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:	30.10.2011

I. 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.

II. Inhaltsverzeichnis

III. Abkürzungsverzeichnis

ACO	= Ant Colony Optimization
GUI	= Graphical User Interface
LE	= Längeneinheiten
TSP	= Travelling Salesman Problem

IV. Glossar

Array	= Datenfeld
Repository	= Speicherort für die Programmdateien
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.

V. Abbildungsverzeichnis

VI. Hauptteil

VI.1. Einleitung

Das Travelling-Salesman-Problem („Problem des Handlungsreisenden“) ist ein kombinatorisches Optimierungsproblem aus dem Bereich der theoretischen Informatik.
Erstmals im Kontext der Mathematik wurde das Problem 1930 von Karl Menger erwähnt, welcher es noch als „Botenproblem“ bezeichnete.

Da das Problem des Handlungsreisenden leicht zu verstehen ist, eignet es sich gut dafür praktisch, in einem Programm, umgesetzt zu werden.

Dabei gibt es verschiedene Möglichkeiten das Traveling-Salesman-Problem zu lösen. Zum einen existieren sogenannte „exakte“ Lösungsverfahren, zum anderen „heuristische“Lösungsverfahren.
Der Unterschied zwischen beiden Möglichkeiten besteht darin, dass exakte Lösungsverfahren die beste Lösung finden, dafür allerdings beliebig lange benötigen können. Heuristische Lösungsverfahren dagegen finden relativ schnell eine Lösung, welche allerdings nicht die beste sein muss.

Die Lösung mittels Ameisensysteme gehört dabei zu den heuristischen Lösungsverfahren, genauer gesagt zu den metaheuristischen Verfahren. Diese Lösung hat einen natürlichen Prozess als Vorbild, das Verhalten von realen Ameisen bei der Futtersuche.

VI.2. Theoretische Hintergrundkenntnisse

VI.2..1. Ameisensysteme

Das Ameisensystem ist ein Beispiel aus der Natur, welches in der Informatik ihre Anwendung findet. Um dies exakt handhaben zu können, benötigt man Kenntnisse über das Verhalten von Ameisen in der Natur. Die Ameise ist ein Tier, welches in einem Kollektiv lebt und arbeitet. Aus diesem Grund treten sie bei der Futtersuche in einem Schwarm auf. Dabei geschieht die Kommunikation, über Pheromon Spuren, welche die erste Ameise hinterlassen hat. Somit wird den darauffolgenden der Weg erklärt, wobei aber nicht alle Ameisen diesen Weg gehen, sondern finden andere Wege, die kürzer sind, um zum Nest zu gelangen. Die folgende Abbildung erklärt dies bildlich.

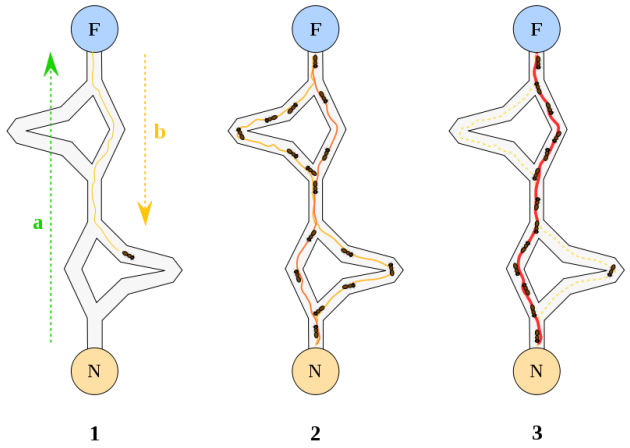

Abb. 1 Ameisen Futtersuche

Dieses Optimierungsverfahren bei der Futtersuche nennt man Ant Colony Optimization. Dabei ist aber zu beachten das sie nicht immer den kürzesten Weg laufen, sowie die Tatsache, dass sich mit der Zeit ein Weg mit den meisten Pheromon Spuren durchsetzt und die Ameisen nicht von diesem abbringen lassen. Vorteile bei diesem Optimierungsverfahren sind die Einfachheit der Individuen, welche aber durch ihre Schwarmintelligenz glänzen und der Simulation am Computer. Trotz aller Naturverbundenheit bei diesem System muss es

