Aufspannende Teilgraphen mit kürzesten Umwegen

Sparse Spanners

Bachelor-Thesis von Christoph Werner Acker aus Bad Orb

August 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Mathematik
Discrete Optimization

Aufspannende Teilgraphen mit kürzesten Umwegen
Sparse Spanners

Vorgelegte Bachelor-Thesis von Christoph Werner Acker aus Bad Orb

1. Gutachten: Dr. habil. Marco Lübbecke
2. Gutachten: Prof. Dr. Stefan Ulbrich

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 4. August 2010

(Christoph Acker)

Inhaltsverzeichnis

1 Einführung

Diese Arbeit befasst sich mit aufspannenden Teilgraphen mit kürzesten Umwegen. Diese Graphen, sog. t-Spanner, werden beispielsweise bei der theoretischen Analyse von Computernetzwerken, aber auch bei der Untersuchung von Schienen- oder Straßennetzen verwendet. Ein weiteres Anwendungsfeld ist die Konstruktion von Netzwerken, in denen die Wege zwischen zwei Knoten im Verhältnis zum kürzesten Weg möglichst kurz gehalten werden sollen.
In dieser Arbeit wird eine Klassifizierung des Problems, eben diese Teilgraphen auf einem bestimmten Netzwerk zu finden, als NP-vollständig gegeben. Weiterhin beschäftigt sich diese Arbeit mit der Suche nach oberen und unteren Schranken für die Kantenanzahl, die ein solcher Teilgraph enthält. Dabei werden gerichtete und ungerichtete Graphen betrachtet. Außerdem wird ein Algorithmus zur Lösung dieses Problems gegeben und auf ein Fallbeispiel aus einem Projektseminar mit der Deutschen Bahn AG angewendet.
Kapitel 2 fasst grundlegende Definitionen zusammen. Hierbei sei auch auf Standartwerke zur Graphentheorie von Diestel (Die06), Wettinghaus (Wet03) oder auch auf „Algorithmen - Eine Einführung“ von Cormen et al. (CLRS07) verwiesen. Die Kapitel 3 und 4 basieren auf dem Artikel „Graph Spanners“ (PS89) aus dem Journal of Graph Theory.
Die beiden Kapitel 5 und 6 sind weitergehende Überlegungen zu einem Algorithmus mit einem Anwendungsbeispiel.
Diese Arbeit stellt meine Bachelorthesis für den Abschluss als Bachelor of Science des Studienganges Wirtschaftsmathematik an der Technischen Universität Darmstadt im Sommersemester 2010 dar.

2 Grundlegende Definitionen

Wir beginnen mit dem nötigen Defintionen, die wir für die Betrachtung der aufspanndenden Teilgraphen mit kürzesten Umwegen brauchen.

Dazu definieren wir zunächst einen Graphen:

Defintion 2.1. *Ein Graph $G = (V, E)$ besteht aus*

1. *einer Knotenmenge $V = \{v_1, v_2, \ldots, v_n\}$ mit n Knoten (vertices)*
2. *einer Kantenmenge $E \subseteq V \times V$ mit $m = |E|$ Kanten (edges)*

Ein Graph heißt *gerichtet*, wenn die Kanten einen Anfangs- oder Startknoten sowie einen Ziel- oder Endknoten besitzen, andernfalls *ungerichtet*. Von einem Graphen können auch nur Teilgraphen, die wir nun definieren wollen, eine wichtige Rolle spielen:

Defintion 2.2. *Ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ heißt $Teilgraph$ oder $Subgraph$ von G.*

Wenn die Menge der Knoten in zwei Gruppen aufgeteilt werden kann, spricht man von einem bipartiten Graphen, der eine spezielle Form von Graphen darstellt.

Defintion 2.3. *Ein Graph $G = (V, E)$ heißt bipartit, wenn sich die Knotenmenge V so in zwei disjunkte Teilmengen X und Y aufteilen lässt, dass jede Kante aus E einen Knoten aus X mit einem Knoten aus Y verbindet. Damit lässt sich G auch schreiben als: $G = (X, Y, E)$.*

Einen Graph bezeichnen wir als *zusammenhängend*, wenn zwischen beliebigen Knotenpaaren u, v $(u, v \in V)$; ein Weg in G besteht.
Ein Weg ist eine Folge $w_1, w_2, \ldots, w_k$ von Knoten aus V mit $(w_i, w_{i+1}) \in E$ für alle $i = 1, \ldots k-1$.
Für den Begriff der Entfernung in Graphen ist noch der Begriff der *Kapazität* einer Kante von Bedeutung. Formal ausgedrückt bedeutet das, dass wir eine Funktion $c : E \to \mathbb{R}$ haben, die jeder Kante des Graphen ein bestimmtes Gewicht $c(e)$ zuordnet. Damit können wir die Länge eines Weges als die Summe der Kantengewichte der Kanten, die auf dem Weg liegen, ausdrücken
In Kapitel 4 benötigen wir den Begriff des Kantenzusammenhanges, der im Folgenden gegeben wird:

