Praktikum Algorithmische Anwendungen
WS 2006/2007

Traveling Salesman Problem

Team A_gelb_Ala0607

Roman Pyro 11042289 ai821@gm.fh-koeln.de
Markus A. Müller 11043150 ai820@gm.fh-koeln.de
René Hilger 11043210 hoshy76@gmx.de

25.01.2007

Inhaltsverzeichnis

Traveling Salesman Problem (TSP) ... 3
- Asymetrisches TSP ... 3
- Symetrisches TSP ... 3
- Metrisches TSP ... 3

Lösungsverfahren ... 4
- Heuristiken ... 4

Ameisenalgorithmus ... 4
- Vorbild ... 4
- Adaption ... 5
- Algorithmus ... 6
  - Auswahl des nächsten Knoten ... 6
  - Aktualisieren der Pheromone ... 8
  - Laufzeit und Güte ... 8

Komplexitätsklasse NP ... 9
- NP-Vollständigkeit ... 9
- P=NP ? ... 10

Programmanleitung ... 10
- Parameterbedeutung ... 10
- Knotenverwaltung ... 10
- XML Dateien ... 11
- Programmablauf ... 11

Quellen ... 13

Traveling Salesman Problem (TSP)

Hinter dem Traveling Salesman Problem (Problem des Handlungsreisenden) verbirgt sich die Aufgabe n Punkte per Rundreise optimal miteinander zu verbinden. Auch wenn dieses Problem im ersten Moment harmlos klingt verbirgt sich dahinter eines der komplexesten und gleichzeitig praxisrelevantesten Probleme der Mathematik.

TSP gehört zur Klasse der NP-äquivalenten Probleme. Dadurch steigt die Laufzeit eines Deterministischen Algorithmus schneller als polynominell zur Problemgröße. Da daher selbst aktuelle Supercomputer bei anwendung von Brute Force Algorithmen sehr schnell überfordert wären (welche allerdings immer die Optimale Route liefern) werden zur Lösung vor allen Heuristiken angewendet. Aus diesem Grund ist das TSP mit der Zeit zu einem beliebten Problem zum entwickeln und testen neuer Optimierungsverfahren geworden.

In der Praxis findet das Problem anwendung in verschiedensten Anwendungen. Sei es in der Logistikoptimierung, die Steuerung von Bohrern in der Leiterplattenherstellung oder aber bei der Verlegung neuer Kommunikationsnetze etc.. TSP ist also nicht nur Mathematisch interessant sondern auch Wirtschaftlich.

Zur Mathematischen Lösung des Problems wird es mittels eines Graphen abgebildet.
Die Knoten stellen hier die einzelnen zu durchlaufenden Stationen (Städte) dar. Die Kanten bekommen eine Wertung welche problemspezifisch ist (Reisedauer / Reisekosten ...). Um das Problem etwas zu vereinfachen wird der Graph vollständig gemacht, also jeder Knoten wird mit jedem verbunden. Damit so keine in der Realität nicht vorkommenden Verbindungen in der kürzesten Route vorkommen wird diesen künstlichen Verbindungen eine sehr hohe Wertigkeit gegeben (z.B. Betrag aller reelen Kanten). Zusätzlich gilt es die 3 follgenden Spezialfälle zu beachten

Asymetrisches TSP

Beim asymetrischen TSP ist es möglich das die Kanten je nach Richtung unterschiedlich gewertet sind. Der Weg von a nach b kann also anders gewertet sein als der Weg von b nach a.
Es gibt hier $(n-1)!$ mögliche Lösungen

Symetrisches TSP

Hier gibt es nur eine Wertung zwischen zwei Knoten. Es gibt $\frac{(n-1)!}{2}$ mögliche Lösungen.