verschiedene Unterschiede zwischen den künstlichen und den realen Ameisen geben. Zum einen wird der Algorithmus mit diskreten Zeitschritten versehen, man kann vermeiden, dass sie in einer Schleife hängen bleiben und die künstlichen Ameisen besitzen ein Gedächtnis für den Pfad und die Länge. Des Weiteren werden die Pheromon Spuren verdampft um die Leistung des Algorithmus zu verbessern. Doch wie sieht der Algorithmus aus? Hierfür stehen die folgenden Bilder.

Definition 1 (Algorithmus Pseudocode)

- Initialisierung
- so lange Abbruchbedingung nicht erreich ist
  - für alle Ameisen
    - bestimme mögliche nächste Knoten
    - für alle möglichen Zielknoten
      - bestimme Wahrscheinlichkeit anhand der Pheromonspuren zu den Knoten
    - wähle durch eine Zufallszahl probalistisch einen Knoten
    - bewege Ameise
    - aktualisiere Pheromonspur auf den Kanten

Abb. 2 Ameisensystem Algorithmus Pseudocode

Definition 2 **(Knotenbestimmung AS)**

Für jede Ameise k wird in jedem Iterationsschritt h am Knoten i die Wahrscheinlichkeit für die Auswahl der folgenden Knoten j bestimmt nach:

$$p_{ij,h}^{k} = \frac{[\tau_{ij,h}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}\left([\tau_{il,h}]^{\alpha}[\eta_{il}]^{\beta}\right)}$$

Dabei sind:

- $j \in N_i^k$ die Knoten j ,welche erlaubte Nachfolger des Knotens i für Ameise k sind,
- $\tau_{ij,h}$, der aktuelle Pheromonwert der Kante $\langle i,j\rangle$ bei Iteration h,
- η_{ij} eine heuristische (vorausschauende) Information zur Bewertung der Attraktivität den Knoten j als Nächsten anzusteuern,
- $\alpha, \beta \geq 0$ Steuerparameter für das Verhalten (s. Abs. 2.2.4)

Definition 3 **(Heuristische Information AS)**

Die heuristische Informationsgröße η_{ij} ist die Inverse der Kosten/Länge der betrachten Kante $\langle i,j\rangle$:

$$\eta_{ij} = \frac{1}{c_{ij}}$$

Definition 4 **(Pheromoninitialisierung AS)**

Zur Initialisierung der Pheromonspur wird oft die *„Nächster-Nachbar-Heuristik"(NN)* verwendet. Alle Kanten $\langle i,j\rangle$ des Netzes werden dabei wie folgt gleich markiert:

$$\tau_{ij,1} = \frac{m}{C^{nn}},$$

wobei m die Anzahl der Ameisen und C^{nn} die gesamten Kosten der *Nächster-Nachbar*-Lösung T^{nn} sind.

Definition 5 **(Pheromonaktualisierung AS)**

Wenn m Ameisen bis zu Iteration h die Lösungen $T_h^1, T_h^2, \ldots T_h^m$ mit den Gesamtkosten /-längen $C_h^1, C_h^2, \ldots C_h^m$ erzeugt haben, wird die Pheromonspur $\tau_{ij,h}$ auf jeder Kante $\langle i,j\rangle$ aktualisiert nach folgender Regel:

$$\tau_{ij,h+1} = \tau_{ij,h}(1-\rho) + \sum_{k=1}^{m}\Delta_{ij,h}^{k}$$

mit:

$$\Delta_{ij,h}^{k} = \begin{cases} 1/C_h^k & \langle i,j\rangle \in T_h^k \\ 0 & \text{sonst} \end{cases}$$

$$0 \leq \rho \leq 1$$

Abb. 3 Ameisensystem Algorithmus Definition

Daraus ergibt sich nun die Möglichkeit die verschiedenen Parametereinstellungen einzustellen, um das Verhalten des Algorithmus zu verbessern. Beispielsweise kann man die Anzahl der Ameisen, sowie der Iterationen verändern. Damit wird die Tiefe der Erkundung im Netzwerk beschrieben. Weiterhin kann die Zeit eingestellt werden, nach der Pheromon Spuren versiegen. Die Exponenten α & β

betonen Unterschiede zwischen den Kanten bei der Bestimmung der Wahrscheinlichkeiten