Defintion 2.4. *Der Kantenzusammenhang $\lambda(G)$ eines gerichteten Graphen $G = (V, E)$ mit einer Kapazitätsfunktion $c : E \to \mathbb{R}$ ist das minimale Gesamtgewicht $c(E') := \sum_{e' \in E'} c(e)$ von Kanten $E' \subseteq E$, so dass der Graph $G' = (V, E \backslash E')$ unzusammenhängend ist ($\lambda(G) := \min\{c(E') | E'$ wie oben$\}$).*

Weiterhin wird auch der Begriff eines Spannbaums benötigt:

Defintion 2.5. *Ein aufspannender Baum oder Spannbaum eines Graphen $G = (V, E)$ mit $|V| = n$ ist ein zusammenhängender Teilgraph $G' = (V, E')$ von G, der keinen Kreis besitzt. Ein Kreis ist ein Weg in einem Graphen, dessen Anfangs- und Endknoten übereinstimmen.*
Ein minimaler Spannbaum in einem gewichteten Graphen ist ein Spannbaum mit minimalem Gewicht, d.h. die Summe über alle Kantengewichte, die in dem Spannbaum enthalten sind, ist minimal.

Bemerkung 2.1. *Ein Spannbaum hat $n - 1$ Kanten.*

Kommen wir nun zur Definition des *t-Spanners*, eines für diese Arbeit zentralen Begriffes:

Defintion 2.6. *Ein Teilgraph G' = (V, E') eines zusammenhängenden Graphen G = (V, E) ist ein t-Spanner, wenn gilt:*
Für alle $u, v \in V$ ist

$$\frac{dist(u,v,G')}{dist(u,v,G)} \leq t$$

Dabei bezeichnet dist(u, v, G) die Länge eines kürzesten Weges von u nach v in G.
t wird auch als Stretch-Faktor bezeichnet.

Das folgende Lemma gibt eine äquivalente und manchmal einfacher zu handhabende Formulierung eines t-Spanners:

Lemma 2.1. *Der Teilgraph G' = (V, E') ist ein t-Spanner des Graphen G = (V, E), wenn für jede Kante (u, v) ∈ E gilt:* $dist(u, v, G') \leq t \cdot c_{(u,v)}$
(wobei $c_{(u,v)}$ das Gewicht der Kante (u, v) bezeichnet).

Beweis. Da jede Kante (u, v) ∈ E ein Gewicht $c_{(u,v)}$ besitzt, folgt die Behauptung durch Einsetzten in die Definition eines t-Spanners □

Weiterhin bezeichnen wir einen t-Spanner mit „wenigen“ Kanten als *dünnen Spanner*. In den folgenden Kapiteln wollen wir nun auf die Existenz solcher dünnen Spanner eingehen und uns mit der Frage nach der Kantenzahl beschäftigen.

3 Dünne Spanner in ungerichteten Graphen

In diesem Abschnitt beschäftigen wir uns mit dünnen Spannern in ungerichteten Graphen.
Im gesamten Abschnitt ist das Kantengewicht jeder Kante in allen Graphen, die wir betrachten, 1. Die Definition eines t-Spanners wurde bereits unter Defintion 2.6 gegeben. Eine natürliche Frage ist die nach der *Existenz* solcher Spanner und wie wir solche effizient konsturieren können. Als Ergebnisse erhalten wir in diesem Abschnitt obere und untere Schranken bzgl. der Kantenmenge für t-Spanner sowie einen ersten Algorithmus, der aus einem Beweis heraus entsteht.
Als erstes Resultat können wir die „Schwere“ des Existenzproblems klassifizieren. Dazu eine kurze Erinnerung:

Zur Erinnerung:

Defintion 3.1. *Ein Problem, genauer gesagt ein Entscheidungsproblem, gehört zur Klasse NP, falls eine „Ja“-Lösung dieses Problems in polynomialer Laufzeit verifiziert werden kann. Ferner heißt ein Entscheidungsproblem NP-vollständig, falls es in der Klasse NP liegt und jedes andere Problem aus NP durch eine Transformation in polynomialer Laufzeit auf das Entscheidungsproblem reduziert werden kann, d.h. dass alle anderen Probleme aus NP um nicht mehr als einen polynomialen Faktor härter sind als das vorliegende Problem.*

Bemerkung 3.1. *Aus Definition 3 folgt also, dass ein NP-vollständiges Problem mindestens so schwer ist wie jedes andere Problem in NP.*

