5 Näherungsverfahren

Im Folgenden versuchen wir den kürzesten Weg über eine sog. Heuristik zu finden, das heißt, man spaltet das Problem in mehrere Teilprobleme auf, die zu einer möglichst guten Lösung führen. Dabei kann es natürlich sein, dass eine sehr schlechte Lösung (in unserem Beispiel ein viel zu langer Weg) das Resultat der Heuristik ist. Es ist jedoch auch nicht ausgeschlossen auf diese Weise die optimale Lösung zu treffen". Der Approximationsalgorithmus für das Problem des Handlungsreisenden, der hier vorgestellt wird, bedient sich der Berechnung eines minimalen Spannbaums (minimum spanning tree - MST) eines vollständigen Graphen und der sogenannten Tiefensuche. Voraussetzung dabei ist, dass die Dreiecksungleichung gilt.

5.0.9 Dreiecksungleichung

Seit G ein vollständiger Graph G=(V,E) mit einer Kostenfunktion

$$c(i,j) = k; i,j \in E; k \in \mathbb{Z}$$

G ist hamiltonsch, da er vollständig ist. Wir sind an der *kürzesten* Hamilton-Tour interessiert. Die Gesamtkosten des Graphen werden beschrieben durch die Funktion c(A); A $\subseteq$ E.

c(A) = $\sum_{(i,j)\in A}$ c(i,j)

Betrachten wir einen (vollständigen) kantengewichteten Graphen mit drei Knoten und drei Kanten, wie in Abbildung XX1 zu sehen.

In vielen Situationen ist es immer günstiger direkt von 1 nach 3 zu gehen, als einen Umweg über 2 zu nehmen. Anders gesagt: Den Knoten 2 aus dem Weg von Knoten 1 nach Knoten 3 herauszunehmen erhöht niemals die Gesamtkosten des Weges.

$$\Rightarrow c(1,3) \leq (1,2) + c(2,3)$$

Sind die Knoten 1, 2 und 3 beispielsweise Städte und die Kanten markieren die Luftlinienentfernung zueinander, so ist die Dreiecksungleichung sicher erfüllt. Stellen die Kanten nun jedoch Reisekosten dar, so gilt die Ungleichung nicht mehr unbedingt: Oft ist es günstiger einen Flug mit einem

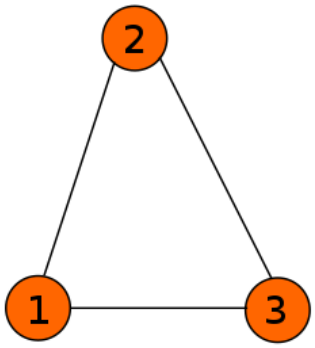


Abbildung 5.1: ein vollständiger Graph mit 3 Knoten

Zwischenstop über eine andere Stadt zu nehmen, als den Direktflug vom Heimatort zum Ziel zu buchen. Für die folgenden Schritte ist es also notwendig, dass die Dreiecksungleichung auf jeden Fall erfüllt ist. Wir kommen später wieder auf diese Eigenschaft zurück.

5.0.10 Der Minimale Spannbaum

Über den minimalen Spannbaum eines vollständigen Graphen lässt sich eine Rundreise berechnen deren Gesamtkosten maximal 2 mal so groß sind wie die optimale Rundreise:

$$c(T) \leq c(A) \leq 2 * c(T)$$

wobei A die Kanten der approximierten Rundreise und T die Kanten der optimalen Rundreise repräsentieren. Der minimale Spannbaum selbst legt eine untere Grenze für den kürzesten Hamilton-Pfad fest. Wir wissen jetzt also, dass das Gesamtgewicht einer optimalen TSP-Tour H* größer oder gleich dem Gesamtgewicht des minimal aufspannenden Baums T ist:

