Zu Übung08, Besprechung: 12.07.2012 14^{00} im Neue Chemie

Aufgabe 8.01: Verständnisfragen

a) Beim Optimierungsverfahren Random-Interchange wird ein Austausch auch beibehalten, falls dies zu einer Verschlechterung der Kosten führt.
Richtig / Falsch

b) Beim Optimierungsverfahren Simulated Annealing wird ein Schritt, welcher zu einer Verbesserung führt immer ausgeführt. Richtig / Falsch

c) Beim Kernighan-Lin-Algorithmus wird eine Vertauschung nicht durchgeführt, wenn Sie zu einer Verschlechterung der Gesamtkosten führt. Richtig / Falsch

d) Beschreiben Sie das Prinzip der Rekombination (Kreuzung) in Bezug auf Evolutionäre Algorithmen:

vermischen von erbmassen, nachkommen ersetzen vorfahren

e) Der Cache kompensiert den Geschwindigkeitsunterschied zwischen prozessor und _____.

f) Definieren Sie den Begriff "Write through" in Bezug auf Caches und nennen Sie einen Vor- und einen Nachteil.

g) Techniken zur Beschleunigung der Befehlsausführung sind pipelining, superskalar und cache.

h) Bei der Interpretation eines Befehls im Speicher wird unterschieden, ob das niedrigstwerte Byte an der ersten Adresse („big endian“, z.B. Apple), oder letzten Adresse („little endian“, z.B. Intel x86) gespeichert wird.

i) Was ist der Hauptunterschied zwischen der von-Neumann- und der Harvard-Architektur? Hinweis: Sie können Ihre Lösung auch graphisch darstellen.

neumann: daten und adressen werden über eine verbindung ausgetauscht ->flaschenhals

harvard: adressen und datenspeicher sind getrennt und haben eigene verbindungen zum prozessor

Aufgabe 8.02: Travelling Salesman

Das Problem des Handelsreisenden ist eines der Bekanntesten in der Informationstechnik und wurde oft in vielen verschiedenen Richtungen untersucht. Es besteht darin, dass ein Reisender *x* Städte, welche auf einer Landkarte zufällig verteilt sind, nacheinander besuchen möchte. Am Ende der Reise möchte er wieder am Ausgangspunkt ankommen. Dabei möchte er jede Stadt nur einmal besuchen und insgesamt den kürzt möglichen Weg zurücklegen.

→ Siehe auch: http://de.wikipedia.org/wiki/Problem_des_Handlungsreisenden

Das Problem soll mit Hilfe des Simulated Annealing Algorithmus

http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo41.php

oder

http://de.wikipedia.org/wiki/Simulierte_Abk%C3%BChlung

gelöst werden.

Der folgende Pseudocode beschreibt den Algorithmus:

ALGORITHMUS:

```
T = T0;                          //Anfangswert der Temperatur
X = j0;                          //Anfangskonfiguration
while( !stop_criterion ) {       //Endtemperatur, oder weniger als Min Austauschungen
  while( !loop_criterion ) {     //Mindestanzahl Versuche bei einer Temperatur
    j = generate( X );           //Generiere neue Konfig. (Austausch zweier Module)
    if( accept( cost( j ), cost( X ), T ) )      //spezielle Akzeptanzfunktion
      X = j;                                     //akzeptierte Lösung abspeichern
    T = update( T );             //langsames Senken der Temperatur (z.B. T := 0.9T)
  }
}
```

AKZEPTANZFUNKTION:

```
//liefert eine 1, wenn ein Austausch akzeptiert wird, wenn nicht, eine 0
accept( cost( j ), cost( i ), T ) {
  diff_cost = cost( j ) - cost( i );       //Differenz der Kosten berechnen
  y = exp( -diff_cost / T );  //je höher die Temperatur, desto größer wird y
  r = random;                 //Zufallszahl zwischen 0 und 1 generieren
  if( r < y )
    return 1;                 //Neue Strecke akzeptiert
  else
    return 0;                 // Neue Strecke nicht akzeptiert
}
```

Die Funktion *generate( )* beschreibt dabei die Generierung einer neuen möglichen Lösung / Weges. Diese kann erreicht werden, indem zufällig ein Teilstück gewählt wird und dieses in der Zukunft in umgekehrter Reihenfolge durchlaufen wird (Abbildung 2). Eine andere Möglichkeit besteht darin, eine Stadt zu einem anderen Zeitpunkt zu besuchen und die Reihenfolge aller anderen Städte beizubehalten (Abbildung 3).