Das folgende Theorem gibt nun die Klassifizierung des Problems, geeignete t-Spanner zu finden:

Theorem 3.1. *Das Problem, für einen gegebenen Graph G = (V, E) und zwei Zahlen t, $s \leq 1$, $t, s \in \mathbb{Z}$ herauszufinden, ob G einen t-Spanner mit s oder weniger Kanten hat, ist NP-vollständig*

Beweis. Sei Π das Entscheidungsproblem, ob ein Graph G einen t-Spanner mit s oder weniger Kanten besitzt. Wir können für eine Instanz $I, I \in \Pi$, in polynomialer Zeit entscheiden, ob die Antwort „ja“ für I korrekt ist. Denn:
Für eine gegebene Lösung, d.h. für einen Teilgraph G’ = (V, E’), kann durch Summieren der Kantengewichte aller Wege zwischen je zwei Knoten u, v aus V entschieden werden, ob er ein t-Spanner ist. Dabei haben wir $\frac{n(n-1)}{2}$ mögliche Knotenpaarungen. Wenn dabei jeder mögliche Weg schlimmstenfalls über $(n-1)$ Kanten läuft (gerade so, dass noch kein Kreis erzeugt wird), dann müssen insgesamt $\frac{n(n-1)}{2} \cdot (n-1)$ Additionen durchgeführt werden. Damit ist unser Problem in NP.

Um nun zu zeigen, dass das Problem *NP-vollständig* ist, führen wir das *Edge Domination Set*-Problem für bipartite Graphen auf den Fall des t-Spanner-Problems mit t = 2 zurück. Das *Edge Domination Set*-Problem ist *NP-vollständig* (siehe (Gar03)). Alle Fälle mit $t > 2$ sind komplizierter und damit auch *NP-vollständig*.
Sei dazu $H = (X, Y, A)$ ein bipartiter Graph mit den beiden Knotenmengen X und Y, und sei $k \geq 1, k \in \mathbb{Z}$. Dabei ist $X = \{x_i | 1 \leq i \leq n_x\}$ und $Y = \{x_i | 1 \leq i \leq n_y\}$.
Eine *Edge Domination Set (EDS)* für H ist eine Teilmenge $A' \subseteq A$ der Kantenmenge von H derart, dass für jede Kante aus A' eine adjazente Kante aus $A \backslash A'$ existiert. Das Problem ist dann zu entscheiden, ob H eine *EDS* mit maximal k Kanten enhält. Wir führen das Problem, eine *EDS* zu finden, zurück auf das folgende Problem, einen 2-Spanner zu finden.
Dazu konstrurieren wir einen Graph *G(H) = (V, E)* wie folgt:
Für jedes Kantenpaar x_i, x_j aus X (y_i, y_j aus Y) konstruieren wir einen neuen Knoten x_{ij} (y_{ij}). Damit können wir setzen:

$$V = X \cup Y \cup \{x_{ij} | 1 \leq i < j \leq n_x\} \cup \{y_{ij} | 1 \leq i < j \leq n_y\}$$

$$E_{XY} = \left\{(x_i, x_j), (x_i, x_{ij}), (x_j, x_{ij}) | 1 \leq i < j \leq n_x\right\}$$
$$\cup \left\{(y_i, y_j), (y_i, y_{ij}), (y_j, y_{ij}) | 1 \leq i < j \leq n_y\right\}$$

$$E = A \cup E_{XY}$$

Damit haben wir die ursprüngliche Knotenmenge *(X, Y)* um $(\binom{n_x}{2} + \binom{n_y}{2})$ Knoten erweitert. In E_{XY} ist so jeder Knoten $x_i(y_i)$ mit dem entsprechenden Paarknoten $x_{ij}(y_{ij})$ verbunden. Durch Hinzunahme der ursprünglichen Knotenmenge A entsteht so die neue Knotenmenge E und wir haben *G(H)* fertig konstruiert.

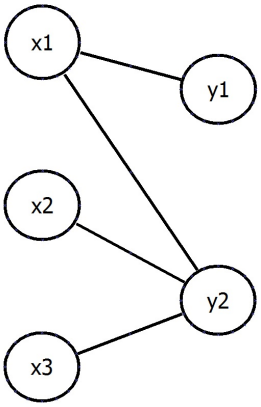

Abbildung 1: Ein Beispielgraph H

Wir setzen

$$s = k + n_x(n_x - 1) + n_y(n_y - 1)$$

Weiter definieren wir jetzt die *normale* Teilmenge von E_{XY} wie folgt:

$$N = \left\{(x_i, x_j), (x_i, x_{ij}) | 1 \leq i < j \leq n_x\right\} \cup \left\{(y_i, y_j), (y_i, y_{ij}) | 1 \leq i < j \leq n_y\right\} \; |N| = n_x(n_x - 1) + n_y(n_y - 1)$$