$$c(T) \leq c(H^*)$$

Ein minimal aufspannender Baum aus einem Graphen lässt sich mit relativ geringem Aufwand berechnen. Die beiden bekanntesten Algorithmen Kruskal und Prim gehören den Greedy-Algorithmen an. Sie arbeiten nicht deterministisch, das heißt für ein und denselben Graphen gibt es mehrere minimal aufspannende Bäume, die jedoch alle die gleiche Eigenschaft der geringsten Gesamtkosten teilen. Wir werden hier insbesondere auf den Kruskal Algorithmus eingehen, der von dem US-Amerikanischen Mathematiker Joseph Kruskal um 1956 vorgestellt wurde:

```
G = (V,E,w): ein vollstaendiger, gewichteter Graph
kruskal(G)
1  E' ← ∅
2  L ← E
3  Sortiere die Kanten in L aufsteigend nach ihrem Kantengewicht.
4  solange L ≠ ∅
5       waehle eine Kante e ∈ L mit kleinstem Kantengewicht
6       entferne die Kante e aus L
7       wenn der Graph (V,E' ∪ { e }) keinen Kreis enthaelt
8            dann E' ← E' ∪ { e }
9  M = (V,E') ist ein minimaler Spannbaum von G.
```

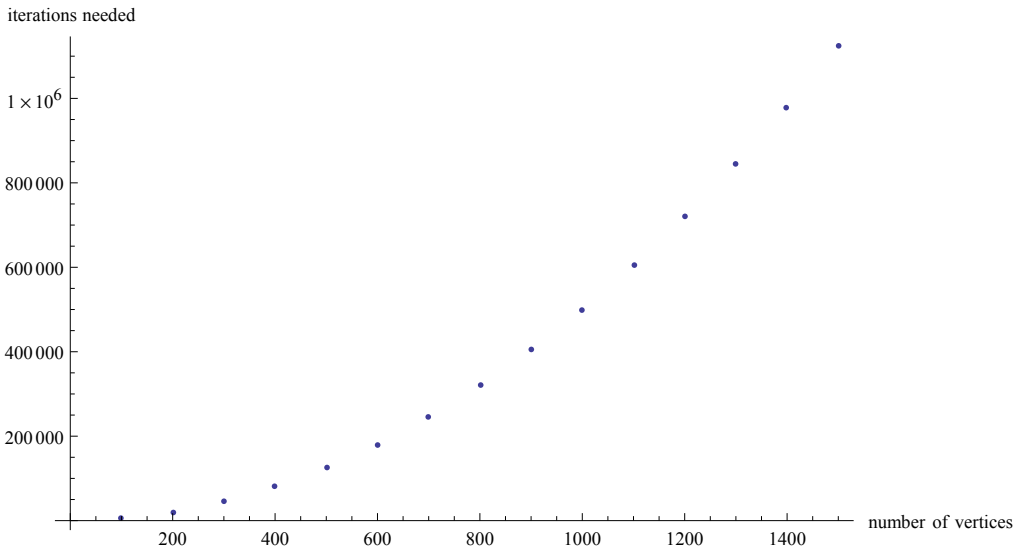
Die Laufzeit des Algorithmus hängt im wesentlichen von der gewählten Datenstruktur ab, die den Graphen modelliert. Außerdem lässt er sich bei genauerem hinsehen auf zwei Teilprobleme reduzieren, nämlich:

- Sortieren der Kanten und
- gibt es einen Kreis beim Einfügen einer Kante in den zu konstruierenden MST?

Natürlich spielen andere Faktoren wie das Initialisieren einer leeren Menge (Zeile 1) oder das Zuweisen der Kantenmenge E des vollständigen Graphen an eine Arbeitsmenge L (Zeile 2) eine Rolle, doch für gewöhnlich passiert dies in konstanter Zeit $\Theta(1)$. Viel wichtiger sind die beiden oben genannten faktoren. Eine eigene Implementierung in Java zeigt, dass die Anzahl der Iterationen im Worst-Case quadratisch steigt. Das ist nicht weiter verwunderlich, denn die Anzahl der Kanten in einem vollständigen Graphen beträgt:

$$\frac{n * (n-1)}{2}$$

wobei n die Anzahl der Knoten ist.



