|  |  |
| --- | --- |
| Lösung des TSP Problems mittels Ameisensystemen  Dokumentation | |
| **Autoren:** | Alexander Salzer;Felix Wiederschein;Philipp Thomas;Tobias Schlufter;Christian Mrosk |
| **Modul:** | IT4111 – Fachübergreifendes Labor |
| **Fachbereich:** | Technik Fachbereich 2, Duales Studium Wirtschaft • Technik, HWR Berlin |
| **Studiengang:** | Angewandte Informatik |
| **Studienjahrgang:** | 2010 |
| **Datum:** |  |

# Abstrakt

**Im Rahmen des Moduls „Fachübergreifendes Labor (IT4111)“ wurde an den Informatik-Kurs des 3. Semesters der HWR-Berlin die Aufgabe gestellt, das Travelling-Salesman-Problem mithilfe von Ameisensystemen unter Anwendung von XP-Techniken zu lösen.**

**In der folgenden Dokumentation wird zunächst eine Einführung in das Travelling-Salesman-Problem, Ameisensysteme sowie die agile Softwareentwicklung gegeben.**

**Weiterhin wird auf die durchgeführten Arbeiten zur Lösung der Problemstellung sowie die verwendeten XP Techniken detaillierter eingegangen.**

**Abschließend zieht die Gruppe ein Fazit über den Verlauf des Projektes und gibt einen Ausblick in Bezug auf eventuelle Verbesserungsmöglichkeiten des Programmes.**

# Inhaltsverzeichnis

[I. Abstrakt 1](#_Toc307505567)

[II. Inhaltsverzeichnis 2](#_Toc307505568)

[III. Abkürzungsverzeichnis 4](#_Toc307505569)

[IV. Glossar 5](#_Toc307505570)

[V. Abbildungsverzeichnis 6](#_Toc307505571)

[VI. Tabellenverzeichnis 6](#_Toc307505572)

[VII. Hauptteil 7](#_Toc307505573)

[VII.1. Einleitung 7](#_Toc307505574)

[VII.2. Theoretische Hintergrundkenntnisse 7](#_Toc307505575)

[VII.2..1. Ameisensysteme 7](#_Toc307505576)

[VII.2..2. Travelling Salesman Problem 7](#_Toc307505577)

[VII.3. Unser Projekt 7](#_Toc307505578)

[VII.3..1. XP-Techniken 7](#_Toc307505579)

[VII.4. Implementierung 7](#_Toc307505580)

[VII.5. ANTS-TSP 7](#_Toc307505581)

[VII.6. Fazit 7](#_Toc307505582)

[VII.7. Ausblick 7](#_Toc307505583)

[2. Theoretische Hintergrundkenntnisse 8](#_Toc307505584)

[1. Ameisensysteme 8](#_Toc307505585)

[2. Travelling Salesman Problem 8](#_Toc307505586)

[VIII. Anhang 10](#_Toc307505587)

[IX. Quellenverzeichnis 11](#_Toc307505588)

[IX.1. Internetquellen 11](#_Toc307505589)

[IX.2. Abbildungsquellen 12](#_Toc307505590)

[X. Ehrenwörtliche Erklärung 13](#_Toc307505591)

# Abkürzungsverzeichnis

GUI = Graphical User Interface

TSP = Travelling Salesman Problem

# Glossar

Abstrakt = Klassen die nur Methodensignaturen deklarieren

Application = Anwendungssoftware

Array = Datenfeld

Deployment = Inbetriebnahme neuer Versionen webbasierter Anwendungen

Entität = ein eindeutig bestimmtes Objekt

Framework = Programmiergerüst

Hibernate = Open-Source-Persistenz-ORM-Framework für Java

ORM = Konzept zum Abbilden von Objekten in relationale Datenbanken

SQL = am häufigsten eingesetzte Datenbanksprache

Sorted List = Liste, die einen Schlüsselwert und den eigentlichen Wert enthält. Durch den Schlüssel ist der direkte Aufruf eines Listenelements möglich, ohne die ganze Liste durchsuche zu müssen.

