Technology
Arts Sciences
TH Köln

Parallelisierter Genetischer Algorithmus für das TSP

Timo Bertsch
Joram Markert
Fabian Meyer

Master Computer Science

Gummersbach, 30.08.2016

Abstract

Thema:	Parallelisierter Genetischer Algorithmus für das TSP
Author:	Timo Bertsch Joram Markert Fabian Meyer
Prüfer:	Prof. Dr. Lutz Köhler Pascal
Datum:	30.08.2016

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

1 Einleitung

einleitungs blabla

2 Das Travelling Salesman Problem

Was ist das problem / Ziel? Auf kombinatorische explosion eingehen; Quellen suchen und erklörung darauf stützen

3 Der Genetische Algorithmus

Im Rahmen dieses Projektes wurde ein genetischer Algorithmus entwickelt, mit dem das TSP gelöst wird. Genetische Algorithmen werden im Allgemeinen zur Lösungsfindung bei komplexen Suchproblemen verwendet, bei denen eine kombinatorische Explosion vorliegt und der Lösungsraum so groß ist, dass eine Lösung mit einer optimalen Brute Force Methode nicht möglich ist.

Die allgemeine Funktionsweise dieses Algorithmus und die Varianten, die in diesem Projekt eingesetzt wurden, werden in Abschnitt 3.1 erläutert. Der Basisalgorithmus ist jedoch nicht nebenläufig konzipiert, daher mussten einige Modifikationen vorgenommen werden, um diesen in einem verteilten System einsetzen zu können. Diese Änderungen werden in Abschnitt 3.2 näher beschrieben.

3.1 Funktionsweise

Genetische Algorithmen basieren auf der Idee der natürlichen Auslese und ermitteln durch Selektion, Kombination und Evolution von möglichen Lösungen eine sehr gute bis perfekte Lösung für das gesuchte Problem.
Eine Lösung des gegebenen Problems wird als *Individuum* bezeichnet. Im Rahmen des TSP sind Individuen Wege, die die Lösungskriterien des TSP erfüllen (siehe Kapitel 2). Die Reihenfolge der Knoten, die bei einem Individuum besucht werden, werden als dessen Eigenschaften oder Gene bezeichnet. Eine Menge von Individuen, auf der der Algorithmus operiert, wird *Population* genannt. Der genetische Algorithmus arbeitet iterativ in sogenannten *Generationen*. Die Individuen einer Population werden schrittweise in sogenannten *Generationen* mit einer *Fitness* bewertet, aussortiert und rekombiniert, um die Qualität der Lösungen zu verbessern. Die Phasen einer Generation werden in Tabelle 3.1 zusammenfassend dargestellt (vgl. [1]).

Phase	Beschreibung
Initialisation	Generiert die initiale Population und wird nur ein einziges Mal zu Beginn des Algorithmus ausgeführt. Die Population wird mit zufällig generierten Individuen erstellt.
Evaluation	Die Individuen der aktuellen Generation werden mit einem Fitnesswert bewertet. Dieser Zahlenwert drückt aus wie gut das Individuum bzw. die Lösung des des Problems ist. In Bezug auf das TSP bedeutet eine kürzere Strecke der Lösung eine höhere Fitness des entsprechenden Individuums.
Selection	Aus den Individuen der aktuellen Generation werden diejenigen ausgewählt, die für die Crossover Phase verwendet und die in die nächste Generation übernommen werden. Hierbei ist es von zentraler Bedeutung, dass Individuen mit hoher Fitness mit einer höheren Wahrscheinlichkeit ausgewählt werden als die Individuen mit niedriger Fitness.
Crossover	Aus den ausgewählten Individuen (sog. Parents) werden duch die Kombination von deren Eigenschaften neue Individuen (sog. Children) erzeugt, die in die nächste Generation übernommen werden. In Bezug auf das TSP werden hier Teilwege der Parents so miteinander, dass die Children wieder eine valide Lösung des TSP darstellen.
Mutation	Die Gene der erzeugten Children werden per Zufall verändert, um eine Stagnation des Genpools zu verhindern und neue Lösungsmöglichkeiten zu erzeugen. Diese Stagnation kann auftreten, wenn über mehrere Generationen hinweg besonders fitte Individuen die Population dominieren.