Die Anzahl der benötigten Iterationen für den MST-Kruskal bei 0 - 1500 Knoten

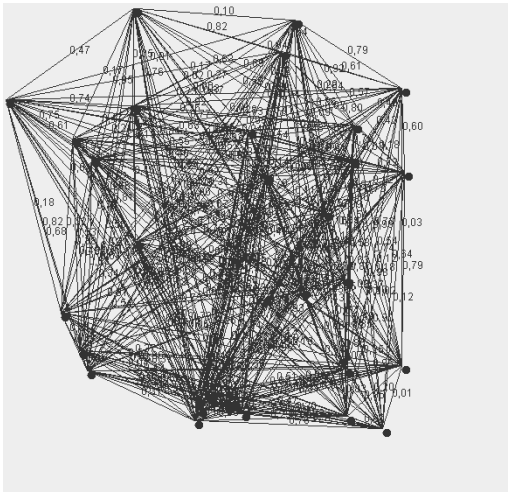


Abbildung 5.2: ein vollständiger Graph mit 30 Knoten

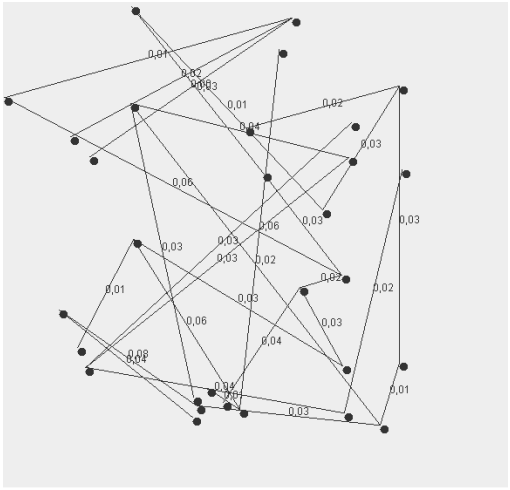


Abbildung 5.3: der zugehörige minimal aufspannende Baum

5.0.11 Tiefensuche

Aufbauend auf einen erzeugten minimalen Spannbaum, welcher, wie oben beschrieben die untere Grenze des Gesamtgewichts einer optimalen TSP-Tour beschreibt, lässt sich eine sogenannte Tiefensuche (depth- first-search, DFS) durchführen. Dabei wird der Graph zuerst nach den von einem angegeben Startknoten aus am weitesten entfernt liegenden Knoten durchsucht. Knoten, die schon einmal von dem Suchalgorithmus besucht wurden werden markiert und (in unserem Fall am besten) geordnet nach dem Zeitpunkt der Markierung auf einem Stack gespeichert. Enthält ein markierter Knoten mehrere Nachbarknoten, so besitzen die alle sie gleiche Priorität und werden, falls möglich, gleichzeitig markiert. In der Praxis speichert man die Knoten, die es weiter zu durchsuchen gibt ebenfalls auf einem eigenen Prioritätsstack. Wurde ein Ast des Gerade zu bearbeitenden Knotens abgearbeitet, so wird dieser von der Prioriätsliste genommen und der nächste Ast wird durch sucht. Die Reihenfolge der markierten (besuchten) Knoten liefert implizit eine Näherungslösung für die TSP-Tour. Es wird also ein Hamilton-Kreis gefunden, ob dieser jedoch wirklich auch der optimale ist, kann nicht garantiert werden.

```
Approx–TSP–Tour(G)
1  Waehle einen Knoten s ∈ V[G] als Start–Knoten
2  Berechne einen MST T fuer G vom Knoten s
3  Lass L eine Liste von Knoten sein, die druch
       eine Tiefensuche gefunden wurde.
4  Gib den Hamilton–Kreis H zurueck, welcher die
   Knoten in der Reihenfolge L besucht.
```

Wir wollen nun zeigen, dass unsere obige Behauptung, dass sich über Approx-TSP-Tour ein Hamilton-Kreis finden lässt, dessen Gesamtkosten im schlimmsten Fall 2x so hoch ist wie die einer optimalen TSP-Tour:

Lass H^* eine optimale TSP-Tour für den gegeben Graphen G mit den Knoten Knoten a, b, c, d, e, f, g, h sein. Sei T der zugehörige aufspannende Baum zu G. Weil T eine untere Grenze für die Kosten von H^* ist, folgt:

$$c(T) \leq c(H^*).$$

Durchläuft man T komplett, also von einem Startknoten a über alle Knoten und wieder zurück, so hat man einen Weg W der folgende Knoten durchläuft:

$$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$$

Dieser Weg durchläuft jede Kante des Baumes genau zwei mal:

$$c(W) = 2c(T) \Leftrightarrow c(T) = \frac{c(W)}{2}$$

$$\Rightarrow \frac{c(W)}{2} \leq c(H^*)$$

$$\Rightarrow c(W) \leq 2c(H^*).(1)$$

Allerdings ist W keine Tour, da Knoten mehr als einmal durchlaufen werden. Hier kommt die Dreiecksungleichung ins Spiel, die für unseren Graphen als Voraussetzung gilt. Deshalb können wir immer einen Knoten "überspringen", ohne dass sich die Kosten des Weges W erhöhen. Für unser Beispiel ergibt dies die Knotenfolge:

$$a, b, c, h, d, e, f, g, a.$$

Diese Folge ist genau die, die wir über die (geordnete) Tiefensuche erhalten. Wenn ein Weg H diesem Weg entspricht, so ist H eine Hamilton-Tour, da jeder Knoten nur einmal besucht wird. H ist genau die Tour, die über Approx-TSP-Tour zurückgegeben wird. Wir können nun sagen, dass:

$$c(H) \leq c(W).(2)$$

Aus (1) und (2) folgt

$$c(H) \leq c(W) \leq 2c(H^*)$$

$$\Rightarrow c(H) \leq 2c(H^*).$$

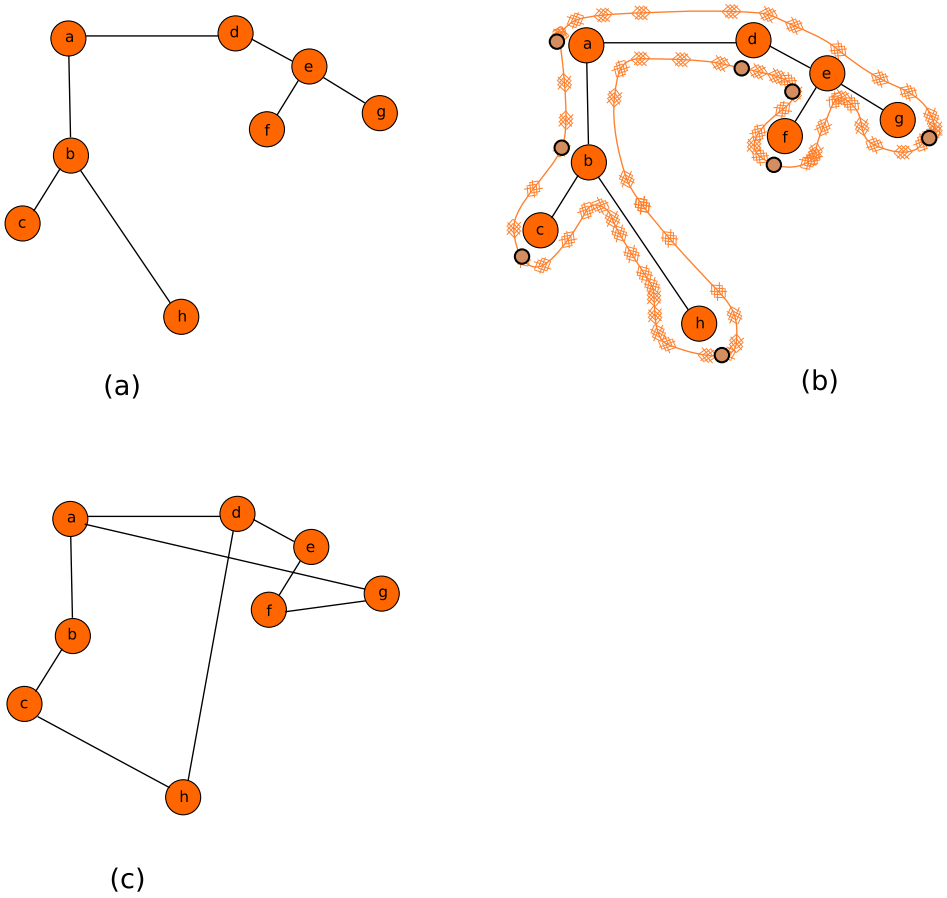


Abbildung 5.4: kommt noch