1 Lernziele

Allgemein Es soll eine Problemstellung aus dem Bereich der wissensbasierten Systeme eigenständig in einer Gruppe bearbeitet werden. Dazu ist eine Einarbeitung in das Themengebiet und das Erstellen eines Projektplans notwendig. Die entwickelten Lösungen sollen implementiert und in einem konkreten Anwendungsfall analysiert werden. Dabei ist das Problem mit Fachwissen unter Anwendung geeigneter Methoden und Verfahren entsprechend des aktuellen Stands der Technik zu lösen, das Projekt zu planen, durchzuführen, abzuschließen und zu dokumentieren.

Dieses Projekt sollte von drei oder vier Studierenden bearbeitet werden.

Speziell In diesem Projekt sollen die Studierenden die verschiedenen Ansätze zur Lösung des Traveling-Salesman-Problems (TSP) kennenlernen und implementieren. Durch die Anwendung von lokaler Suche, dynamischer Programmierung und ganzzahliger linearer Programmierung soll ein tieferes Verständnis für die Herausforderungen und Möglichkeiten der Problemlösung in der Informatik entwickelt werden. Die Tests mit Beispielen aus der TSP-LIB ermöglichen eine praktische Evaluierung der Algorithmen und deren Vergleich hinsichtlich Effizienz und Genauigkeit.

1.1 Traveling Salesperson Problem

Beim Traveling-Salesman-Problem[1] (TSP) ist die kürzeste mögliche Route zu finden, die eine Anzahl von Städten besucht, wobei jede Stadt genau einmal besucht wird, bevor man zum Ausgangspunkt zurückkehrt. Das TSP ist ein klassisches Beispiel für ein kombinatorisches Optimierungsproblem und hat Anwendungen in verschiedenen Bereichen, wie Logistik, Routenplanung und Netzwerkdesign. Unternehmen nutzen TSP-ähnliche Modelle, um die Effizienz ihrer Lieferketten zu verbessern oder um Routen für Lieferfahrzeuge zu optimieren.

Das Problem ist bekannt dafür, dass es für eine große Anzahl an Städten sehr schwierig zu lösen ist, da die Anzahl der möglichen Routen exponentiell ansteigt. Es gehört zur Klasse der NP-schweren Probleme, was bedeutet, dass es keine bekannte effiziente Methode gibt, um die optimale Lösung für große Instanzen des Problems in akzeptabler Zeit zu finden.

1.2 TSP-LIB

Die TSP-LIB[2] ist eine Sammlung von Benchmark-Daten für TSP und verwandte Probleme. Sie wurde entwickelt, um Forschern und Praktikern eine standardisierte Basis zu bieten, um Algorithmen zur Lösung des TSP zu testen und zu vergleichen. Die TSP-LIB enthält verschiedene Datensätze, die in einem einheitlichen Format vorliegen. Diese Datensätze umfassen Informationen über die Städte, deren Koordinaten und die Entfernungen zwischen ihnen. Die TSP-LIB ist öffentlich zugänglich und kann von jedem genutzt werden, der an der Forschung oder Entwicklung von Lösungen für das TSP interessiert ist.

[1] `https://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden`
[2] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

2 Projekt

Es soll eine Software entwickelt werden, die das TSP mithilfe verschiedener algorithmischer Ansätze löst. Dazu soll lokale Suche, dynamische Programmierung und ganzzahlige lineare Programmierung genutzt werden. Die entwickelten Programme werden anschließend an Beispielen aus der TSP-LIB getestet, um ihre Effizienz und Genauigkeit zu evaluieren.

2.1 Lokale Suche

Die lokale Suche[3] ist ein heuristischer Ansatz, der darauf abzielt, eine gegebene initiale Lösung schrittweise zu verbessern, indem „benachbarte“ Lösungen untersucht werden. Begonnen wird mit einer zufälligen oder heuristisch generierten Tour (z.B. Nearest Neighbor[4]). Nachbarschaftsoperationen, wie 2-opt oder 3-opt[5], sortieren bestimmte Kanten in der Tour um, damit die Gesamtlänge der Tour reduziert wird.

- 2-opt: Entfernt zwei Kanten und verbindet die Endpunkte der beiden Kanten in umgekehrter Reihenfolge, siehe Abbildung 1 links.
- 3-opt: Entfernt drei Kanten und fügt die Endpunkte in einer neuen Reihenfolge wieder zusammen, siehe Abbildung 1 rechts.

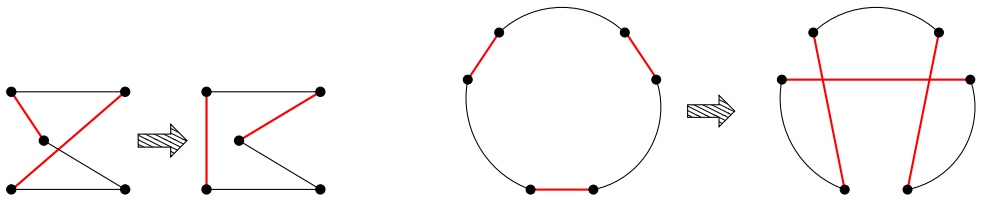

Abbildung 1: k-Opt-Heuristik: links $k = 2$, rechts $k = 3$.

