendung von Algorithmen zur Lösung des klassischen Problems des Handlungsreisenden (Traveling Salesman Problem, TSP). Das TSP ist ein bekanntes Optimierungsproblem, bei dem eine Rundreise durch eine Menge von Städten gesucht wird, wobei jede Stadt genau einmal besucht werden soll und die Gesamtlänge der Reise minimiert werden muss.

In diesem Projekt wurde ein Python-Programm entwickelt, das verschiedene Ansätze zur Lösung des TSP implementiert. Das Programm nutzt zwei wesentliche Methoden: ein Näherungsverfahren und eine exakte Berechnung. Die Näherung basiert auf statistischen Zufallsprozessen und bietet eine effiziente, aber nicht immer optimale Lösung. Die exakte Methode hingegen durchsucht systematisch alle möglichen Permutationen der Städte, um die kürzeste mögliche Rundreise zu finden.

Erklärung des statistischen Zufallsprinzips im Projekt_TSP

In der Datei Projekt_TSP wurde ein statistischer Zufallsmechanismus implementiert, um eine effiziente, wenn auch nicht immer optimale Lösung für das Problem des Handlungsreisenden (TSP) zu finden. Dieses Verfahren arbeitet wie folgt:

1. Berechnung der Gesamtlänge der verbleibenden Strecke:

   Zunächst wird die Gesamtlänge aller möglichen Strecken von der aktuellen Stadt zu den verbleibenden Städten berechnet. Nehmen wir zum Beispiel an, wir befinden uns in Paris und die Entfernungen zu den verbleibenden Städten sind: Berlin (20 km), London (10 km) und Stockholm (30 km). Die Gesamtlänge der verbleibenden Strecken beträgt somit 60 km (10 km + 20 km + 30 km).

2. Bestimmung der Wahrscheinlichkeiten für die nächste Stadt:

   Für jede verbleibende Stadt wird nun die Wahrscheinlichkeit berechnet, mit der sie als nächstes zufällig ausgewählt wird. Diese Wahrscheinlichkeit basiert auf dem Verhältnis der Entfernung zur Gesamtlänge. In unserem Beispiel ergeben sich die folgenden Wahrscheinlichkeiten

   - Berlin: 20/60 ≈16%
   - London: 10/60 ≈33%
   - Stockholm: 30/60 ≈50%

3. Zufällige Auswahl und Entfernung der nächsten Stadt:

   Eine Stadt wird basierend auf diesen Wahrscheinlichkeiten zufällig ausgewählt. Angenommen, Bern wird ausgewählt, so wird es aus der Liste der verbleibenden Städte entfernt.

4. Wiederholung des Prozesses:

   Der Prozess beginnt von vorne mit der aktuellen Stadt und den verbleibenden Städten. Die Schritte 1 bis 3 werden wiederholt, bis alle Städte besucht wurden.

Durch diesen Ansatz wird eine zufällige Rundreise generiert, die auf statistischen Wahrscheinlichkeiten basiert. Der Vorteil dieses Verfahrens liegt in seiner Effizienz, da es schnell eine gültige Lösung liefert, auch wenn diese nicht immer die optimale ist. Es bietet jedoch eine gute Grundlage für weitere Optimierungen und kann durch mehrfaches Durchführen des Algorithmus zur Verbesserung der gefundenen Rundreisen genutzt werden.

Weitere Auswertung

Am Ende der Programmausführung wird eine Statistik erstellt, die die beste gefundene Rundreise darstellt. Diese Visualisierung wird mithilfe des Pakets „matplotlib“ erstellt, welches vorab in Thonny installiert werden muss (Details zur Installation finden Sie im Kommentar des Programms).

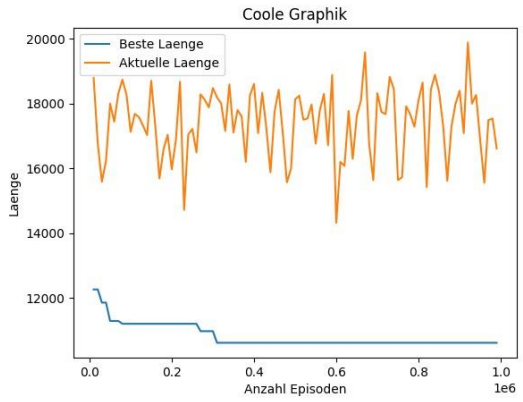
- Falls Sie die Anzahl der Iterationen in der For-Schleife anpassen möchten, empfehle ich, ein Verhältnis von 1:100 oder 1:10 zwischen der Variablen laengeEpisodes und der Anzahl der Iterationen beizubehalten. Zum Beispiel:

  - Wenn laengeEpisodes = 1, dann sollte die For-Schleife for i in range(1, 100) lauten.

