

# Lösung des Traveling-Salesman-Problem mithilfe einer parallelisierten Anwendung der Optimierung durch den Ameisen-Algorithmus

Studienarbeit

des Studienganges Angewandte Informatik an der  
Dualen Hochschule Baden-Württemberg Mosbach



von  
Viktor Rechel

Bearbeitungszeitraum	2 Semester; 6 Monate
Matrikelnummer, Kurs	6335802, Inf15A
Hochschule	DHBW Mosbach
Gutachter der Dualen Hochschule	Dr. Carsten Müller

7. November 2017

# Abstract

# Zusammenfassung

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>vi</b>
<b>Abbildungsverzeichnis</b>	<b>vii</b>
<b>Tabellenverzeichnis</b>	<b>viii</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Stand der Technik &amp; Forschung</b>	<b>2</b>
2.1 Traveling Salesman Problem . . . . .	2
2.2 Ant Colony Optimization . . . . .	2
<b>3 Konzeptionierung</b>	<b>3</b>
3.1 Architektur . . . . .	3
3.1.1 Persistenz . . . . .	3
3.1.2 Applikation . . . . .	4
3.1.3 TSP . . . . .	4
3.1.4 ACO . . . . .	4
3.2 UML-Diagramm . . . . .	4
3.3 Ausgewählte Algorithmen . . . . .	5
3.3.1 Ant - iteration() . . . . .	5
3.3.2 Colony - killAnt() . . . . .	5
3.3.3 Colony - updatePheromone() . . . . .	6
3.4 Parameteranalyse . . . . .	6
3.5 Umsetzung der SOLID-Prinzipien . . . . .	6
3.5.1 Single Responsibility . . . . .	6
3.5.2 Open / Closed . . . . .	6
3.5.3 Liskov Substitution . . . . .	6
3.5.4 Interface Segregation . . . . .	6
3.5.5 Dependency Inversion . . . . .	6
3.6 Sensitivitätsanalyse . . . . .	6
<b>4 Implementierung</b>	<b>7</b>
4.1 Klassendiagramm . . . . .	7
4.2 Beschreibung der Implementierung . . . . .	7
4.3 ER-Diagramm . . . . .	7
4.4 Komponenten-Diagramm . . . . .	7

---

4.5	Paket-Diagramm . . . . .	7
4.6	Perforamce-Analyse und -Optimierung . . . . .	7
<b>5</b>	<b>Fazit</b>	<b>9</b>
	<b>Literaturverzeichnis</b>	<b>10</b>

# Abkürzungsverzeichnis

**ACO** Ant Colony Optimization oder Ameisenalgorithmus

**TSP** Traveling Salesman Problem

**UML** Unified Modeling Language

**ER** Entity Relationship

# Abbildungsverzeichnis

3.1	UML-Modellierung des Architekturentwurfs . . . . .	5
4.1	Modellierung der Software-Architektur als Paketdiagramm. Zu sehen ist der logische Aufbau der Software, sowie der Zusammenhang der einzelnen Pakete. . . . .	8

# Tabellenverzeichnis



# 1 Einleitung

## 2 Stand der Technik & Forschung

### 2.1 Traveling Salesman Problem

### 2.2 Ant Colony Optimization

## 3 Konzeptionierung

Um eine präzise und effiziente Umsetzung dieser Arbeit zu gewährleisten, wird zu Beginn der Fokus auf eine durchgängig durchdachte Konzeptionierung gelegt. Durch diese wird es im Anschluss deutlich eleganter möglich sein, die Architektur umzusetzen. Im Folgenden wird die Planung einzeln analysiert und dargestellt.

### 3.1 Architektur

Die Architektur wurde inhaltlich so aufgeteilt, dass eine modulare Implementierung möglich ist. So wurden die vier folgenden Bereiche definiert: Persistenz, Applikation, TSP und ACO

Der Begriff der Persistenz beschreibt in diesem Fall zusätzlich zum dauerhaften Abspeichern der Daten, auch das Einlesen der verschiedenen Problemstellungen.

Applikation umfasst den Teil des Programms, der sich nicht direkt mit Daten befasst aber auch nicht zur Problemlösung beiträgt, wie die zwei folgenden Bereiche.

Innerhalb des Begriffs TSP werden alle nötigen Parameter und Methoden behandelt, die basierend auf dem Traveling Salesman Problem notwendig werden.

Zuletzt gibt es in der Architektur noch das Feld ACO, welches die komplette Berechnung des zu lösenden Problems übernimmt. In dem Fall, dieser Arbeit handelt es sich um das TSP. Allerdings könnte das Problemfeld auch dadurch ausgewechselt werden, dass der Bereich TSP um das neue Problem ersetzt wird.