Die Nachbarschaftsoperationen werden wiederholt, solange eine Verbesserung der Tour gefunden wird. Das Verfahren stoppt, wenn keine weiteren Verbesserungen mehr möglich sind oder eine maximale Anzahl Iterationen durchgeführt wurden. Das Ziel der lokalen Suche ist es, eine möglichst kurze Tour zu finden, die jedoch nicht garantiert die optimale Lösung ist.

2.2 Dynamische Programmierung

Algorithmen wie der Held-Karp-Algorithmus[6] berechnen exakte Lösungen für kleine Problemgrößen durch die Zerlegung des Problems in kleinere Teilprobleme. Dazu wird ein Zustand als eine Teilmenge von besuchten Städten sowie der letzten besuchten Stadt definiert. Die rekursive Formel, um die minimalen Kosten für das Besuchen aller Städte zu berechnen, lautet:

$$C(S, j) = \min_{i \in S, i \neq j} (C(S \setminus \{j\}, i) + d(i, j))$$

Hierbei sind $C(S, j)$ die minimalen Kosten, um alle Städte in S zu besuchen und in Stadt j zu enden, und $d(i, j)$ die Distanz zwischen den Städten i und j.

[3] https://de.wikipedia.org/wiki/Lokale_Suche
[4] https://de.wikipedia.org/wiki/Nearest-Neighbor-Heuristik
[5] https://de.wikipedia.org/wiki/K-Opt-Heuristik
[6] https://en.wikipedia.org/wiki/Held-Karp_algorithm

Die obige Formel wird aber nicht top-down rekursiv, sondern bottom-up iterativ gelöst, indem die Ergebnisse der Teilprobleme gespeichert werden, um redundante Berechnungen zu vermeiden. Auf diese Art wird eine optimale Lösung für das TSP gefunden, jedoch mit einer exponentiellen Zeitkomplexität von $\mathcal{O}(n^2 \cdot 2^n)$, dabei ist n die Anzahl der Städte.

2.3 Lineare Programmierung

Das TSP kann als gemischt ganzzahliges lineares Programm[7] formuliert und mittels eines MIP-Solvers gelöst werden. Das Ziel ist es, die Gesamtkosten der Tour zu minimieren:

$$\text{Minimiere} \sum_{i,j} c_{ij} x_{ij}$$

wobei c_{ij} die Distanz zwischen den Städten i und j ist und x_{ij} binäre Variablen sind, die angeben, ob die Kante zwischen i und j in der Tour enthalten ist.

Um sicherzustellen, dass jede Stadt genau einmal besucht wird und dass die Tour eine gültige Rundreise bildet, müssen Nebenbedingungen hinzugefügt werden. Jede Stadt muss genau einmal besucht werden:

$$\sum_j x_{ij} = 1 \quad \forall i \in \text{Städte} \qquad \sum_i x_{ji} = 1 \quad \forall j \in \text{Städte}$$

Um sicherzustellen, dass keine Subtouren entstehen, siehe dazu Abbildung 2, müssen zusätzliche Bedingungen formuliert werden, die die Bildung von Subtouren verhindern.

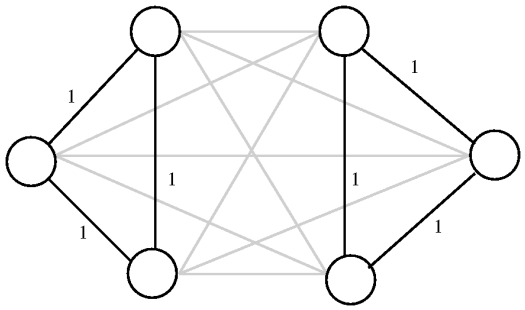

Abbildung 2: Subtouren bei TSP. (Quelle wikipedia)

Eine gängige Methode, Subtouren zu verhindern, ist die Verwendung von Miller-Tucker-Zemlin-Bedingungen[8]. Es werden Variablen u_i hinzugefügt, die die Reihenfolge der Besuche darstellen:

$$u_i - u_j + 1 \leq (n-1) \cdot (1 - x_{ij}) \quad \forall i \neq j$$

Hierbei ist n die Anzahl der Städte.

Es soll ein frei verfügbarer MIP-Solver wie glpsol[9] oder cbc[10] genutzt werden, um das formulierte Problem zu lösen. Die Methode der ganzzahligen linearen Programmierung kann effizienter sein als die dynamische Programmierung, insbesondere bei größeren Instanzen, wenn geeignete Solver verwendet werden.

Dieses Projekt sollte von drei oder vier Studierenden bearbeitet werden.

[7] `https://de.wikipedia.org/wiki/Gemischt-ganzzahlige_Optimierung`

[8] C.E. Miller, A.W. Tucker, R.A. Zemlin: Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM, 7(4), 1960. `https://dl.acm.org/doi/pdf/10.1145/321043.321046`

[9] `https://www.gnu.org/software/glpk/`

[10] `https://www.coin-or.org/Cbc/cbcuserguide.html`