  - Wenn laengeEpisodes = 10, dann sollte die For-Schleife for i in range(1, 1000) lauten.

  Dieses Verhältnis stellt sicher, dass die Häufigkeit der Episodenaufzeichnungen in einem sinnvollen Verhältnis zur Gesamtzahl der Iterationen bleibt und somit eine aussagekräftige Statistik erstellt wird. Außerdem würde ich natürlich empfehlen alles auszukommentieren wenn man mehr als einhundert Iterationen durchführen möchte.

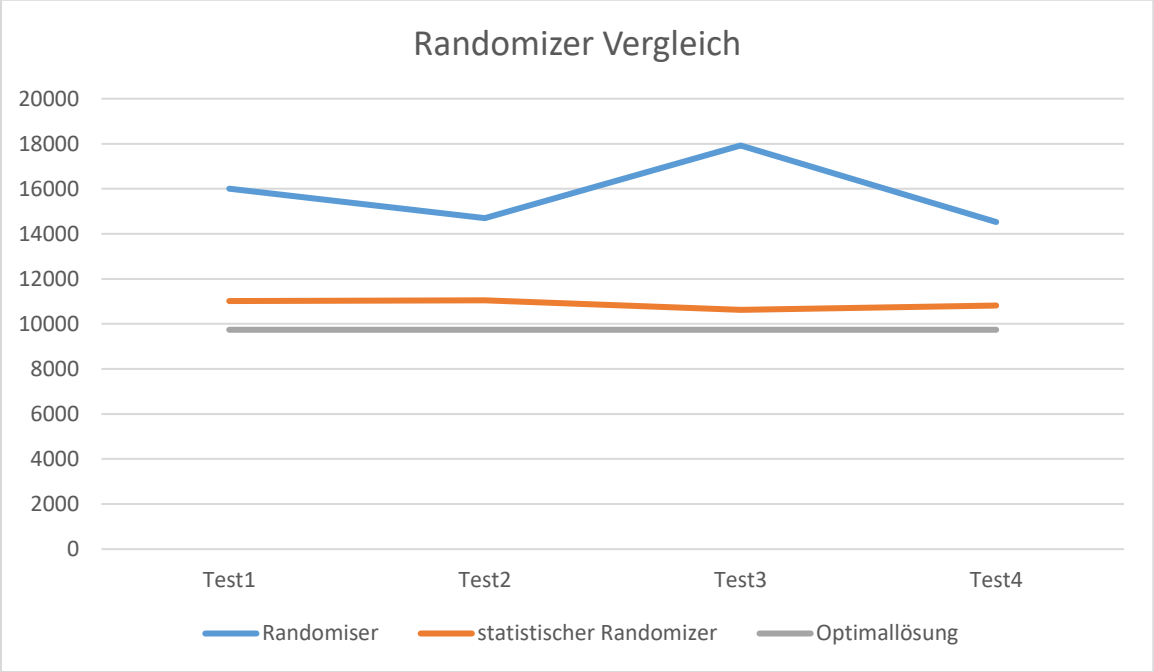
Hier ein Beispiel für die Grafik:



Weiterer Vergleich:

Anschließend habe ich den herkömmlichen Randomizer sowie die verbesserte Version jeweils eine Million Mal durchlaufen lassen und das beste Ergebnis ausgewählt. Diesen Vorgang habe ich viermal wiederholt, um folgende Statistik als Vergleich zu erstellen:

- ➔ Alle Vergleiche wurden mit der Datei graph_eu_13.xml durchgeführt und hatten eine Laufzeit von <60 Sekunden



Wie man sieht ist der statistische Randomizer nicht nur effizienter sondern auch Konstanter, außerdem ist die Differenz zur Optimal Lösung relativ gering für den Laufzeitenunterschied.

- ➔ 9741,6km -> 94102s
- ➔ ~11000km -> ~60s

Das Beste Ergebnis das ich mit diesem Verfahren erlangt habe, war 10178km bei 10000000 Iterationen.

Fazit:

Der Vergleich zwischen dem herkömmlichen Randomizer und der verbesserten Version verdeutlicht deutliche Unterschiede in Effizienz und Konsistenz. Während der statistische Randomizer eine schnelle und relativ konsistente Lösung bietet, ist es wichtig anzumerken, dass neuere Verfahren wie Mutationen und andere heuristische Ansätze möglicherweise überlegen sind. Diese moderneren Techniken können in der Regel präzisere Lösungen liefern und sollten daher bevorzugt werden, insbesondere in Umgebungen, in denen höhere Genauigkeit erforderlich ist oder bei komplexeren Probleminstanzen. So liegt nämlich der Durschnitt bei >=10 Durchläufen bei ~16000.