#### 3.1.1 Persistenz

Um ein stures Definieren des Problems innerhalb der Applikation zu verhindern, werden zu Beginn des Programms alle Parameter, wie Städtematrix, Wahrscheinlichkeiten und Lösungsparameter aus einer gegebenen XML-Datei eingelesen. Durch eine Bearbeitung der XML ist ein einfaches Abändern der Problemstellung möglich. Hierdurch ist auch gesichert, dass die Algorithmen effizient getestet werden können, da mehrere verschiedenen bekannte Testwerte benannt werden können.

### 3.1.2 Applikation

Wie bereits genannt, liegt bei der vorliegenden Architektur ein Fokus auf Modularität, Portabilität und Usability. Aber auch auf verlässliche und effiziente Algorithmen muss geachtet werden. Daher wird für die Wahrscheinlichkeitsberechnung die externe Klasse *MersenneTwisterFast*<sup>1</sup> genutzt, welche eine bessere Distribution der Pseudozufallszahlen bieten als die Default-Implementierung in Java.<sup>2</sup> Ein weiterer Bestandteil des Applikationsbereichs werden das Logging der Arbeitsvorgänge, sowie die zentrale Konfiguration der Problemstellung auf der die Lösung basiert.

### 3.1.3 TSP

Der Bereich des TSP definiert sich in dieser Architektur rein durch die Städte-Objekte, welche zur Darstellung der Städtematrix genutzt werden. Der einzige andere Bestandteil ist die zentrale Aufstellung der Städtematrix, die von den restlichen Klassen nur kopiert wird.

### 3.1.4 ACO

Um den Ameisenalgorithmus umzusetzen sind deutlich mehr Aufwände nötig, als zur Darstellung des TSP. Hier werden mindestens eine Ameisenkolonie benötigt<sup>3</sup>, sowie je Kolonie mehrere Ameisen.

Die Ameisen werden in der Applikation als einzelne Threads gestartet, die über eine *CyclicBarrier* kontrolliert werden. Dies erlaubt ein möglichst effizientes Parallelisieren der Lösung.

## 3.2 UML-Diagramm

Um an der Planung auch visuell arbeiten zu können, wurde diese zu Beginn als UML-Diagramm erarbeitet. Wie in Abbildung 3.1 zu sehen ist, reichen für die Implementierung eine wenige Klassen aus. So wird der Teil des Programms, welcher das TSP-Problem darstellt nur durch die Klassen *Landscape* und *City* vertreten. *Landscape* beinhaltet die Matrix der Städte inklusive der Distanzen zwischen den Städten.

---

<sup>1</sup>s. <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/emt.html>

<sup>2</sup>Um eine einfache und schnelle Benutzung zu gewährleisten, wird in dieser Arbeit darauf verzichtet echte Zufallszahlen zu nutzen, die beispielsweise aus Weltallstrahlung berechnet werden. Diese seien hier nur zur Vollständigkeit halber erwähnt.

<sup>3</sup>Die Architektur würde bei ausreichenden Systemressourcen auch eine parallele Berechnung mehrerer Städtematrizen erlauben