- $\alpha > \beta$
    - höhere Bedeutung der durch Iterationen erzeugten Lösungen und dynamischen Bewertungen der Kanten
- $\alpha < \beta$:
    - statische heuristische Information erhält über die Kantenlängen höheren Einfluss

Bei der Verwendung dieses Optimierungsverfahrens sind viele Vorteile vorhanden. Man besitzt eine einfache Struktur, sowie keine Beschreibung von angepassten Vererbungsvorgängen. Außerdem kann man es vielfältig anwenden bei gleich bleibender Charakteristika, jedoch muss man beachten, dass es sehr schnell zu einer Konvergenz in einem lokalen Optima kommt. Trotz alledem findet es in verschiedenen Bereichen ihre Anwendung, wie der Routenplanung und der Zuordnung von Objekten zu einem Satz von Ressourcen.

VI.2..2. 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?

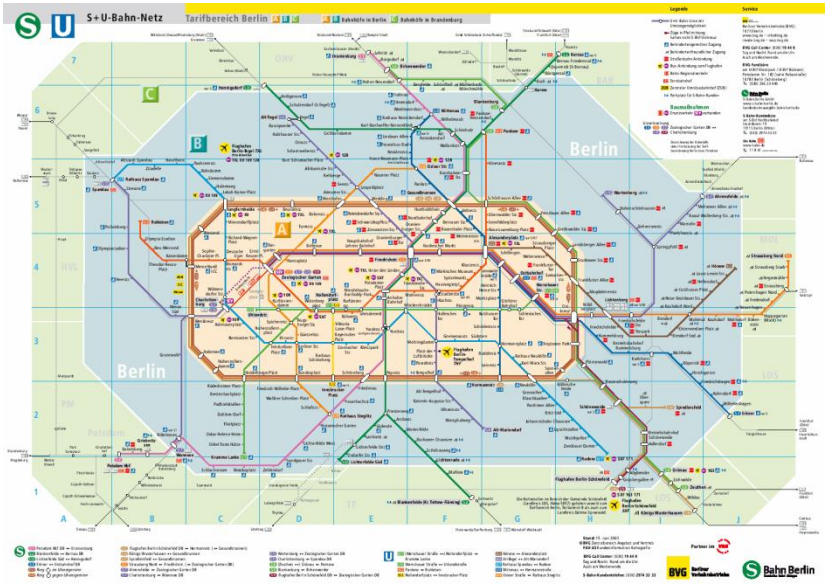

Abb. 4 S- und U-Bahn Plan Berlin

Im Alltag findet das TSP sehr häufig Anwendung. Zum Beispiel werden Netzfahrpläne (vgl. Abb. 4) 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:

$$51 \times 50 \times 49 \times 48 \times \ldots \times 4 \times 3 \times 2 \times 1 = 51!$$
$$= 1.55111875 \times 10^{66}$$

Somit sind $1.55111875 \times 10^{66}$ 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:

$$51 \times 52 = 2.652 \text{ Schritte}$$

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

Die Komplexität beträgt also:

$$O(n) = n \times (n-1) = n^2-n$$

VI.3. Unser Projekt

VI.3..1. XP-Techniken

Zur Realisierung des Projektes war dem Team eine Zeit von acht Wochen gegeben. Diese wurden in zwei *Iterationen* mit einem Zwischenrelease und einem Endrelease. Um mit dem Prinzip *Collective-Code-Ownership* zu arbeiten wurde am Anfang der ersten Iteration ein Repository aufgesetzt. Nachdem nach dem ersten von zwei *Planning-Games* die Anforderungen an das Programm für das erste Release klar waren, wurden die ersten *Story-Cards* erstellt. Da das Team keine Möglichkeit hatte in einem konstanten, lokalen Umfeld zu arbeiten, war es nicht möglich ein *Story-Board* zu erstellen um die Cards zu verwalten. Eine digitale Lösung musste helfen. Mittels eines Tabellenkalkulationsprogramms wurden also die Story-Cards gehalten und über das Repository und aktualisiert. Daraufhin ging es dann in die Entwicklungsphase. Hier wurden zuerst *Coding Standards* festgelegt, die es strikt einzuhalten galt. Codiert wurde zum großen Teil mit der Technik *Pairprogramming*. Nach der Fertiggestellung einer Funktion, wurde diese sofort getestet (Unit-Test) und mit Refactoring möglichst leicht verständlich strukturiert. Ein großer Vorteil für den Entwicklungsprozess war, dass der Kunde immer für Fragen bereitstand (*On Site-Customer*). So konnten Unklarheiten sofort geklärt werden.

Immer wenn das Team zusammengetroffen ist, wurde ein *Stand-Up-Meeting* vollzogen um einen besseren Gesamtüberblick über den Stand des Projektes zu gewährleisten.

VI.4. Implementierung

Aufbau der GUI

Abb. 5 Eingabefenster

Im Eingabeteil der GUI (siehe Abb. 5) 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.

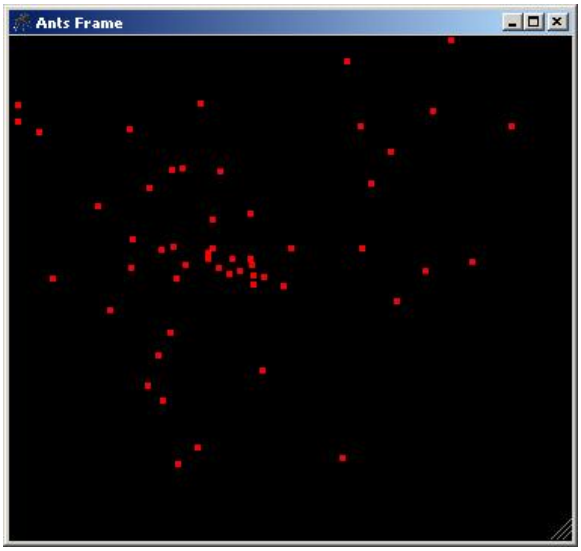

Abb. 6 Zeichenfenster mit Berlin52 TSP

In einem Menüpunkt kann ein leeres TSP erstellt werden. Der Benutzer sieht dann ein leeres Fenster, in das per Mausklick eigene Punkte gesetzt werden können.
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 (siehe Abb. 6). 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 Pheromon Gehalt 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 zugegriffen 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*, c*ity* 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. Diese Stadt wird 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 (siehe Abb. 3). Im Anschluss wird der Pheromon Gehalt 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.

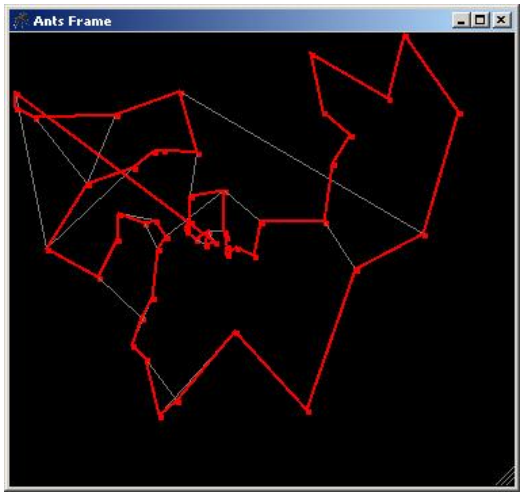

Abb. 7 Zeichenfenster nach dem Durchlaufen des Algorithmus

Der Algorithmus endet, wenn eines der drei Abbruchkriterien erreicht ist:

- Vorgegebene Anzahl von Iterationen durchlaufen.
- Vorgegebenen Schwellwert für eine Tour erreicht.
- Bestwert für das Problem erreicht.

Die Werte für die Abbruchkriterien werden im Eingabefenster festgelegt.

