TSPSolver: Lösungsgenerator für das Traveling Salesman Problem

Teilleistung 3 im Kurs „Soft Computing“, Studiengang VAWi, Universität Duisburg-Essen, Sommersemester 2015

Wolfgang Bongartz

# Zusammenfassung

Das von mir erstellte Programm erzeugt Lösungen für das Traveling Salesman Problem auf Basis eines genetischen Algorithmus. Bei der Implementierung des Algorithmus habe ich mich am Buch „Genetische Algorithmen und Evolutionsstrategien“ (Eberhard Schöneburg et al, Addison-Wesley, 1. Auflage, 1994, ISBN: 3-89319-493-2)[[1]](#footnote-1) orientiert. Der Algorithmus lässt sich durch die Angabe der zu verwendenden Komponenten in gewissen Grenzen konfigurieren. Es ist also prinzipiell möglich, bspw. das Ersetzungsschema, das Heiratsschema oder das Selektionsschema auszutauschen[[2]](#footnote-2).

# Start und Bedienung des Programms

Das Programm wurde in Java erstellt und läuft auf jedem Computer, auf dem eine Java-Laufzeitumgebung (Version 1.7 oder höher) installiert ist. Es wird in einem Terminalfenster durch Eingabe von

java -jar TpsSolver.jar

gestartet. Optional können Name und Pfad eines Textfiles als Parameter angegeben werden:

java -jar TpsSolver.jar tsp\_test.txt

Im Textfile sind dann die zur Steuerung des Programms nötigen Parameter enthalten. Ohne Textfile startet das Programm mit Standardwerten.

# Steuerungsparameter

Im o.g. Textfile werden die Steuerungsparameter im Format

<parameter> = <wert>

angegeben. Nachfolgend sind die möglichen Parameter beschrieben.

| **Parameter** | **Funktion** |
| --- | --- |
| locations2visit | Liste der zu besuchenden Orte (komma-separiert). Möglich sind alle Orte, zu denen das Konfigurationsfile ‚distances.csv’ Entfernungsangaben enthält. |
| numberOfGenerations | Anzahl der Durchläufe („Generation“), in denen nach einer möglichst optimalen Lösung gesucht werden soll. |
| populationSize | Anzahl der Lösungen („Individuen“), die in jedem Durchlauf („Generation“) erzeugt werden sollen. |
| numberOfChildren | Anzahl der neuen Lösungen („Kinder“), die im nächsten Durchlauf aus jeweils einer selektierten Lösung erzeugt werden sollen. |
| mutationRate | Prozentualer Anteil der neuen Lösungen, der nach dem Zufallsprinzip verändert („mutiert“) werden soll.  Angabe in Prozentpunkten (0-100). |
| elitismRate | Prozentualer Anteil der Population aus dem letzten Durchlauf, der in den nächsten Durchlauf übernommen wird.  Angabe in Prozentpunkten (0-100).  Wird nur bei Verwendung des Ersetzungsschemas „elitism“ verwendet. |
| showProgress | Schaltet die „Fortschrittsanzeige“ ein (Wert=’yes’) bzw. aus (jeder andere Wert). |
| showIntermediateResults | Schaltet die Anzeige von Zwischenergebnissen ein (Wert=’yes’) bzw. aus (jeder andere Wert). |
| crossover scheme | Wahl des Rekombinationsschemas. Mögliche Werte[[3]](#footnote-3):  ERX: "edge recombination crossover"[[4]](#footnote-4)  none: Es wird kein Rekombinationsschema angewendet. |
| mutation scheme | Wahl des Mutationsschemas. Mögliche Werte:  random exchange: Austausch zweier zufällig gewählter Gene.  none: Es wird kein Mutationsschema angewendet. |
| substitution scheme | Wahl des Ersetzungsschemas. Mögliche Werte:  elitism: Ein prozentualer Anteil der besten Lösungen der Population werden in die nächste Generation übernommen.  generational replacement: Die gesamte Population wird ersetzt. |
| evaluation scheme | Wahl des Evaluationsschemas. Also der Methode, nach der die Bewertung der Lösungen vorgenommen wird. Einzig möglicher Wert: round trip (also die Berechnung der Gesamtstrecke für eine Rundreise)[[5]](#footnote-5). |
| selection scheme | Wahl des Heiratsschemas. Einzig möglicher Wert: relative fitness[[6]](#footnote-6). |

# Beispiele

Dem Programm liegen beispielhaft drei Konfigurationsdateien bei, die hier kurz erläutert werden sollen.