Tool = Anwendungsprogramm, welches andere Programme analysiert, erstellt, modifiziert oder manipuliert

# Abbildungsverzeichnis

[Abb. 1 Java Logo **Fehler! Textmarke nicht definiert.**](#_Toc301539158)

[Abb. 2 Hibernate Logo **Fehler! Textmarke nicht definiert.**](#_Toc301539159)

# Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Hauptteil

## Einleitung

## Theoretische Hintergrundkenntnisse

## Ameisensysteme

## Travelling Salesman Problem

## Unser Projekt

## XP-Techniken

## Implementierung

**Aufbau der GUI**

<noch ein Bild der GUI und der Map einfügen>

Im Eingabeteil der GUI können die Parameter, die den Algorithmus beeinflussen, eingestellt werden. Über eine Schaltfläche kann der Algorithmus gestartet und wieder beendet werden. Im Ausgabeteil der GUI werden beim Durchlaufen des Algorithmus globale und iterative Durchschnitts- und Bestwerte angezeigt. Um den Fortschritt sehen zu können, wird angezeigt, wie lange der Algorithmus bereits läuft, welche Ameise gerade durch ihren Weg geht und im der welche Iteration gerade durchlaufen wird.

In einem Menüpunkt kann ein leeres TSP erstellt werden. Der Benutzer sieht dann ein leeres Fenster, in das der Benutzer per Mausklick eigene Punkte setzen kann.

Ein anderer Menüpunkt lässt eine TSP-Datei laden. Die Koordinaten, die in der TSP-Datei hinterlegt sind, werden in ein Array aus Punkten gespeichert und in ein Fenster gezeichnet. Zu den nun bereits vorhanden Punkten lassen sich per Mausklick wieder eigene, neue Punkte hinzufügen. Das Fenster lässt sich in der Größe beliebig verändern.

Nachdem der Algorithmus durchgelaufen ist, können die erreichten Werte in einer XML-Datei abgespeichert werden. Diese Datei enthält die Bestwerte, die Parameter, die Dauer und den optimalen Weg.

**Der Algorithmus**

Hauptbestandteil der Implementierung war die Umsetzung des Algorithmus, der das Ameisenverhalten simuliert. Das erfolgte in Pair-Programming-Sitzungen. Einer der Partner war von Anfang an mit der Erstellung des Programms beschäftigt. Der zweite Partner hat sich vorher mit der Theorie des Ameisenalgorithmus auseinander gesetzt und dabei bereits Ideen gesammelt, wie sich der Algorithmus am besten implementieren lässt.

Die Städte wurden aus der gegebenen TSP-Datei in eine Liste gelesen. Jede Stadt hat als Attribute ihren X- und Y-Wert im Koordinatensystem, sowie eine weitere Liste. In dieser Liste sind alle nachfolgenden Städte, die dazugehörigen Entfernungen sowie die Attraktivität und den Pheromongehalt des Weges angegeben. Es entsteht die Struktur einer oberen Dreiecksmatrix. Dadurch ist sichergestellt, dass die Informationen über die Wege zwischen zwei Städten nur einmal vorhanden sind. Zum besseren Aufruf eines Listenelements werden *Sorted Lists* verwendet. Das Durchsuchen und Aktualisieren der Listen ist dadurch einfacher. Bevor auf ein Listenelement zugergriffen wird, prüft eine Methode die richtige Reihenfolge der beiden Schlüsselwerte. So wird ein Zugriff auf die untere, leere Hälfte der Matrix ausgeschlossen.

Jede Ameise besitzt die Attribute *firstCity*, *City* und *walkedDistance*. Darin werden die Ausgangsstadt, die aktuelle Stadt und die bereits zurückgelegte Distanz gespeichert. Des Weiteren besitzt jede Ameise eine Liste mit allen Städten, die sie noch nicht besucht hat. Die Ameisen sind in einer Liste arrangiert, um sie nacheinander abarbeiten zu können.