Metrisches TSP

Ein TSP ist ein Metrisches TSP, wenn die Kanten die Dreiecksungleichung erfüllen. Dies bedeutet das die direkte Verbindung zwischen zwei Knoten immer kürzer ist als der Weg über einen dritten Knoten. Ein Anwendungsbeispiel ist etwa eine Rundreise per Flugzeug. Der direkte Luftweg von Flughafen a nach Flughafen b ist immer kürzer als ein Umweg über Flughafen c.

Lösungsverfahren

Die einfachste Methode TSP zu lösen ist, alle Möglichkeiten auszurechnen und so die minimale Strecke zu finden. Unglücklicherweise ist dies schon bei relativ kleinen Problemen in der Praxis undurchführbar. Bei 20 Städten und einem Symetrischen TSP müsste man schon $\frac{(20-1)!}{2} = 60.822.550.204.416.000$ mögliche Lösungen ausrechnen. Selbst wenn das speichern eines einzelnen Weges und einer zugehörigen Bewertung nur etwa 50 Byte benötigen würde bräuchte man bei dieser Mothode etwa 3,04 millionen Terrabyte Speicher. Zum Vergleich: In Deutschland gibt es rund 2000 Städte.

Heuristiken

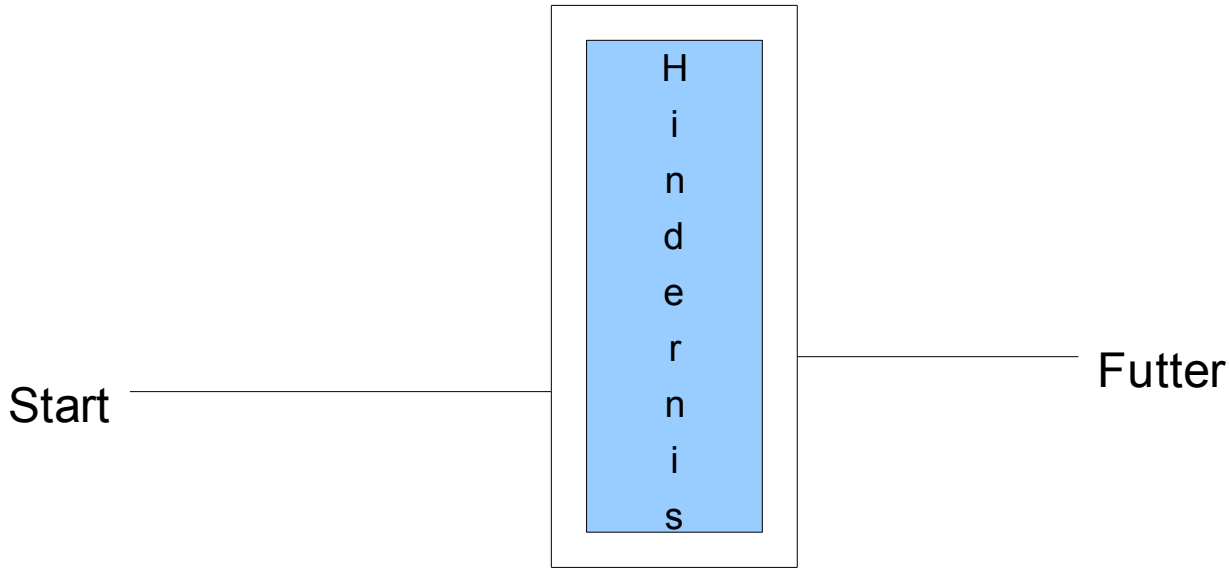
Aufgrund der Problematik eine optimale Lösung in kurzer Zeit zu finden versucht man mittels Heuristiken welche weit schneller arbeiten gute Wege zu finden. Eine Heuristik kann man auch als eine Strategie zur Lösungsfindung bezeichnen. Man versucht den Kreis der möglichen Lösungen einzugrenzen, behält aber einen Zufallsfaktor. Der von uns gewählte Ansatz zur Findung eines guten Weges ist ein Ameisenalgorithmus. Die sich dahinter befindliche Heuristik gehört zu den Metaheuristiken und simuliert eine Schwarmintelligenz.