## tsp\_test.txt

Tour aus 15 Städten. Das Optimum scheint bei einer Gesamtlänge von 3260 Kilometern zu liegen. Bei der im File definierten Konfiguration wird dieser Wert häufig schon nach 10 bis 15 Generationen erreicht und konnte bisher auch nach Versuchen mit mehreren Tausend Generationen nicht verbessert werden[[7]](#footnote-7).

Aufruf: java -jar TpsSolver.jar tsp\_test.txt

## tsp\_schoeneburgLike.txt

Aufruf: java -jar TpsSolver.jar tsp\_schoeneburgLike.txt

Berechnung einer Rundreise, die der in [GA] auf Seite 278 vorgestellten Reise recht ähnlich ist[[8]](#footnote-8). Sie enthält insgesamt 25 Städte, die jedoch teilweise weiter voneinander entfernt sind als die in [GA] verwendeten.

## tsp\_max.txt

Aufruf: java -jar TpsSolver.jar tsp\_max.txt

Berechnung einer Rundreise, die alle Orte enthält, für die Entfernungsangaben vorhanden sind (insgesamt 30).

# Die Programmstruktur

## Das Hauptprogramm

Das Hauptprogramm besteht aus der Klasse MainProgram. Diese Klasse stellt die main-Methode bereit, kümmert sich um das Einlesen und Interpretieren des Konfigurationsfiles, erzeugt nach den Vorgaben im Konfigurationsfile die Komponenten für den Genetischen Algorithmus und startet den Optimierungsvorgang schließlich (durch Aufruf der in der Klasse GeneticAlgorithm definierten run-Methode).

## Das Package ‚distances’

Hier ist im Wesentlichen die Klasse Distances enthalten, die eine in einer CSV-Datei definierte Entfernungstabelle einliest und danach Auskunft über die Entfernung zwischen zwei Orten geben kann.

## Das Package ‚genetic\_algorithm’

Die Klasse GeneticAlgorithm enthält die übergreifende Logik des Optimierungsvorgangs, wie er in [GA] grundsätzlich für einen genetischen Algorithmus vorgeschlagen wird. Die Klasse verwaltet die Instanzen aller nötigen Komponenten und sorgt für deren sinnvolles Zusammenspiel. Die run-Methode kapselt den eigentlichen Optimierungsprozess.

Die weiteren Klassen in diesem Package sind:

* Generation  
  Kapselt eine Generation („Population“) von Lösungen. Also die Menge aller Lösungen, die innerhalb eines Optimierungsdurchlaufs bewertet und aus der neue Lösungen für den nächsten Durchlauf erzeugt werden.
* Individual  
  Kapselt genau eine Lösung („Chromosom“), die sich bewerten und für die sich ein Fitness-Wert berechnen lässt.
* IndividualComparator  
  Erlaubt den Vergleich zweier Lösungen anhand des Fitness-Wertes. Wird programmintern benötigt, um Lösungen effizient verwalten und vergleichen zu können.
* IndividualIterator  
  Wird programmintern benötigt, um die Klasse Individual in For-Each-Schleifen verwenden zu können.
* Pair  
  Erlaubt die paarweise Anordnung von Lösungen im Heiratsschema.

## Das Package ‚genetic\_algorithm.component’

Dieses Package enthält Interfaces für alle Komponenten einen genetischen Algorithmus, für die verschiedene Verfahren existieren und die sich daher sinnvollerweise austauschen lassen sollten. Außerdem sind einige Verfahren in entsprechenden Klassen implementiert.

Die Interfaces im Einzelnen:

* Crossover\_Algorithm  
  Schnittstelle für Rekombinations-Schemata.
* Evaluation\_Algorithm  
  Schnittstelle für die Bewertungsfunktion.
* Mutation\_Algorithm  
  Schnittstelle für Mutations-Schemata.
* Selection\_Algorithm  
  Schnittstelle für Heirats-Schemata.
* Substitution\_Algorithm  
  Schnittstelle für Ersetzungs-Schemata.