Damit entspricht s der Summe aus den Kanten in N und den (maximal) k Kanten, die Lösung des *EDS*-Problems von oben sind. Nach Lemma 2.1 ist dann eine *Lösung* für das 2-Spanner-Problem auf $G(H)$ jede Teilmenge $E' \subseteq E$ mit $|E'| \leq s$, welche die Bedingung erfüllt, dass für jede Kante $(u, v) \in E \backslash E'$ ein Knoten $w \in V$ existiert, so dass $(u, w), (w, v) \in E'$ gilt. Denn wenn eine Kante $(u, v) \in E$ existiert, muss nach Lemma 2.1 mit dem Kantengewicht 1 der Weg von u nach v in $E \backslash E'$ maximal die Länge 2 haben, da sonst kein 2-Spanner vorliegt.

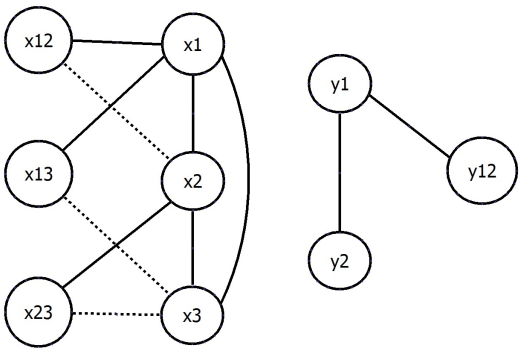

Abbildung 2: Der Graph H mit der Kantenmenge E_{XY}. Durchgezogene Linien stellen die Kanten aus N dar, gestrichelte Linien die aus E_{XY}

Eine *normale Lösung* für das 2-Spanner-Problem auf *G(H)* ist eine Lösung $E' = N \cup A'$, wobei $A' \subseteq A$ gilt. Jetzt können wir beobachten, dass E' genau dann eine normale Lösung für das 2-Spanner-Problem auf *G(H)* ist, wenn A' eine Lösung für das *EDS*-Problem auf H ist. Denn: Wenn $N \cup A'$ die 2-Spanner-Bedingung erfüllt, dann muss für jede Kante aus $N \cup A'$ die oben formulierte Bedingung gelten. Die normale Teilmenge, vereinigt mit A', erfüllt diese Bedingung durch ihre Konstruktion. Damit kann der Weg zwischen je zwei Knoten, die in $N \cup A$ durch eine Kante verbunden waren, in $N \cup A'$ nicht länger als 2 sein, und jede Kante aus $A \backslash A'$ muss somit mit einer Kante in A' adjazent sein. Andererseits, wenn A' eine Lösung für das *EDS*-Problem auf H ist, so muss jede Kante in $A \backslash A'$ adjazent zu einer Kante in A' sein, womit durch die Konstruktion von N der 2-Spanner $N \cup A'$ entsteht.
Bleibt abschließend noch die Beobachtung, dass, wenn $E'' = B \cup A'$ und $B \subseteq E_{XY}$ gilt und $A' \subseteq A$ ein 2-Spanner auf *G(H)* ist, $E' = N \cup A'$ eine normale Lösung auf *G(H)* mit $|N| \leq |B|$ ist. Diese Beobachtung folgt aus der Konstruktion von der Kantenmenge N, die die gewünschten Eigenschaften für den 2-Spanner erfüllt und mit jeder Kante aus A und damit auch aus A' verbunden ist.

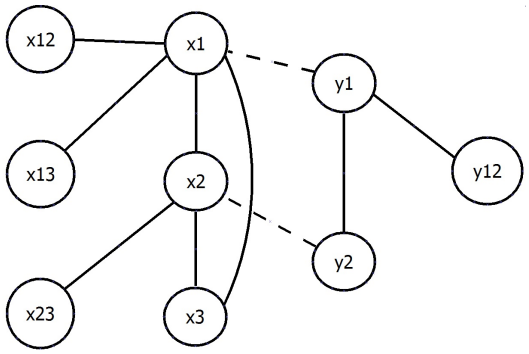

Abbildung 3: Ein normaler 2-Spanner $N \cup A'$ auf H. Durchgezogene Linien stellen die Kanten aus N dar, gestrichelte Linien die aus A'

Nach Voraussetzung ist das EDS-Problem $NP - vollstndig$ (siehe (Gar03)). Damit können wir das Problem, einen 2-Spanner auf *G(H)* zu finden, auf das Problem, eine EDS zu finden, zurückführen, womit dann gemäß Def. das Problem *NP-vollständig* ist. □