VI.5. Testfälle

Nach der Implementierung des Algorithmus konnten Tests durchgeführt werden. Dabei wurde erkannt, dass die Ergebnisse besser werden, wenn wenige Ameisen und viele Iterationen benutzt werden.
Zu Beginn der Tests wurden alle Parameter auf 1 gesetzt. Am Anfang des Algorithmus wurden gute Werte gefunden. Da aufgrund des hohen Wertes für den Verdunstungsparameter bereits nach dem ersten Pheromonupdate fast kein Pheromon mehr auf den Wegen lag, gingen die Ameisen zum Schluss lange Wege und konnten keine bessere Lösung mehr finden.
Der Wert für die Pheromonverdunstung wurde verringert. Daraufhin pendelten sich die Touren nach ein paar Iterationen bei einem relativ guten Wert ein, der globale Bestwert konnte aber nicht noch einmal unterboten werden. Dieser wurde immer noch direkt am Anfang erreicht. Um dieses Problem zu beheben, wurde der heuristische Parameter verringert. Dadurch werden die Wege, die viele Ameisen gehen stärker mit Pheromon belegt.
Die Durchschnittswerte verringerten sich wieder ein bisschen, der Bestwert wurde aber immer noch direkt zu Beginn erstellt. Nun wurde Alpha verringert und Beta vergrößert. Nun wurde der Bestwert immer mal wieder unterboten. Der Algorithmus wurde also mit jeder Iteration besser, das Ziel für die optimalen Parameter war erreicht. Zum Abschluss wurde noch mit großen Alpha und kleinem Beta getestet. Die erreichten Werte waren jedoch sehr schlecht.

Damit sind folgende optimale Parameter festgestellt worden:

- Alpha = 0,2
- Beta = 8
- Rho = 0,3
- $Tau_0 = 1$
- Q = 0,01

VI.6. Fazit

Das Fazit der Gruppe zum Projekt „Lösung des TSP Problems mittels Ameisensystemen“ fällt insgesamt positiv aus.

Das Ziel des Projektes bestand darin, die im Teilmodul „Projektmanagement“ kennengelernten modernen Projektmanagement- und Programmiertechniken anzuwenden.

Dies war in vielen Fällen, wie beispielsweise dem Pair Programming oder dem Collective Code Ownership sehr gut möglich.

Für einige Techniken war das Projekt der Meinung der Gruppe nach allerdings eher weniger geeignet (Story Cards), da Aufgrund der Anzahl der Aufgaben und der Gruppengröße der direkte Kommunikationsweg der schnellere und einfacher zu organisierende war.

Als weiterer positiver Punkt wird angesehen, dass ein Problem der theoretischen Informatik, welches bereits in früheren Semestern erläutert wurde praktisch gelöst wurde.

VI.7. Ausblick

Während den Arbeiten an der zweiten und somit aktuellen Programmversion von ANTS-TSP wurden an verschiedenen Stellen des Programms Verbesserungspotenziale entdeckt.

Diese sind sowohl in der GUI als auch in der Logik bzw. den Funktionen zu finden.

So würde es die Benutzerfreundlichkeit steigern, wenn man den Fenstern die Möglichkeit des „andockens“ bietet.

Im Bezug auf die Logik bieten sich Verbesserungsmöglichkeiten dahingehend, dass die Performance, also die Abarbeitungszeit des Algorithmus deutlich verbessert werden kann.

Eine funktionale Erweiterung die sich anbietet, wäre dem Benutzer die Möglichkeit zu geben gespeicherte Dateien erneut zu laden, um so genutzte Eigenschaftswerte und durchlaufene Touren erneut darzustellen.

VII. Anhang

1. Ausdruck der Story Cards

VIII. Quellenverzeichnis

VIII.1. Internetquellen

http://www.informatik.uni-kiel.de/~gej/publ/halle2.pdf

VIII.2.Abbildungsquellen

1. http://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Aco_branches.svg/400px-Aco_branches.svg.png
2. http://www.schillrich.de/dateien/SA_OR_AntSystems.pdf
3. http://www.schillrich.de/dateien/SA_OR_AntSystems.pdf
4. http://www.sterne-hotels-berlin.de/img/s_bahn_plan_l.gif
5. *eigene Abbildung*
6. *eigene Abbildung*
7. *eigene Abbildung*