Tabelle 3.1: Phasen des genetischen Algorithmus

Die Umsetzung dieser Phasen im Rahmen dieses Projekts wird in den folgenden Unterabschnitten anhand des Beispielgraphen in Abbildung 3.1 detailliert beschrieben.

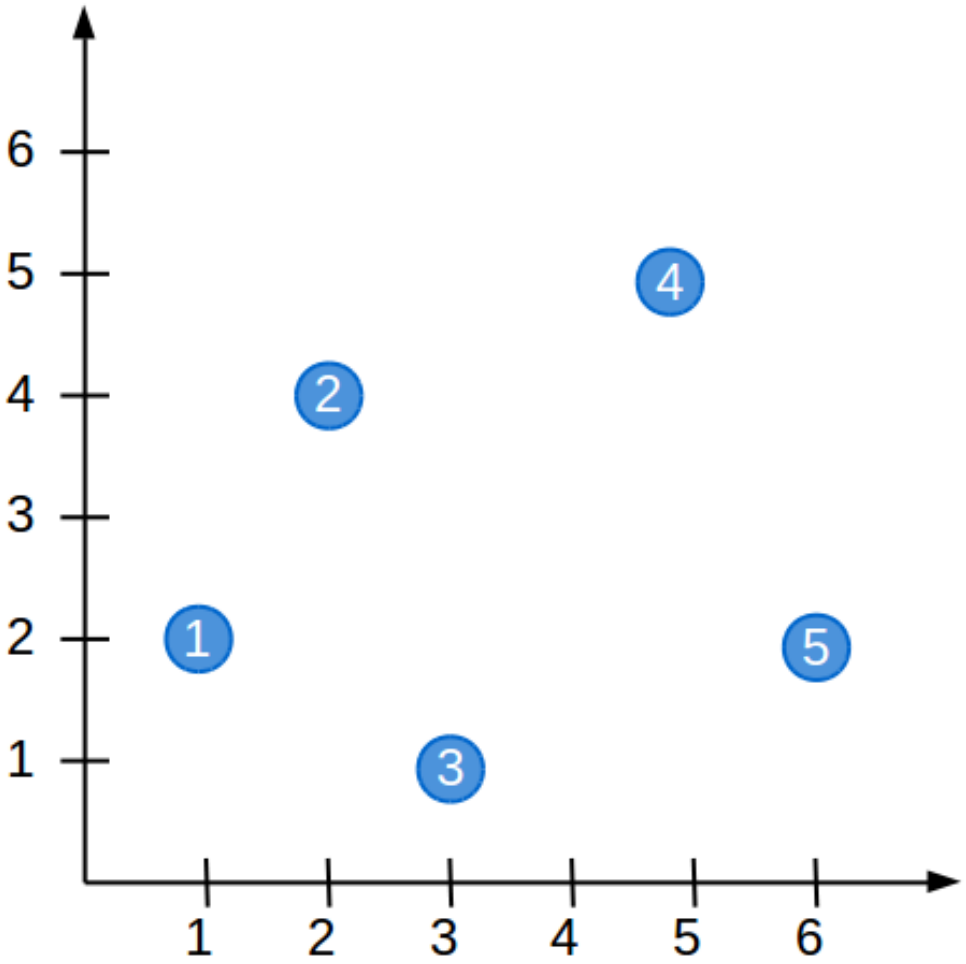

Abbildung 3.1: Beispielgraph

Dieses Beispiel zeigt einen Graphen mit fünf Knoten, die jeweils durch die Nummer in ihrer Mitte identifiziert werden können.