Mit Theorem 3.1 haben wir nun die Schwere des Problems klassifiziert. Im Folgenden geben wir jetzt asymptotische Abschätzungen für die Größe von Kantenmengen von t-Spannern. Auch der Frage nach der Existenz und nach bestimmten Eigenschaften von t-Spannern wollen wir weiter nachgehen. Dazu brauchen wir die folgende Definition einer *k-Partition*:

Defintion 3.2. *Eine k-Partition eines Graphen G = (V, E) ist eine Menge von Clustern C_i mit*

$$P = \{(c_i, C_i) | C_i \subseteq V, c_i \in C_i, i \in \{1, \ldots, l\}\}$$

und der Eigenschaften, dass die Cluster eine disjunkte Zerlegung von V bilden (d.h. es gilt: $V = \bigcup_{i=1}^{l} C_i$, $C_i \cap C_j = \emptyset$, $i \neq j$), dass jeder durch C_i erzeugte Teilgraph zusammenhängend ist und dass für alle $u \in C_i$ gilt: $dist(u, c_i) \leq k$.
Weiterhin ist die Dichte $\varrho(P)$ die Anzahl aller Paare C_i, C_j von Clustern, die durch eine Kante verbunden sind.

Die Frage nach der Existenz solcher k-Partitionen, die für die Konstruktion von Spannern benötigt werden, wird in folgendem Lemma beantwortet.

Lemma 3.1. *Für jeden Graphen G mit n Knoten und für alle* $1 < k < n$ *existiert eine (in polynomieller Zeit konstruierbare)* $(\log_k n)$*-Partition mit Dichte* $\varrho(P) \leq (k-1)n$

Beweis. (Für den Beweis folgen wir (Awe85))
Sei dazu $k \in \{2, \ldots, n-1\}$ und $G = (V, E)$ mit $|V| = n$ ein Graph.
Auf G können wir durch einen *Search_for_Leader* Algorithmus (s. dazu (Awe85), Seite 813 ff.) einen sogenannten *Leader-Knoten* c_i in dem Restgraphen finden, der entsteht, wenn die Konten aller konstruierten Cluster aus V und alle adjazenten Kanten aus E entfernt werden.
Ausgehend von c_i starten wir eine BFS-Traversierung (zu BFS s. (CLRS07)) in dem Restgraphen. Für den nächsten Cluster wird eine Knotenmenge C_m erzeugt, in die zunächst der Leader-Knoten c_m eingeht. Jetzt wird jede Traversierungebene mit min. (k mal Anzahl aller Knoten, die schon in C_m enthalten sind) Knoten in C_m eingefügt, und der neue Restgraph, der durch Ausschluss von C_m aus aus dem alten Restgraphen entsteht, wird analog zu oben weiter in Cluster zerlegt, bis $V = \emptyset$ ist.
Damit hat jede Ebene eines Clusters C_i min. k Knoten. Die Anzahl Ebenen eines Clusters C_i ist, C_i als Baum betrachtet, höchstens $dist(u, c_i)$.
Jetzt gilt für die Anzahl der Knoten eines Clusters:

$$|C_i| \geq k^{dist(u,c_i)}$$

Da ebenfalls$|C_i| \leq n$ gelten muss, folgt $\log_k n \geq dist(u, c_i)$.

Ferner ist es noch nötig, die Abschätzung der Dichte $\varrho(P)$ zu zeigen. Hierfür benötigen wir den Begriff einer *Interclusterkante*, d.h. einer Kante, die zwei benachtbarte Cluster C_i und C_j verbindet.
Jetzt ist bei der Konstruktion der Cluster über BFS-Ebenen zu beobachten, dass die erste BFS-Ebene, die nicht mehr zu einem Cluster hinzugefügt wird, höchstens $(k-1)q$ Möglichkeiten für Interclusterkanten zu Clustern späterer Iterationen bietet; wobei q die Anzahl aller Knoten ist, die in dem Cluster schon enthalten sind.
Nach der Konstruktion der $(\log_k n)$-Partition zählen wir die vorhandenen Interclusterkanten durch. Diese verbinden per Definition jeweils ein Clusterpaar. Hierbei zählen wir die abgehenden Interclusterkanten. Aufsummieren über diese liefert insgesamt bei n Knoten die Dichteabschätzung:

$$\varrho(P) \leq (k-1)n$$

□

Damit kommen wir zu einem Theorem, das die Existenzfrage nach einem t-Spanner beantwortet:

Theorem 3.2. *Für jeden Graphen G mit n Knoten und für alle* $1 < k < n$ *existiert ein (in polynomieller Zeit konstruierbarer)* $(4\log_k n)$*-Spanner mit höchstens* kn *Kanten.*