Ameisenalgorithmus

Der Ameisenalgorithmus gehört zu jenen Algorithmen welche auf Basis eines natürlichen Vorbilds entstanden. Den ersten Ameisenalgorithmus stellte Marco Dorigo vor.

Vorbild

Abgeleitet wurde er von der Futterbeschaffung einer Ameisenkolonie. Ameisen hinterlassen beim laufen ein Pheromon welches weitere Ameisen anzieht. Gibt es auf dem Weg zum Futter mehrere Wege begehen Ameisen diese zu Beginn zufällig.



Die Ameise welche das Hindernis unten umgeht wird allerdings in gleicher Zeit öfter zwischen Start und Futter pendeln und somit mehr Pheromone auf dem Weg hinterlassen. Da dies wiederrum die anderen Ameisen anzieht werden diese auch den unteren Weg gehen und auch Pheromone abgeben. Es wird aber auch immer wieder Ameisen geben die den markierten Weg verlassen und so ist es möglich dass solche Ausreißer einen kürzeren Weg finden.
Ein weiterer elementarer Effekt der die Optimierung erst möglich macht ist, dass das Pheromon verdunstet. Da auf dem langen Weg die Ameisendichte wegen der größeren Entfernung geringer ist, verdunstet auf diesem das Pheromon schneller, wenn ein Ausreißer also einen kürzeren Weg finden und einige Ameisen beginnen diesem Weg zu folgen wird die Ameisendichte auf dem längeren Pfad immer schneller immer geringer und der kürzere Weg setzt sich durch.

Adaption

Das Problem wird als ein vollständiger gewichteter Graph dargestellt, wobei die Knotenmenge der Städteanzahl entspricht und das Kantengewichte die Entfernung widerspiegelt. Man platziert an jedem Knoten einen Agenten (eine Ameise) und lässt diesen eine Rundreise machen (Jeder Knoten darf nur einmal besucht werden). Wenn ein Agent alle Knoten besucht hat wird überprüft wie lang sein zurückgelegter Weg war. Ist er kurz, so wird die gesamte Strecke mit einer „hohen Pheromonkonzentration“ markiert, ist er lang nur mit einer schwachen Spur versehen. Das Speichen der Pheromondichte zwichen den Knoten wird über eine Matrix gelöst.
Zu Beginn bestimmt man einen Startpheromonwert, welcher für alle Kanten gesetzt wird. Auch führt man 2 Variablen α und β als Gewichtsfaktoren ein um zu bestimmen wie stark der Einfluß von Pheromon (α) und Kantengewicht (β) sein soll. Die maximale Menge an Pheromonen die ein Agent abgeben kann wird durch Q definiert und die Lebensdauer durch rho Auf diese Weise werden schlechte Strecken schnell unwichtig.

Algorithmus

Agenten setzen
alpha setzen
beta setzen
rho setzen
initial pheromone setzen
Q setzen

```
while !Abbruch
{
        für alle Agenten
        {
                gehe Weg
                aktualisiert Pheromone
        }
}
```

Auswahl des nächsten Knoten

Kernstück des Algorithmus ist die Auswahl des nächsten zu besuchenden Knoten.

Dazu wird für jeden gültigen potenziellen nächsten Knoten eine mit folgender Formel eine Warscheinlichkeit ausgerechnet.