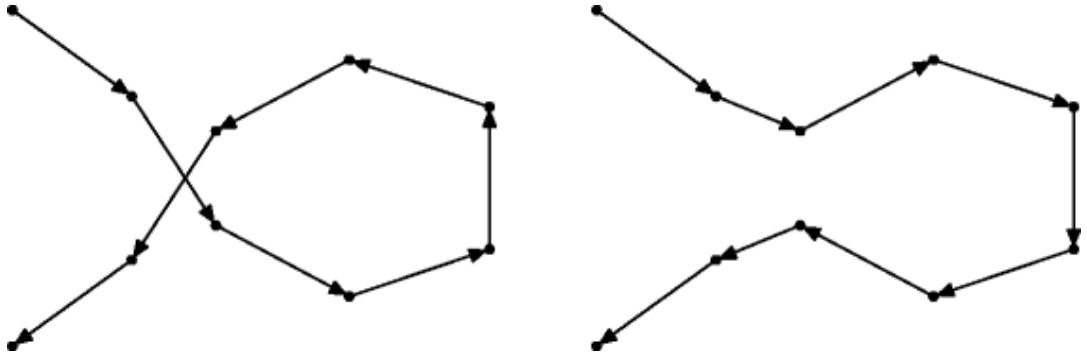

Abbildung 2: Neuer Weg durch Umkehr der Laufrichtung in einem Teilstück

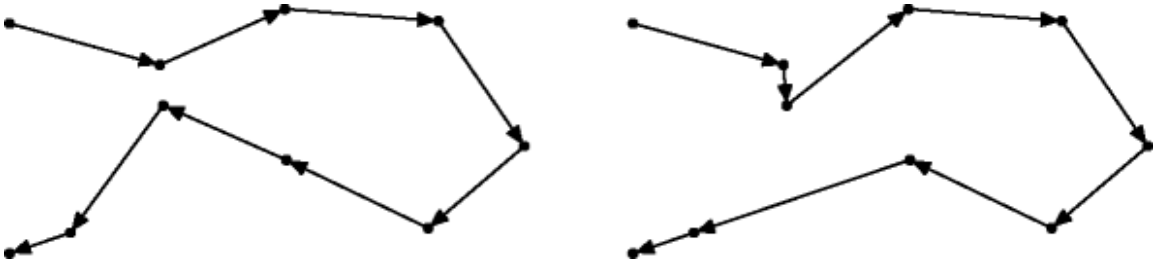

Abbildung 3: Neuer Weg durch Veränderung der Besuchsreihenfolge der Städte

Die Funktion *cost( )* beschreibt die Kosten- bzw. Fitnessfunktion. Dabei gilt in dieser Aufgabenstellung die insgesamt zurückgelegte Strecke (um alle Städte nacheinander bei der gewählten Route zu besuchen) als Kostenfunktion.

Gegeben ist die folgende Tabelle mit 6 Städten und deren Abständen untereinander. Schreiben Sie ein C++ Programm, das die gegebenen Daten in einer geeigneten Datenstruktur speichert. Implementieren Sie anschließend den Simulated Annealing Algorithmus, um einen kurzen Weg für den Handelsreisenden zu finden. Bei der Generierung einer neuen Lösung können Sie eine der zwei vorgestellten Möglichkeiten auswählen. Geben Sie am Ende ihr Ergebnis auf dem Bildschirm aus.

Tabelle 1: Abstände der Städte untereinander

	Aachen	Bonn	Düsseldorf	Frankfurt	Köln	Wuppertal
Aachen	—	91	80	259	70	121
Bonn	91	—	77	175	27	84
Düsseldorf	80	77	—	232	47	29
Frankfurt	259	175	232	—	189	236
Köln	70	27	47	189	—	55
Wuppertal	121	84	29	236	55	—

Abbildung 4: Beispielausgabe zu Travelling Salesman

Aufgabe 8.03: Cacheorganisation

Angenommen, dass ein Prozessor einen Hauptspeicher der Größe 1 GByte hat, wo jedem einzelnen Byte im Speicher eine 30 Bit lange Adresse zugeordnet wird. Zusätzlich verfügt der Prozessor über einen Cache Baustein der Technologie SRAM, der 512 KBytes (nur Daten) vom Hauptspeicher aufnehmen kann (unabhängig vom notwendigen Speicher zur Speicherung der weiteren Bits, wie z.B. Tag). Der Cache hat eine Blockgröße von 8 Bytes. Zusätzlich wird auf dem Cache für jeden Block ein Valid-Bit vorgesehen.

a) Nennen Sie die drei Möglichkeiten der Cacheorganisation.

b) Berechnen Sie für jeden Typ die Anzahl der Cache-Blöcke und der Cache-Sets.

   Hinweis: Bei dem einen Typ, wo eine Variable n festgelegt werden soll, nehmen Sie den Wert 4.

c) Erläutern Sie zu jeden Typ die Speicherorganisation des Caches bezüglich im Cache abzulegenden Daten und Adressierung.

d) Berechnen Sie für jeden Typ die in Wirklichkeit notwendige Speicherkapazität für den Cache Baustein.

a) voll assoziativ, direkt abbildend, n-wege assoziativ

b) 1. 1 cache block, 8 cache sets

   2. 1 cache block,