3.1.1 Initialisation

Die *Initialisation* Phase wird nur ein Mal zu Beginn des Algorithmus ausgeführt. In dieser Phase wird die initiale Population generiert, mit der der Algorithmus dann arbeitet. Die Individuen werden hierbei vollständig zufallsgeneriert, wobei jedes Individuum eine valide Lösung des TSP darstellt. Dieser Vorgang wird in Listing 3.1 dargestellt.
Um ein Individuum zu generieren, wird eine Liste (*nodes*) aller möglichen Knoten für das Individuum, also Knoten aus dem Graphen, erstellt (l. 1). Aus dieser Liste wird dann ein Knoten zufällig ausgewählt (l. 5) und dem Individuum hinzugefügt (l. 6). Danach wird der Knoten aus der Liste der übrigen Knoten entfernt

```
1 List nodes = //...
2 List indivduum
3
4 while(not nodes.empty())
5     int index = randomInt()
6     individuum.add(nodes[index])
7     nodes.remove(index)
```

Listing 3.1: Generierung eines Individuums

(l. 7) und dieser Prozess wird solange wiederholt bis keine Knoten mehr in *nodes* vorhanden sind (l. 4).

Mit diesem Verfahren können die Individuen in Abbildung 3.2 aus dem Beispielgraphen in Abbildung 3.1 generiert werden.

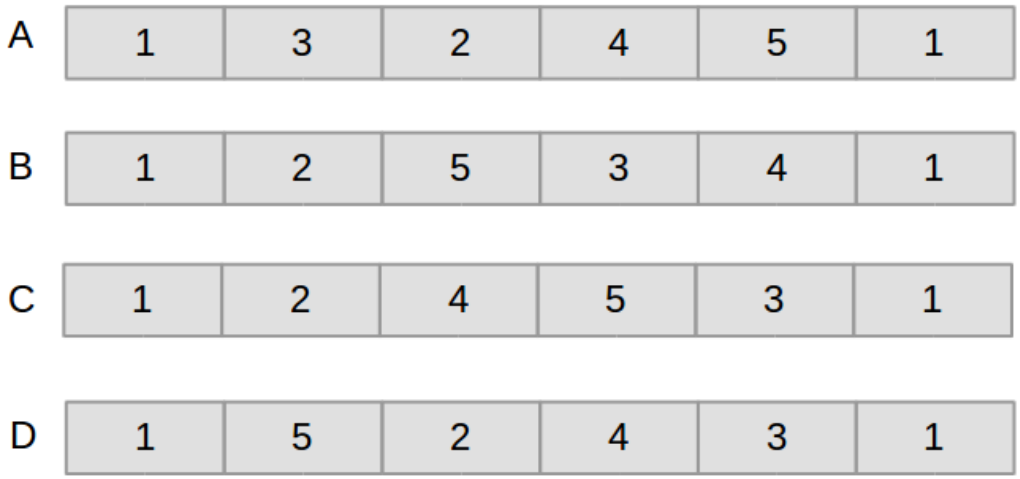

Abbildung 3.2: Beispiel: Zufällig generierte Individuen

3.1.2 Evaluation

In der *Evaluation* Phase werden die Individuen der Population der aktuellen Generation bewertet und damit ihr Fitness Wert errechnet. Jedes Individuum stellt einen Weg durch den Graphen, also eine Liste aufeinander folgender Knoten, dar. Die Fitness errechnet sich aus der Länge dieses Weges. Diese Länge wird durch die Summe des euklidischen Abstands zwischen in der Liste aufeinanderfolgende Knoten beschrieben, wie in der folgenden Gleichung dargestellt wird.

$$l_{individuum} = \sum_{i=1}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

Dies wird für alle Individuen wiederholt. Hierbei gilt, dass umso länger der Weg ist umso niedriger ist der Fitnesswert eines Individuums. Damit verhält sich die Fitness also *antiproportional* zur Länge des Weges.
Aufgrund von Rundungsfehlern bei Gleitkommazahlen reicht ein einfacher Kehrwert der Länge des Weges nicht aus um die Fitness eines Individuums zu berechnen, da der Weg je nach Größe des Graphen entsprechend lang werden kann. Daher wird aus allen Individuen der aktuellen Population dasjenige gesucht, das den längsten Weg l_{max} besitzt. Mit diesem Wert wird nun die Fitness $f_{individuum}$ jedes Individuums auf folgende Weise berechnet.

$$f_{individuum} = \frac{l_{max}}{l_{individuum}}$$

Tabelle 3.2 zeigt die Weglänge und die Fitness der Individuen aus Abbildung 3.2.