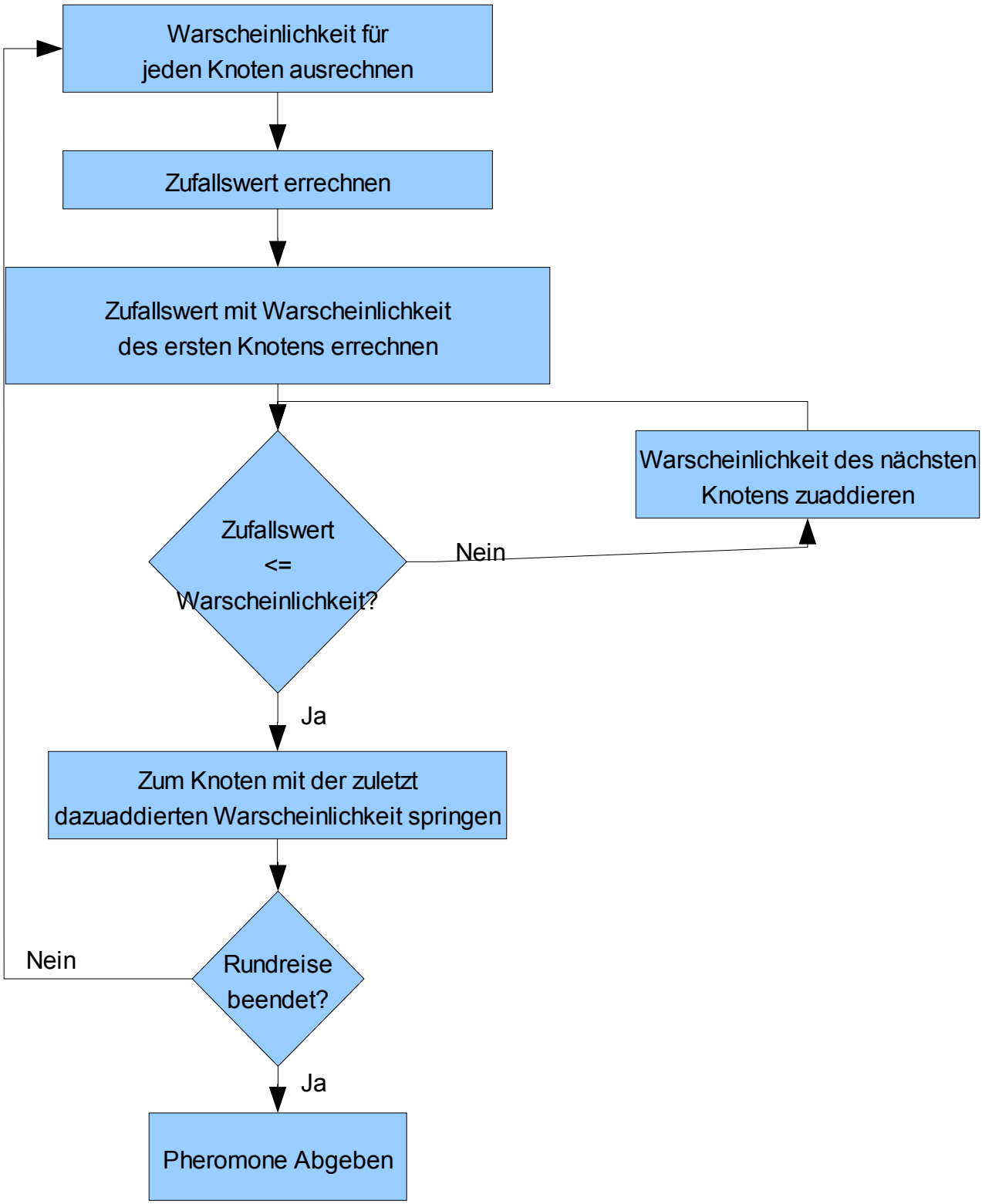
$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_k \tau_{ik}^{\alpha} \eta_{ik}^{\beta}}$ Wenn Knoten j noch nicht besucht wurde

$P_{ij} = 0$ Wenn Knoten j schon besucht wurde

Wobei:

α : *Gewichtung der Pheromonstärke*
β : *Gewichtung des Kantengewichts*
τ_{ij}^{α} : *Pheromonstärke zwischen Knoten i und j*
η_{ij}^{β} : *Kantengewicht zwischen Knoten i und j*
$\sum_k \tau_{ik}^{\alpha} \eta_{ik}^{\beta}$: *Summe der Produkte aller Pheromonstärken und Kantengewichte zwischen i und nicht besuchten Knoten k*

