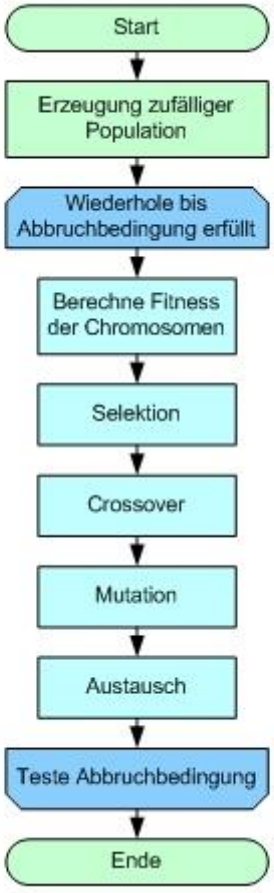
# Lösung des Traveling Salesman Problem mit dem Genetischen Algorithmus

Der in diesem Abschnitt beschriebene Python Code ist im Modul geneticAlgo.py zusammengefasst. Die Implementierung ist keine typische Python-Implementierung. Der rein prozedurale Ansatz begründet sich darin, dass ich den Algorithmus ursprünglich in Matlab geschrieben hatte und das Matlab Script mit relativ wenig Aufwand in ein Python/Numpy Programm übertragen habe. Das Programm implemenitert die Lösung des Travelling Salesman Problem (TSP) mit dem Genetischen Algorithmus (GA). Die einzelnen Schritte des GA sind nicht in allgemeiner Form implementiert, sondern speziell für die Lösung des TSP.

# Das Travelling Salesman Problem

Ein Handlungsreisender muss eine Menge von Städten besuchen und zum Schluss wieder in die Stadt, in der er seine Reise begann, zurück kehren. Es ist davon auszugehen, dass zwischen allen Städten direkte Verbindungen bestehen. Das Handlungsreisende (Traveling Salesman) steht vor dem Problem die Reihenfolge der zu besuchenden Städte derart festzulegen, dass die geamte Distanz seiner Reise minimal ist. Die Lösung des TSP ist NP-complete. Eine suboptimale Lösung kann z.B. mit dem GA berechnet werden.

# Allgemeiner Ablauf



# Die Teilschritte des Genetischen Algorithmus in der Anwendung TSP-Lösung

## Darstellung der Chromosomen

Im Fall des TSP wird ein Chromosom repräsentiert durch eine Rundreise, welche alle Städte passiert und in der Stadt endet in welcher sie begann. Die Population besteht dann aus einer Anzahl (POPSIZE) solcher Rundreisen. Eine Rundreise wird dann als ein Vektor mit 11 Elementen modelliert. Jedes Element ist ein Integer zwischen 1 und 10. Ausser der Zahl, welche die Ausgangsstadt repräsentiert, darf jeder Integer im Vektor nur einmal vorkommen. Das elfte Element ist immer gleich dem ersten Element. Eine mögliche Rundreise wäre also [ 3 6 5 2 8 7 9 4 1 10 3 ].

## Fitness

Für das TSP bietet sich eine Berechnung der Fitness in Abhängigkeit von der Gesamtdistanz einer Rundreise an. Allerdings ist die direkte Benutzung der Distanz nicht angebracht, denn per Definition soll der Fitness Wert proportional zur Eignung (Nutzen), d.h. antiproportional zu den Kosten eines Chromosoms sein. Für die Kosten eines Chromosoms wird die Gesamtdistanz, für die Fitness der inverse Kostenwert (1/Kosten) verwendet.

## Selektion

Die Auswahl eines Chromosomenpaars aus der Population, soll derart implementiert werden, dass die Wahrscheinlichkeit für die Auswahl eines Chromosomes umso größer ist, je höher sein Fitness-Wert ist. Realisiert werden kann dies indem eine in den Grenzen zwischen 0 und Summe über alle Fitness-Werte gleichverteilte Zufallszahl erzeugt wird. Der ganze Wertebereich der Zufallszahl wird in Abschnitte unterteilt, deren Größe genau den Fitnesswerten der einzelnen Chromosomen entspricht. Der Bereich in welchen die gewürfelte Zufallszahl fällt bestimmt das Chromosom das selektiert wird.

# Kreuzung

Für die Kreuzung wird zunächst ein zufälliger Kreuzungspunkt innerhalb des Vektors (eines ausgewählten Chromosomes) gewürfelt. Für das TSP wird folgende Realisierung des Crossover gewählt. Der Anfang (Head) des neuen Vektors ist bis zum Kreuzungspunkt indentisch mit dem Anfang des alten Vektors. Der zweite Teil vom Kreuzungspunkt bis zum Ende des Vektors (Tail), wird wie folgt bestimmt. Scanne vom Anfang bis zum Ende des zweiten ausgewählten Vektors. Sobald dort eine Zahl gefunden wird, die noch nicht im neu zu bildenden Chromosom steht, füge diese Zahl an die nächste freie Stelle im Tail des neu zu bildenden Chromosoms ein. An der letzten Stelle des neuen Chromosomes muss die gleiche Zahl stehen wie an seiner ersten Stelle. Dieser Prozess wird für beide ausgewählte Chromosomen durchgeführt. Die Wahrscheinlichkeit mit der eine Kreuzung in einer Iteration ausgeführt sollte als Parameter einstellbar sein (Typisch: Wahrscheinlichkeitswert nahe bei 1).

Beispiel:

Die Kreuzung der beiden Chromosomen

[ 3 6 5 2 8 7 9 4 1 10 3 ]

[ 7 8 10 2 1 3 6 9 4 5 7 ]

am Kreuzungspunkt cp=6 ergibt die beiden Kinder

[ 3 6 5 2 8 7 9 10 1 4 3 ]

[ 7 8 10 2 1 3 6 5 9 4 7 ]

# Mutation

Mit einer eher kleinen Wahrscheinlichkeit wird in einer Iteration auch eine Mutation der ausgewählten und evtl. durch Kreuzung modifizierten Chromosomen durchgeführt. Im Fall des TSP wird Mutation wie folgt implementiert. Falls eine Mutation stattfindet wird ein zufälliges Paar von Indizees mit Werten zwischen 1 und der Anzahl der Städte erzeugt. Dann werden die Integer-Werte des Vektors (Chromosoms) an diesen beiden Indizees vertauscht. Dabei muss die Eigenschaft, dass Start- und Zielwert identisch sind aufrecht erhalten bleiben.

Beispiel:

Wenn für die Mutation des Chromosoms

[ 7 8 10 2 1 3 6 5 9 4 7 ]

das zufällige Indexpaar gleich (3,5) ist, dann werden der 3. und der 3. Wert im Chromosom vertauscht. Der neue Vektor nach der Mutation ist dann [ 7 8 10 3 1 2 6 5 9 4 7 ]

# Austausch

In jeder Iteration werden in der Population die beiden schlechtesten Chromosomen, d.h. diejenigen mit den geringsten Fitness-Werten, durch die beiden neuen Chromosomen ersetzt, falls letztere bessere Fitness-Werte haben. Dieser Austausch findet jedoch nur dann statt, wenn das neue Chromosom nicht schon in der alten Population vorhanden war. Damit wird gewährleistet, dass in der Population alle Chromosomen verschieden sind.