Beweis. Sei $G = (V, E)$ ein Graph. Zuerst konstruieren wir eine Partition P von G wie in Lemma 3.1. Jetzt können wir für jeden Cluster C_i mit $(C_i, c_i) \in P$ einen minimalen Spannbaum mit Wurzel c_i finden, da jede Kante aus C_i nach Definition 2.5 mit jedem Knoten $u \in C_i$ durch einen Weg verbunden ist.
Die Menge E' aller Kanten des t-Spanners, den wir suchen, enthält alle Kanten aller dieser Spannbäume (wobei $|E'| \leq n$ gilt, da wir nur Spannbäume betrachten).
Nun fügen wir für jedes benachbarte Clusterpaar C_i, C_j eine Interclusterkante e_{ij} (s. dazu auch Bew. zu Lemma 3.1) zu E' hinzu. Nach Lemma 3.1 finden wir höchstens $(k-1)n$ solcher Kanten, damit haben wir: $|E'| \leq n + (k-1)n = kn$
Jetzt müssen wir nur noch zeigen, dass der gefundene Teilgraph den Strech-Faktor $4\log_k n + 1$ hat. Dazu nehmen wir eine Kante $(u, v) \in E$. ObdA können wir annehmen, dass u im Cluster C_i liegt und v im Cluster C_j. Für den Radius r, also den Abstand jedes Clusterknotens zu c_i, gilt nach Lemma 3.1: $r \leq \log_k n$. Falls jetzt $i = j$ gilt, so läuft der Pfad von u nach v über c_i und hat höchstens eine Länge von $2r$.
Falls $i \neq j$ gilt, existiert ein Pfad von u nach c_i mit einer maximalen Länge von r, ein Pfad von c_i nach c_j

über e_{ij} mit einer maximalen Länge von $r+1+r$ und ein Pfad von v nach c_j mit einer maximalen Länge von r. Damit hat der Pfad von u nach v höchstens eine Länge von $r+r+1+r+r = 4r+1$, womit die Behauptung bewiesen wäre. □

Mithilfe von Theorem 3.2 können wir nun für $k = 2$ eine einfache O-Abschätzung für die Existenz von t-Spannern geben und deren Eigenschaften geben:

Korollar 3.1. *Für jeden Graphen G mit n Knoten existiert ein (in polynomieller Zeit konstruierbarer) $O(\log n)$-Spanner mit $O(n)$ Kanten*

Beweis. Folgt sofort durch Setzen von $k := 2$ in Theorem 3.2. □

Genauso können wir folgendes Resultat für $k = n^{\frac{1}{r}}, r \geq 1$ festhalten:

Korollar 3.2. *Für jeden Graphen G mit n Knoten existiert ein (in polynomieller Zeit konstruierbarer) $(4r+1)$-Spanner mit $O(n^{\frac{1+l}{r}})$ Kanten*

Beweis. Folgt sofort durch Setzen von $k = n^{\frac{1}{r}}, r \geq 1$ in Theorem 3.2 □

Aus den Beweisen von Theorem 3.2 und von Lemma 3.1 ergibt sich, unter Verwendung der Methoden `Cluster_Creation Procedure`, `Search_for_Leader` und `Link_Election` aus (Awe85), ein Algorithmus zum Berechnen eines $(4\log_k n)$-Spanners auf einem Graphen G (Dieser unterscheidet sich allerdings von dem in Kapitel 5 vorgestellten Algorithmus 5.2).
Im Wesentlichen müssen nur die einzelnen Schritte im Beweis von Theorem 3.2 nachvollzogen werden. Zuerst wird mit der Methode `Cluster_Creation Procedure` eine Partition auf G zu berechnet. Weiter berechnet der Algorithmus zu jedem Cluster einen Spannbaum, z.B. mithilfe des Algorithmus von Kruskal (s. dazu Kapitel 5 oder (CLRS07)). Durch Hinzufügen von Interclusterkanten entsteht so schlußendlich der $(4\log_k n)$-Spanner auf G. Formal sieht der Algorithmus dann wie folgt aus:

Algorithmus 3.1. ***Gegeben:*** *Ein ungerichteter, zusammenhängender Graph $G = (V, E)$ sowie ein positiver, ganzzahliger Parameter t.*

Gesucht: *Ein t-Spanner auf G.*

```
1:  P =Cluster_Creation Procedure(G);
2:  G_Out = ∅;
3:  for i := 1 to |P| do
4:     {Erzeuge Spannbaum auf Cluster i und füge ihn zu G_Out hinzu}
5:     for j := 1 to |P| do
6:        if Cluster i != Cluster j then
7:           {Füge Interclusterkante zwischen Cluster i und Cluster j zu G_Out hinzu}
8:        end if
9:     end for
10: end for
```