Diese Warscheinlichkeit wird für jede Kante vom Knoten i ausgerechnet und in ein Array gespeichert und ein Zufallswert wird errechnet. Dieser Zufallswert wird mit der Warscheinlichtkeit an der ersten Stelle des Arrays verglichen. Ist der Zufallswert kleiner oder gleich dem Arraywert wird der Knoten auf den an der Arraystelle verwiesen wird als nächster Knoten genommen und alles beginnt von vorn. Ist der Zufallswert größer wird zu der Warscheinlichkeit die Warscheinlichkeit an der nächsten Arraystelle addiert. Sollte der Zufallswert nun kleiner oder gleich sein wird der Knoten dieser zuaddierten Arraystelle als nächster Knoten genommen.



Dadurch erreicht man 2 Dinge: Zum einen ist es sehr warscheinlich das Kanten mit hohen Warscheinlichkeiten genommen werden (gute Kombination aus dichter Pheromonspur und Weglänge) aber es besteht auch die Möglichkeit das vorher uninteressante Teilstücke neu begangen werden (Ausreißer).

Aktualisieren der Pheromone

Hat ein Agent auf diese Weise jeden Knoten besucht und ist wieder am Start angekommen wird die Pheromonmatrix aktualisiert.

$$\tau_{ij}^{neu} = \rho * \tau_{ij}^{alt} + \sum_{k=1}^{Anzahl\ Agenten} \Delta \tau_{ij}^{k}$$

$$\Delta \tau_{ij} = \frac{Q}{L_k}$$

Wobei:

τ_{ij}^{neu} : *Neuer Pheromonwert zwischen i und j*

ρ : *Lebensdauer des Pheromons*

τ_{ij}^{alt} : *Alter Pheromonwert zwischen i und j*

Q : *Maximale Pheromonmenge welche ein Agent abgeben darf*

L_k : *Länge der Rundreise*

Die Pheromonmenge welche verteilt wird hängt also stark von der Weglänge aber auch von der Agentenmenge ab.

Laufzeit und Güte

Ameisenalgorithmen zeichnen sich vor allen dadurch aus in sehr kurzer Zeit brauchbare Ergebnisse zu liefern. Zusätzlich sind sie aber auch sehr anpassungsfähig. Durch die Simulation von Pheromonen ist es möglich auch zur Laufzeit Kantengewichte zu ändern und trotzdem fast sofort neue gute Strecken zu bekommen. Durch sein nicht exponentielles Wachtum ist es auch möglich größere Mengen an Knoten in die berechnung einzubeziehen.

Die von uns vorgestellte Form den nächsten Knoten auszuwählen hat mit der Zeit auch mehrere Abwandlungen erfahren. So fügt zum beispiel die Monte-Carlo-Auswahl ein random zu beginn hinzu und einen zusätzlichen Schwellwert. Ist das Random größer wird die oben gezeigte auswahl getroffen. Ist der Randomwert kleiner wird nach dem Stärksten Produkt aus Kantengewicht und Pheromonspur gesucht und dieser Knoten wird dann definitiv als nächstes besucht. Man verschiebt also den Zufallsfaktor und stärkt sehr gut bewertete Kanten zusätzlich (schützt diese also bis zu einem gewissen Grad vor dem Zufall). Die Laufzeit welche gebraucht wird damit alle Agenten eine Rundreise abschließen schwankt zwischen O(n³) und $O(n^4)$. Allerdings hat man meist schon nach sehr wenigen Komplettdurchläufen einen der besten Wege gefunden. Eine genaue aussage über die Güte der gefundenen Wege kann man nicht geben. Die Lösungen können beliebig schlecht / gut sein. Die Güte hängt vor allen von gut gewählten Parametern ab. Sehr ungeschickt gewählte Parameter führen zu einer sehr schlechten Lösung. Zur Verbesserung der Laufzeit wäre es möglich eine Liste der bereits gefundenen Wege zu speichern und entsprechende Abbruchbedingungen in die Heuristik einzufügen. Dies kann aber wie schon erwähnt zu sehr hohen Platzbedarf führen.