Nach dem Start des Algorithmus werden die Städteliste und die Ameisenliste erstellt. Für den Pheromongehalt auf den Wegen zwischen den Städten wird der Startwert genommen. Jede Ameise wird zufällig in einer Stadt abgesetzt, ihrer ersten Stadt. Diese Stadt wird aus der Liste der Ameise gelöscht.

Nun beginnt der eigentliche Algorithmus. Die erste Ameise in der Liste berechnet, solange sie noch nicht alle Städte besucht hat, in welche Stadt sie als nächstes geht. Zur Berechnung wird die oben beschriebene Formel verwendet. Es wird die Wahrscheinlichkeit zu jeder noch in der Liste vorhanden Stadt berechnet, die Ergebnisse werden gespeichert. Die Ameise geht anschließend in die Stadt, die das größte Ergebnis besitzt. Die Stadt wird wieder aus der Liste gelöscht.

Das Attribut *walkedDistance* wird aktualisiert, indem zu dem aktuellen Weg die soeben zurückgelegte Wegstrecke dazu addiert wird. Zusätzlich werden in ein Array aus Punkten (mit X- und Y-Werten) die Koordinaten der neuen Stadt geschrieben. Dieses Array wird benutzt, um später den Weg der Ameise zu zeichnen.

Ist die Liste mit den noch nicht besuchten Städten leer, geht die Ameise zurück zu ihrer Ausgangstadt. Auf dem Ausgabefenster wird der Weg der Ameise gezeichnet. Ist der aktuelle Weg kürzer als der vorher kürzeste Weg, wird er im Ausgabefenster hervorgehoben gezeichnet, ansonsten im Hintergrund in grau. Im Anschluss wird der Pheromongehalt jedes Weges mit der oben beschriebenen Formel aktualisiert. Dieser Vorgang wiederholt sich für alle Ameisen. Sind alle Ameisen durchgelaufen, ist eine Iteration beendet. Vor der nächsten Iteration werden die Listen mit den nicht besuchten Städten der Ameisen wieder aufgefüllt. Nun kann die nächste Iteration beginnen.

## ANTS-TSP

## Fazit

## Ausblick

1. ****Einleitung****

## Theoretische Hintergrundkenntnisse

## Ameisensysteme

## Travelling Salesman Problem

Das Travelling Salesman Problem (TSP) beschäftigt sich mit dem Problem des Handelsreisenden. Dieser möchte eine vorgegebene Menge von Städten nacheinander besuchen und am Ende wieder zum Ausgangsort zurückkehren. Die Reihenfolge, in der er die Städte besucht, kann er dabei selbst festlegen. Diese bestimmte Art von Reise wird auch als Tour durch alle Städte bezeichnet. Der Handelsreisende hat dabei einen Plan zur Verfügung, der ihm alle Entfernungen (z.B. in Kilometer) zwischen jeweils zwei Städten anzeigt. Jede einzelne Tour liefert eine Gesamtlänge, die der Handelsreisende für diese Tour zurücklegen muss. Er kann dafür zwei Ziele verfolgen. Das höchste Ziel wäre, eine Tour mit minimaler Gesamtlänge zu finden. Falls ihm dieses Ziel jedoch zu ehrgeizig ist, kann er auch nach einer Tour mit möglichst kleiner Gesamtlänge suchen.

Das Handelsreisendenproblem ist eines der bekanntesten Probleme der theoretischen und praktischen Informatik. Einige Gründe dafür wären, dass es einfach zu verstehen ist, aber sich zugleich auch schwer implementieren und lösen lässt. Außerdem kann man an diesem Problem praktisch testen, was Computerprogramme für bestimmte Beispiele leisten können. Somit stellt sich die Frage, in welcher Laufzeit finden sie Touren mit welcher Qualität?