Mit Theorem 3.2 bzw. den Korollaren 3.1 und 3.2 haben wir eine obere Schranke für t-Spanner gefunden. Im folgenden letzten Teil von Kapitel 3 werden wir uns mit der Frage nach unteren Schranken beschäftigen.
Zunächst brauchen wir wieder eine Definition:

Defintion 3.3. *Der Umfang g eines Graphen $G = (V, E)$ ist die Länge eines kürzesten Kreises in G. Ist G zyklenfrei, so ist: $g := \infty$.*

Wir benötigen folgendes Lemma, das eine Existenzaussage über Graphen mit bestimmtem Umfang macht:

Lemma 3.2. *Für alle $r \geq 1$ existieren (unendlich viele) Graphen $G = (V, E)$ mit n Knoten und einem Umfang $u \geq r$ sowie einer Kantenzahl $|E| \geq \frac{1}{4}n^{1+\frac{1}{r}}$.*

Beweis. Siehe (Bol78) □

Ferner erhalten wir auch ein Resultat zu t-Spanner auf Graphen mit einem bestimmten Umfang:

Lemma 3.3. *Für alle $t \geq 1$ und $r \geq t + 2$ und für alle Graphen $G = (V, E)$ mit einem Umfang $u \geq r$ ist der einzige t-Spanner von G der Graph selbst.*

Beweis. Angenommen es existiert ein weiterer t-Spanner $G' = (V, E')$, in dem die Kante $(u, v) \in E$ nicht enthalten ist. Damit exisitiert aber in G' ein Weg von u nach v mit der Länge t. Durch Hinzufügen der Kante (u, v) erhalten wir einen Kreis in G mit der Länge $t + 1 < r$. G ist aber vom Umfang $u \geq r$ und wir hätten damit im Widerspruch dazu einen kleineren Kreis in G gefunden. □

Damit kommen wir zum letzten Ergebnis dieses Abschnitts. Wir können nun für unendlich viele Graphen eine untere Schranke bezüglich der Kantenzahl in einem t-Spanner geben.

Theorem 3.3. *Für alle $r \geq 3$ existieren (unendlich viele) Graphen $G = (V, E)$ mit n Knoten, für die jeder $(r-2)$-Spanner $\Omega(n^{1+\frac{1}{r}})$ viele Kanten benötigt.*

Beweis. Gegeben $r \geq 3$. Nach Lemma 3.2 existieren (unendliche viele) Graphen $G = (V, E)$ mit n Knoten und einem Umfang $u \geq r$ und einer Kantenmenge der Größe $|E| \geq \frac{1}{4}n^{1+\frac{1}{r}}$. Mit $t := r-2$ folgt aus Lemma 3.3, dass G der einzige t-Spanner von G ist. Damit gilt für den $(r-2)$-Spanner $\Omega(n^{1+\frac{1}{r}})$ □

4 Dünne Spanner in gerichteten Graphen

In diesem Kapitel beschäftigen wir uns mit dem Fall gerichteter Graphen. Hier ist die Suche nach dünnen Spanner sehr viel schwerer als im ungerichteten Fall, da es hier vorkommen kann, dass gar kein dünner Spanner existiert. Damit ist auch das Problem, einen t-Spanner für gerichtete Graphen zu finden, NP-vollständig. So existieren Graphen mit n Knoten und $\Omega(n^2)$ Kanten, die höchstens einen gerichteten Pfad zwischen zwei Knoten haben (wie z.B. der vollständig bipartite Graph (X, Y, E) mit $|X| = |Y| = \frac{n}{2}$, dessen Kanten alle gerichtet von X nach Y sind. Für diese Graphen ist kein Teilgraph ein t-Spanner (außer dem Graph selbst)).
Das folgende Lemma impliziert nun, dass für jedes feste $t \geq 1$ unendlich viele dichte Graphen mit hohem Kantenzusammenhang existieren, für die der beste t-Spanner der Graph selbst ist. Auch im gesamten folgenden Kapitel ist das Kantengewicht jeder Kante jedes Graphen 1.

Lemma 4.1. *Für alle $t \geq 1$ und alle $n \geq t$ existiert ein gerichteter Graph G mit n Knoten, der Kanten der Größenordnung $\Omega(\frac{n^2}{t^2})$ besitzt und der $O(\frac{n}{t^2})$-kantenzusammenhängend ist und dessen einziger t-Spanner G selbst ist.*