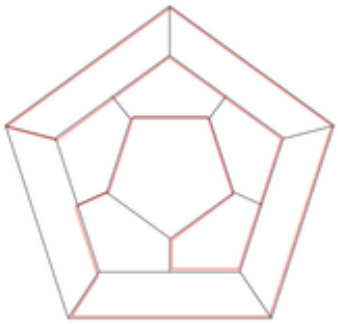
Komplexitätsklasse NP

Ein NP Problem ist ein Problem, welches von einer **N**ichtdeterministischen Turingmaschine in **p**olynomineller Zeit gelöst werden kann. Zu beachten sei hier das eine Nichtdeterministische Maschine nicht gleichwertig zu heutigen Computern ist und daher nur ein theoretisches Modell darstellt. NP Probleme sind bisher nur in Exponentieller Laufzeit auf einer Deterministischen Turingmaschine lösbar.
Eine polynominelle Laufzeit ist deshalb wichtig, da man davon ausgeht das ein Problem welches nicht stärker als mit einer Polynomfunktion wächst in vertretbarer Zeit lösbar ist, während man bei einem stärkeren Anstieg schon bei kleinen Problemen keine überschaubare Laufzeit mehr hat (unabhängig des Technologischen Stands).

NP-Vollständigkeit

Der Begriff NP-Vollständig ist ein von Stephen A. Cook eingeführter Begriff und beschreibt die Probleme welche die Problemstellung aller Sprachen aus der Komplexitätsklasse NP besitzen.

Sie müssen also zum einen mit einer Nichtdeterministischen Maschine in Polynominalzeit Lösbar sein und vor allen müssen sie NP-Schwer sein.
Das bedeutet das jede man ein Problem aus einem anderen schon bewiesen NP-Vollständigen Problem ableiten kann. Das Traveling Salesman Problem kann man z.B. leicht aus dem Hamiltonkreisproblem (welches zu den 21 Klassischen NP-Vollständigen Problemen gehört) ableiten.



Der Hamiltonkreis bezeichnet einen regulären Dodekaeder auf welchem jeder Per Rundreise jede Stadt genau ein mal besucht werden soll und Start = Endpunkt ist. Das größte Unterschied zum TSP ist also nur das dieses allgemein auf alle Graphen anwendbar ist.

Umgekehrt müsste ein Algorithmus der ein NP-Schweres Problem löst auch auf alle anderen NP Schweren Probleme ableitbar sein. Dies wiederrum würde den Beweis erbringen das P=NP ist.

P=NP ?

NP-Vollständige Probleme wären nur dann in polynomineller Zeit lösbar, wenn NP=P wobei P alle von einer Deterministischen Turingmaschine in **p**olynomineller Zeit lösbaren Probleme sind. Dies wäre nur dann wahr wenn NP eine Teilmenge von P wäre (P ist Teilmenge von NP da eine Deterministische Lösung Teilmenge einer Nichtdeterministischen ist). Um zu beweisen das gilt NP=P müsste ein Algorithmus gefunden werden welcher ein NP-Vollständiges Problem in polynomineller Zeit löst. Wenn solch ein Algorithmus gefunden würde, wäre dies der Beweis das alle NP Probleme auf Deterministischen Maschinen in vertretbarer Zeit lösbar sind. Allerdings wurde bis heute nichts derartiges nachgewiesen werden und man geht davon aus das NP ungleich P ist. Diese Frage ob P=NP ist eines der wichtigsten Probleme in der Mathematik und eine Lösung wird von Clay Mathematics Institute (CMI) mit einer Millionen Dollar belohnt.

