WiMa-Praktikum OR
Wintersemester 2013/14

Dr. Jens Maßberg

# Operations Research Praktikum
# Aufgabenblatt 4 "Bonuszettel"

## 1 Einleitung

Die Hauptaufgabe auf diesem Aufgabenblatt wird sein, einen Travelling Salesman Tour (TSP) Algorithmus zu implementieren.
Die Implementierungen werden weiterhin in C++ unter Verwendung von Code::Blocks [1] erfolgen.

## 2 TSP Algorithmus

Informieren Sie sich über die Definition von "Travelling Salesman Touren" (TSP-Touren, Problem des Handlungsreisenden, siehe zum Beispiel [2]).
Eine Instanz $(K_n, c)$ des Travelling Salesman Problems besteht aus dem vollständigen ungerichteten Graphen $K_n$ und einer nicht-negativen Kostenfunktion $c : E(G) \to \mathbb{K}_{\mathbb{n}\geq 0}$ auf den Kanten von $K_n$. Wir nehmen an, dass die Kostenfunktion die Dreiecksungleichung einhält, d.h. $c(ac) \leq c(ab)+c(bc)$ für je drei Knoten $a, b, c \in V(G)$.
Die Aufgabe ist es, einen kürzeste Travelling Saleman Tour in $G$ zu berechnen mit minimalen/geringen Kosten, d.h. einen bezüglich $c$ kürzesten Kreis in $G$, der jeden Knoten von $G$ genau einmal enthält. Ein guter Approximationsalgorithmus für das Travelling Salesman Problem ist der Double-Tree-Algorithmus (s.u.).
Implementieren Sie den Double-Tree-Algorithmus. Informieren Sie sich über die verwendeten Begrifflichkeiten (z.B. Euler-Tour, siehe z.B. [2]): Verwenden Sie die in den von Ihnen für die vorherigen Aufgabenblätter implementierten Datenstrukturen und Algorithmen für Minimum Spanning Trees.
Überlegen Sie sich, wie Sie das Ergebnis, d.h. die berechnete Travelling Salesman Tour sinnvoll ausgeben können.
Testen Sie Ihren Algorithmus auf

- eigenen Testinstanzen und
- auf zufälligen euklidischen Instanzen, die Sie mit der entsprechenden Funktion vom zweiten Aufgabenzettel generieren.

**Algorithmus 2.1** (`Double-Tree Algorithmus` für metrische TSP)**.**

**Input**: *Eine* METRISCHE TSP *Instanz* $I = (K_n, c)$.
**Output**: *Eine TSP-Tour* $C$ *in* $K_n$.

1 $T \leftarrow$ MINIMUM SPANNING TREE *für* $I$;
2 *Verdoppele alle Kanten in* $T$ *und konstruiere einen Hamiltonischen Kreis* $C$ *aus einer Euler-Tour* $E$ *im erhaltenen Graphen* $G$;
3 **return** $C$;

# 3 Programmaufruf und Dokumentation

Wie in den bisherigen Zetteln sollen die einzelnen Funktionen des Programms über Kommandozeilenparameter aufgerufen werden können. Insbesondere soll der folgende Parameter unterstützt werden:

```
<PROGRAMMNAME> -TSP <INPUTDATEINAME>
```

liest einen vollständigen Graphen $G$ mit Kostenfunktion $c$ aus der Datei mit Namen `<INPUTDATEINAME>` ein und berechnet mit Hilfe des DOUBLE-TREE-ALGORITHMUS eine TSP-Tour in $G$.
Sie können wieder davon ausgehen, dass alle Ein- und Ausgabedateien im gleichen Verzeichnis wie das Programm liegen. Pfadangaben müssen nicht unterstützt werden.

Schreiben Sie wieder eine Kurzdokumentation (höchstens zwei Seiten).
In Ihrer Dokumentation beschreiben Sie, welche Schwierigkeiten bzw. Probleme bei der Implementierung über die hier beschriebene Aufgabenstellung hinaus auftraten und wie Sie sie gelöst haben.
**Bitte schicken Sie bis zum 14.04.2014 den Quellcode Ihrer Implementation an** `<jens.massberg@uni-ulm.de>` .

# Literatur


[1] Code::Blocks, http://www.codeblocks.org

[2] Bernhard Korte und Jens Vygen, *Cominatorial Optimization*, ab Auflage 4 (auch in Deutsch).