Die folgenden Klassen enthalten die konkreten Implementierungen:

* Crossover\_ERX  
  Implementiert das in [GA] beschriebene Rekombinationsschema "edge recombination crossover".
* Crossover\_NoCrossover  
  Kann verwendet werden, wenn keine Rekombination gewünscht ist. Überführt die Eltern einfach in die nächste Generation.
* Crossover\_PMX  
  Implementiert das in [GA] beschriebene Rekombinationsschema "partially matched crossover". Hinweis: Derzeit ist die Implementierung leider fehlerhaft. Daher kann dieses Schema nicht verwendet werden.
* Evaluation\_RoundTrip  
  Bewertungsfunktion für Touren. Berechnet mithilfe der in der Klasse Distances enthaltenen Entfernungstabelle die Gesamtstrecke einer Lösung.
* Mutation\_NoMutation  
  Kann verwendet werden, wenn keine Mutation gewünscht ist.
* Mutation\_RandomExchange  
  Vertauscht mit einer definierten Wahrscheinlichkeit zwei Orte miteinander.
* Selection\_RelativeFitness  
  Berechnet einen Fitnesswert, der sich reziprok zur Bewertung (also der Streckenlänge) verhält. Die beste Lösung der Generation hat den Fitnesswert 1. Die schlechteste Lösung den Fitnesswert 0.  
  Berechnungsvorschrift:

1. Bestimme die kleinste Routenlänge innerhalb der Generation (all\_best)
2. Bestimme die größte Routenlänge innerhalb der Generation (all\_worst)
3. Bestimme die Differenz zwischen der längsten und der kürzesten Route innerhalb der Generation (range)
4. Bestimme für jede Lösung die Differenz zwischen der Routenlänge (my\_distance) und der längsten Route (all\_worst)
5. Berechne den Fitnesswert des Lösung:  
   fitness = ( all\_worst – my\_distance ) / range

* Substitution\_Elitism  
  Implementierung des in [GA] auf Seite 206 beschriebenen Ersetzungsschemas „Elitismus“.
* Substitution\_GenerationalReplacement  
  Ersetzt eine Generation vollständig durch ihre Nachkommen.

# Dateien und Ordner

Im Hauptverzeichnis der Teilleistung befinden sich folgende Dateien und Ordner:

* TSPSolver.jar (Datei)  
  Das ausführbare Programm.
* Dokumentation.docx (Datei)  
  Die vorliegende Dokumentation.
* Die oben beschriebenen Beispieldateien
* distances.csv (Datei)  
  Enthält die Entfernungstabelle. Sie wird beim Programmstart von der Klasse Distances eingelesen.
* doc (Verzeichnis)  
  Enhält eine mit JavaDoc aus den Quelltextkommentaren erzeugte Dokumentation der Programmartefakte (Startpunkt: index.html).
* src (Verzeichnis)  
  Enthält den Sourcecode.

1. Im Folgenden mit [GA] referenziert. [↑](#footnote-ref-1)
2. Allerdings wurden nur für das Rekombinationsschema und das Ersetzungsschema jeweils Varianten implementiert. Mit dem Programm ließen sich durch Implementieren passender Komponenten prinzipiell aber auch andere Optimierungsprobleme als das TSP lösen, ohne die Programmstruktur ändern zu müssen. [↑](#footnote-ref-2)
3. Das in [GA] ab Seite 273 beschriebene „partially matched crossover“ ist zwar implementiert, doch diese Implementierung funktioniert leider noch nicht. Sie kann deshalb nicht ausgewählt werden. [↑](#footnote-ref-3)
4. Siehe [GA], S. 276 ff. [↑](#footnote-ref-4)
5. Diese Konfigurationsmöglichkeit besteht nur, damit ich das Programm später ggf. erweitern kann. Sie hat derzeit noch keine Bedeutung. [↑](#footnote-ref-5)
6. Auch diese Konfigurationsmöglichkeit dient nur der späteren Erweiterbarkeit. [↑](#footnote-ref-6)
7. Das könnte natürlich auch auf einen bisher nicht enddeckten Programmfehler zurückzuführen sein. [↑](#footnote-ref-7)
8. Leider konnte ich die im Buch verwendete Tour nicht exakt nachbilden, weil der von mir verwendeten Entfernungstabelle die Orte Freiburg, Karlsruhe, Wuppertal, Essen, Bielefeld, Kiel, Magdeburg und Braunschweig fehlen. Die Entfernungstabelle habe ich aus einer Tabelle erzeugt, die Angaben zu 30 Orten enthält. Um die fehlenden vier Orte zu ergänzen hätte ich also insgesamt 240 Entfernungsangaben recherchieren müssen. [↑](#footnote-ref-8)