Programmanleitung

Zum Starten des Programms reicht es die ant.jar Datei zu öffnen. Das Programm bietet für alle wichtigen Parameter recht sinnvolle default Einstellungen und einen schon fertig eingerichteten Graphen, welcher aber beliebig abänderbar ist.

Parameterbedeutung

Alpha: Gewichtung der Pheromonstärke. Bei höheren Werten wird die Pheromonstärke stärker in den Suchalgorithmus einbezogen. Defaultwert 1.0 (Siehe Kapitel: Auswahl des nächsten Knoten)

Beta: Gewichtung der Kantenlänge. Bei steigendem Wert wird diese stärker in die Suche mit einbezogen. Defaultwert 2.0 (Siehe Kapitel: Auswahl des nächsten Knoten)

Rho: Verfallszeit der Pheromone. Bei kleinen Werten verschwinden Pheromone schneller Defaultwert 0.5 (Siehe Kapitel: Aktuallisieren der Pheromone)

Tau: Startpheromonwert. Alle Kanten werden mit dieser Pheromonstärke Initialisiert Defaultwert 0.1

Q: Maximale Pheromonmenge die ein Agent abgeben darf pro Rundreise. Defaultwert 100 (Siehe Kapitel: Aktuallisieren der Pheromone)

Knotenverwaltung

Add Node: Fügt einen Knoten hinzu. Dieser wird immer mit den Koordinaten 0;0 initialisiert. Sinnvolle Werte sollten also eingefügt werden.

Remove Node: löscht den letzten Knoten.

XML Dateien

Eine Knotenliste kann inclusive aller Parameterwerte in eine XML Datei gespeichert werden, oder aus einer solchen ausgelesen werden.

Über File -> Save wird gespeichert, über File -> Open kann eingelesen werden.

Die einzulesende Datei sollte dabei folgende Struktur haben:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXPERIMENT>
		<ALPHA>1.0</ALPHA>
		<BETA>2.0</BETA>
		<RHO>0.5</RHO>
		<TAU>0.1</TAU>
		<Q>100</Q>
		<NODES>
				<NODE x="5" y="5"/>
				...
				...
				<NODE x="25" y="15"/>

		</NODES>
</EXPERIMENT>
```

Programmablauf

Per Start Simulation wird ein Fenster geöffnet welches den erstellten Graphen anzeigt und noch einmal alle Parameter + Legende zeigt.

Die Farbgskala zeigt die Pheromondichte. Von Weiß (kaum Pheromone) bis Rot (viele Pheromone)

Cycles: Menge der Komplettdurchläuft der Agenten.

Best tour: Die beste Tour des Aktuellen durchlaufs

Best tour length: Lenge der Aktuellen besten Tour

Best ever tour: Die beste Tour gefundene Tour

Best ever tour length: Lenge besten gefundenTour

Über den Schieberegler unterhalb der Bedienelemente kann man die Pause zwischen den Durchläufen einstellen. Dies ist für Präsentationszwecke Sinnvoll. Default: 1000ms

Per Run Button wird das Programm gestartet. Suspent hält das Programm an und Zeigt alle aktuelle gegangene Wege. Show Route Zeigt die beste gefundene Route. Reset bricht die Suche ab und startet neu.

Es können mehrere Fenster mit verschiedenen Graphen gleichzeitig geöffnet sein.

Quellen

Internet:

Http://de.wikipedia.org/*

http://www.tsp.gatech.edu

http://elib.zib.de/pub/UserHome/Groetschel/Spektrum/index2.html

Und follgende Seiten im besonderen

http://de.wikipedia.org/wiki/NP-Vollständig

http://de.wikipedia.org/wiki/NP_(Komplexitätsklasse)

http://de.wikipedia.org/wiki/P/NP-Problem

http://de.wikipedia.org/wiki/Ameisenalgorithmus

Das vorgestellte Java Programm wurde auf Basis von Sinclair, M.C., AntTSP Java Applet erstellt.

Bücher:

Algorithmen - Eine Einführung von Thomas Cormen, Charles Leiserson, Ronald Rivest