Individuum	$l_{individuum}$	$f_{individuum}$
A	$1 + \sqrt{10} + \sqrt{10} + \sqrt{10} + 5 = 15.49$	$\frac{19.34}{15.49} = 1.25$
B	$\sqrt{5} + \sqrt{20} + \sqrt{10} + \sqrt{20} + 5 = 19.34$	$\frac{19.34}{19.34} = 1$
C	$\sqrt{5} + \sqrt{10} + \sqrt{10} + \sqrt{10} + 1 = 12.72$	$\frac{19.34}{12.72} = 1.52$
D	$5 + \sqrt{20} + \sqrt{10} + \sqrt{20} + 1 = 18.11$	$\frac{19.34}{18.11} = 1.07$

Tabelle 3.2: Beispiel Fitness und Distanz

3.1.3 Selection

Während der *Selection* Phase werden die Individuen (Parents) ausgewählt, aus denen die Individuen für die nächste Generation (Children) generiert werden. Die Auswahl geschieht anhand der Fitness der Individuen.Wichtig ist hierbei auch, dass ein einzelnes Individuum mehrfach ausgewählt werden kann.
In diesem Projekt wurde zur Auswahl der Parents das *Roulette-Wheel-Verfahren* verwendet. Hierbei werden die Individuen durch eine Zufallszahl ausgewählt, doch haben besonders fitte Individuen eine besonders hohe Chance ausgewählt zu werden. Hierbei wird die Fitness der Individuen auf das Intervall [0;1] normiert. Daraufhin wird eine Auswahlintervall für jedes Individuum berechnet, indem die Liste der Individuen angefangen beim fittesten Individuum durchlaufen wird und die Fitness aller vorangegangenen Individuen aufsummiert wird. Der sich daraus ergebende Wert ist die untere Grenze g_{xunten} des Auswahlintervalls des Indivi-

duums x. Die obere Grenze g_{xoben} wird durch die Summe aus unterer Grenze und der Fitness des aktuellen Individuums gebildet. Die folgenden Gleichungen beschreiben den Rechenweg nochmals.

$$g_{xunten} = \sum_{i=1}^{i<x} f_i$$

$$g_{xoben} = g_{xunten} + f_x$$

Abbildung 3.3 zeigt die Auswahlintervalle der Individuen aus Abbildung 3.2.

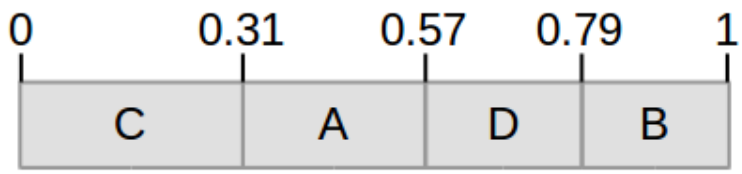

Abbildung 3.3: Beispiel: Auswahlintervalle der Individuen

3.1.4 Crossover

3.1.5 Mutation

3.2 Parallelisierung

prinzipielle parallelisierung; was ist die idee?; ohne auf MPI einzugehen!

4 Technologie

Programmiersprache; json; build system; unit test; github; travis CI; gnuplot etc.

4.1 MPI

wie funktioniert mpi prinzipiell?; p2p verbindung; ssh; broadcast, send, receive

5 Testumgebung

MAC pool; programmierung auf linux;

6 Ergebnisse

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Lee Jacobsen. Creating a genetic algorithm for beginners. http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3, 2012. Accessed: 20.05.2016.