Im Alltag findet das TSP sehr häufig Anwendung. Zum Beispiel werden Netzfahrpläne nach dem TSP strukturiert, da alle Verkehrsmittel vorgegebene Haltestellen anfahren. Auch in der Lagerlogistik nimmt das TSP eine wichtige Bedeutung ein. Die Einzelposten einer Bestellung werden im Lager zusammengesucht. Weiterhin wird das TSP auch für die Tourenplanung von Einsatzfahrzeugen, wie zum Beispiel der Deutschen Post, DHL oder UPS, verwendet. Die einzelnen Fahrzeuge fahren ihre verschiedenen Depots nach einer optimierten Route ab. Die Optimierung der Routen ist notwendig, um unnötige Wege, verschwendete Zeit und damit auch eventuell entstehende Zusatzkosten zu vermeiden.

Im Folgenden werden jeweils das Finden einer minimalen Tour, sowie das Finden einer möglichst guten Tour am Beispiel des TSP für die 52 Orte in Berlin dargestellt.

Finden einer minimalen Tour:

Es werden alle möglichen Touren durchprobiert, wobei die kürzeste Tour dann ausgewählt wird. In der Theorie ist dies die beste Methode. Sie führt jedoch dazu, dass sehr viele Touren durchprobiert werden müssen. Folglich wird die Laufzeit, durch die hohe Komplexität, erhöht.

* Beispiel:
  + Ameisen TSP 52 Orte in Berlin

Am Startort hat jede Ameise 51 Möglichkeiten zu einem Ort zu wandern. Am ersten Durchgangsort sind es dann nur noch 50 Möglichkeiten. Dies wird bis zum Schluss durchgeführt. Somit hat er am 51-ten Durchgangsort nur noch eine Möglichkeit um zum letzten Ort zu wandern.

* + Insgesamt:

51 × 50 × 49 × 48 × … × 4 × 3 × 2 × 1 = 51!

= 1.55111875 × 1066

Somit sind 1.55111875 × 1066 Touren bei 52 Orten möglich. Da für beliebige Anzahl n von Orten = (n-1)! Touren durchzuprobieren sind.

Die Komplexität ist also:

O(n) = (n-1)!

Finden einer möglichst guten Tour:

Bei dieser Methode geht man in jedem Schritt zum nächstgelegenen Ort. Wenn dann alle Orte besucht wurden, geht man wieder zum Startort. Vorteil dieser Methode ist, dass am Anfang sehr kurze Entfernungen zurückgelegt werden. Daraus resultiert jedoch auch, dass am Ende die Auswahl der noch nicht besuchten Orte sehr gering sein kann. Somit ist es möglich, dass die Entfernungen, die noch zurückgelegt werden müssen, sehr groß sein können. Für diese Methode wird aber eine Liste benötigt, die jeder Ort mitführt, in der die Entfernungen zu den anderen Orten vermerkt sind.

* Beispiel:
  + Ameisen TSP 52 Orte in Berlin

Am Startort hat jede Ameise für jede der 52 Orte die 51 Entfernungen zu den anderen Orten zu vergleichen. Die minimale Entfernung wird notiert.

* + Insgesamt:

51 × 52 = 2.652 Schritte

Somit sind 2.652 Schritte für eine beliebige Anzahl n von Orten durchzuführen.

Die Komplexität beträgt also:

O(n) = n × (n-1) = n²-n

# Anhang

# Quellenverzeichnis

## Internetquellen

## Abbildungsquellen

1. <http://www.diso.ch/fileadmin/user_upload/rechte_Spalte/java-logo.gif>
2. <http://blog.aplikacja.info/wp-content/uploads/2009/09/hibernate_logo_a.png>

# Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich:

1. dass ich meinen Praxistransferbericht ohne fremde Hilfe

angefertigt habe,

1. dass ich die Übernahme wörtlicher Zitate aus der Literatur

sowie die

Verwendung der Gedanken anderer Autoren an den

entsprechenden

Stellen innerhalb der Arbeit gekennzeichnet habe,

1. dass ich meinen Praxistransferbericht bei keiner anderen

Prüfung vorgelegt habe,

1. dass ich Quellen, die dem World Wide Web entnommen oder

in einer sonstigen digitalen Form verwendet wurden, der Arbeit beigefügt sind.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

04.03.2011

