Schatzer, Goller, Unterleitner, Olivotto, Bosin

PROJEKT: PAKETLIEFERSERVICE

Travelling salesman problem (TSP): Dabei hat man eine bestimmte Anzahl von Knoten, welche man jeweils einmal besuchen muss. Der Startpunkt ist gleichzeitig der Endpunkt und wird deshalb zwei Mal besucht. Die gesamte Reisestrecke soll möglichst kurz sein. Um eine kurze Route möglichst schnell zu erhalten, kann man verschiedene Algorithmen benutzen. Erhöht man die Anzahl der Knoten, welche man besuchen will, steigt die Anzahl der möglichen Routen sehr stark. Um die Anzahl der möglichen Routen zu berechnen nutzt man folgende Formel, wobei n für die Anzahl der Städte steht:

$\frac{(n-1)!}{2}$ = mögliche Routen

Bei 15 Städten hat man so 43 Milliarden mögliche Routen und bei 18 Städten ca. 177 Billionen Routen.

Ein **Knoten** kann in unserem Fall wie ein Ort oder ein Punkt angesehen werden. Diese Knoten haben zu jedem anderen Knoten einen Abstand, welchen man aus der Adjazenmatrix auslesen kann.

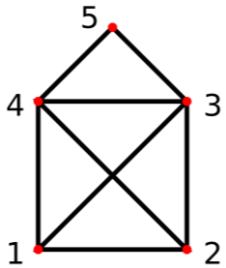
Heuristische Algorithmen sind Algorithmen, die mit begrenztem Wissen und begrenzter Zeit eine passable Lösung finden können.

Brute Force Algorithmen sind Algorithmen, die alle möglichen Lösungen ausprobieren und die beste Lösung benützen.

Adjazenzmatrix ist eine Matrix, die alle Verbindungen von verschiedenen Punkten festhält. Diese Punkte bauen wir in eine CSV-Datei ein.

CSV-Datei ist eine Datei mit Comma-separated-values, die verschiedenen Zahlen mit Komma abtrennt.

Hamiltonkreis ist eine geschlossene Route mit mehreren Knoten. Jeder Knoten wird genau einmal besucht. Ein bekanntes Beispiel dafür ist das Haus vom Nikolaus.



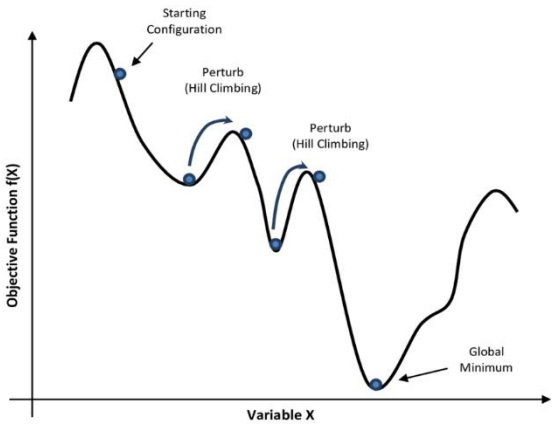
Beispiele Heuristische Algorithmen zur Lösung des Travelling Salesman Problems:

Nearest Neighbor Algorithm: Vom fixen Startpunkt aus wird der am wenigsten entfernte Knoten besucht. Von dem zweiten Punkt wird dann wieder der am geringsten entfernte Knoten besucht, solange bis alle Knoten besucht worden sind. So kommt es zu einem Hamiltonkreis. Dieses Verfahren wählt so meistens eine kurze Tour, jedoch fast nie die Optimale.

Nearest Insertion Algorithm: Ziel ist es wieder eine relativ kurze Rundreise über alle Knoten zu erhalten. Der Algorithmus wählt bei jedem Schritt einen Knoten aus, welcher die geringste Entfernung zu einem Knoten hat, welcher auf der bereits vorhandenen Teilroute vorhanden ist. Dieser Punkt wird dann in die vorhandene Teilroute so eingebaut, dass diese Teilroute sich am geringsten verlängert. Die entstandene Route kann nicht länger sein als die doppelte Strecke der optimalen Route.

Simulated Annealing ("Simuliertes Abkühlen"):

Man startet zuerst mit der Nearest Neighbor Route, dann beginnt man zufällig zwei Punkte in der Route zu tauschen. Dann schaut man, ob sich die Route dadurch verbessert hat oder nicht. Man würde zu einem Punkt kommen, an dem sich die Route nicht mehr verbessern kann, aber noch nicht die beste Route ist. Um das so lange wie möglich zu verhindern, wird die Route auch kurzzeitig verschlechtert, um dann wieder ein besseres Ergebnis zu erreichen. Am Ende kann die Route wieder länger sein, als sie in der Mitte des Programms war, das beste Ergebnis wird aber gespeichert. Das folgende Diagramm sollte den Algorithmus recht gut beschreiben:



Ob die neue Route akzeptiert wird oder nicht, wird wie folgt berechnet:
Wenn die neue Route kürzer als die alte Route ist, dann wird sie klarerweise akzeptiert, ansonsten verwendet man folgende Formel:
Math.exp(-delta-sigma) > Math.random().
Dabei ist -delta die negative Differenz der Länge der neuen Route und der Länge der alten Route und sigma ein Toleranzwert, der sich bei jedem Durchgang verringert.

Quellen:

https://en.wikipedia.org/wiki/Travelling_salesman_problem

https://de.wikipedia.org/wiki/Nearest-Neighbor-Heuristik

https://de.wikipedia.org/wiki/Nearest-Insertion-Heuristik

https://www.sciencedirect.com/topics/engineering/simulated-annealing-algorithm#:~:text=The%20simulated%20annealing%20algorithm%20is,state%20is%20reached%20%5B143%5D.

https://en.wikipedia.org/wiki/Convex_hull_algorithms

https://de.wikipedia.org/wiki/Hamiltonkreisproblem