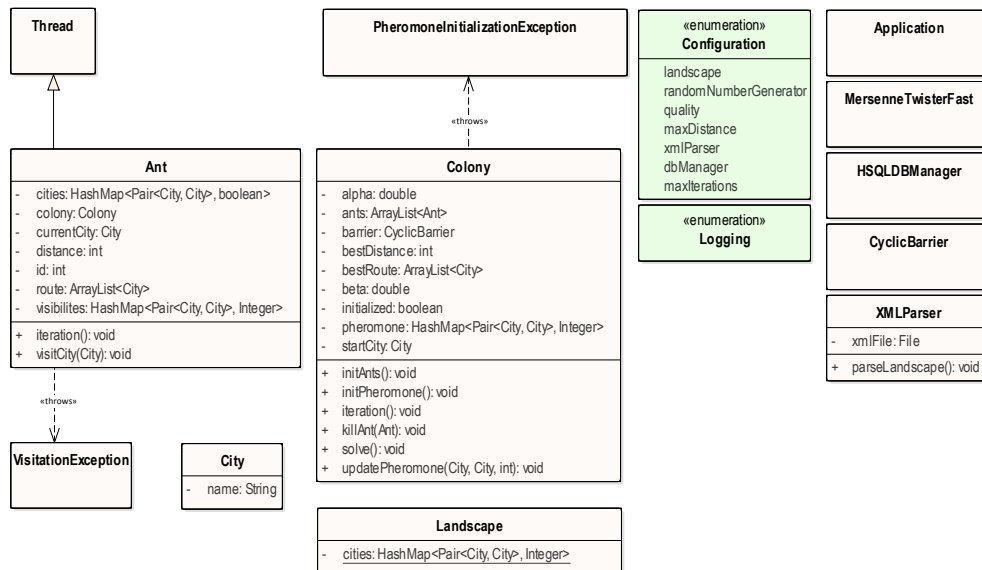


Abbildung 3.1: UML-Modellierung des Architekturentwurfs

## 3.3 Ausgewählte Algorithmen

### 3.3.1 Ant - iteration()

### 3.3.2 Colony - killAnt()

Ebenfalls in dem in Abbildung 3.1 gezeigten UML-Diagramm erkennbar ist, dass die Ameisenkolonie die Möglichkeit besitzt einzelne Ameisen zu "töten". Diese Methode sollte in einem einwandfreiem Programm keinerlei Verwendung finden, allerdings kann man sich nicht auf eine dauerhafte fehlerfreie Implementierung verlassen. Im Bereich der Software Tests ist dieses Verhalten durch das Pestizid Paradox bestätigt.<sup>4</sup> Denn die Methode hat die Funktion, im Falle des Fehlverhaltens eine Ameise aus der Liste der aktiven Ameisen bzw. Threads zu löschen und den Thread zu stoppen. Genutzt werden wird diese vor allem im Bereich der aufgefangenen Fehler innerhalb der Implementierung der Ameisen. Sollte eine aufgetretene Exception schwerwiegend und unlösbar sein, wird die Applikation automatisch die Funktion aufrufen.

<sup>4</sup>vgl. [bibid]

### 3.3.3 Colony - updatePheromone()

## 3.4 Parameteranalyse

## 3.5 Umsetzung der SOLID-Prinzipien

### 3.5.1 Single Responsibility

### 3.5.2 Open / Closed

### 3.5.3 Liskov Substitution

### 3.5.4 Interface Segregation

### 3.5.5 Dependency Inversion

## 3.6 Sensitivitätsanalyse

## 4 Implementierung

### 4.1 Klassendiagramm

### 4.2 Beschreibung der Implementierung

### 4.3 ER-Diagramm

### 4.4 Komponenten-Diagramm

### 4.5 Paket-Diagramm

### 4.6 Performamce-Analyse und -Optimierung

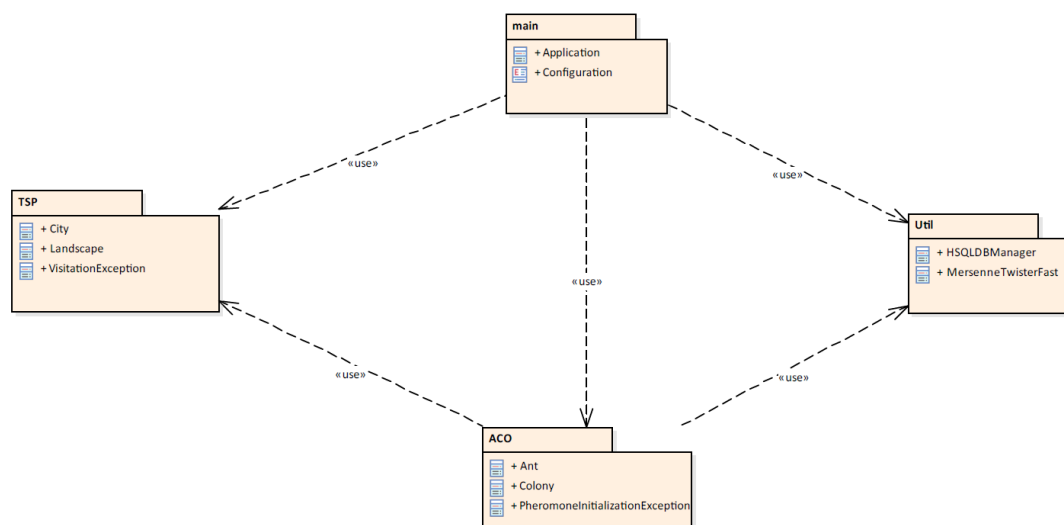


Abbildung 4.1: Modellierung der Software-Architektur als Paketdiagramm. Zu sehen ist der logische Aufbau der Software, sowie der Zusammenhang der einzelnen Pakete.



# 5 Fazit

Ich bin ein Fazit

# Literaturverzeichnis

- [1] B. Booba; Dr. T. V. Gopal. „Comparison of Ant Colony Optimization & Particle Swarm Optimization In Grid Scheduling“. English. In: *Australian Journal of Basic and Applied Sciences* (8. Juni 2014), S. 1–6.
- [2] David P. Williamson. *The Traveling Salesman Problem: An Overview*. English. Presentation. Cornell University Ebay Research, 21. Jan. 2014.