Beweis. Mit t, n gegeben konstruieren wir einen gerichteten Graphen $G = (V, E)$ wie folgt:
Sei $V = \{0, \ldots, n-1\}$ die Knotenmenge (die wir uns für die bessere Vorstellung als Ring angeordnet denken können). Weiterhin setzen wir $p := \left\lfloor \frac{(n-t)}{t^2} \right\rfloor$. Für alle $0 \leq l \leq p$ setzen wir eine Teilkantenmenge E_l:

$$E_l = \{(i, ((i + lt + 1) \bmod n)) | 0 \leq i \leq n-1\}$$

Damit können wir die Kantenmenge E als Vereinigung aller E_l's setzen: $E = \bigcup_{0 \leq l \leq p} E_l$; mit $E \in \Omega(\frac{n^2}{t^2})$.
Ferner gilt für den Kantenzusammenhang von G die Abschätzung $O(\frac{n}{t^2})$. Denn: Entfernen aller maximal p Kanten, die von jedem i ausgehen, zerstört den Zusammenhang in G, und es gilt $p = \left\lfloor \frac{n}{t^2} - \frac{t}{t^2} \right\rfloor = \left\lfloor \frac{n}{t^2} - \frac{1}{t} \right\rfloor$.
Nun können wir fordern, dass jeder t-Spanner von G die komplette Kantenmenge E enthält. Das können wir nun wie folgt einsehen:
Angenommen, wir entfernen eine beliebige Kante $e = (i, i') \in E_l (0 \leq l \leq p)$ aus E und nehmen an, der resultierende Teilgraph $G' = (V, E \setminus \{e\}$ sei nun ein t-Spanner für G. Die Kante e erfüllt nach Konstruktion die Bedingung $i' \equiv (i + lt + 1) \bmod n$. Da nach Annahme G' ein t-Spanner vpn G ist, existiert ein Weg $e_1, \ldots, e_m$, $2 \leq m \leq t$ von i nach i' in G', wobei i Startknoten von e_1 und i' Endknoten von e_m ist. Jetzt können wir weiter annehmen, dass $e_j \in E_{l_j}$ ist für alle $1 \leq j \leq m$, also dass jeder Knoten des Weges von i nach i' in einer der Teilkantenmengen E_l liegt, wie wir sie oben definiert haben. Dieser Definition folgend, können wir nun bestimmen, wohin der Weg, der in e_1 mit i beginnt, führt:

$$\begin{aligned}
e_1 &= (i, (i + l_1 t + 1) \bmod n) \\
e_2 &= (((i + l_1 t + 1) \bmod n), (((i + l_1 t + 1) \bmod n) + l_2 t + 1) \bmod n) \\
&\ldots \\
e_m &= ((i + \sum_{1 \leq j \leq m-1} (l_j t + 1) \bmod n), (i + \sum_{1 \leq j \leq m} (l_j t + 1) \bmod n)
\end{aligned}$$

Also können wir den Endkonten i'' dieses Weges angeben:

$$i'' = (i + \sum_{1 \leq j \leq m} (l_j t + 1) \bmod n) = ((i + m + t \sum_{1 \leq j \leq m} l_j) \bmod n)$$

Jetzt ist wichtig zu sehen, dass, selbst wenn der Weg nur Kanten aus E_p enthält, die maximale Entfernung, die durchlaufen wird, $t + t^2 p \leq n$ (*) ist.

Denn $m \leq t$ und $t \sum_{1 \leq j \leq m} l_j \leq t \sum_{1 \leq j \leq t} p = t^2 p$ und da für i'' obiger Zusammenhang gilt. Dabei ist zu bemerken, dass es somit nicht möglich ist, eine komplette Runde in dem Knotenring zu laufen, denn eine solche erfordert $n + 1$ Kanten.

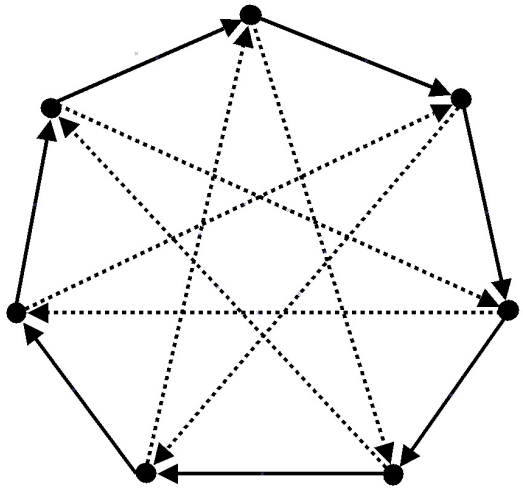

Abbildung 4: Ein Beispielgraph mit n = 7 und t = 2. Die Kanten von E_0 sind durchgehend, gestrichelte Kanten sind die Kanten aus E_1

Weiterhin muss $i' = i''$ sein, damit muss mit (*) gelten:

$$m + t \sum_{1 \leq j \leq m} l_j = 1 + tl$$

Damit ergibt sich: $l = \frac{m + t \sum_{1 \leq j \leq m} l_j - 1}{t}$

Das führt zu einem Widerspruch, denn $m \geq 2$ und $0 \leq l \leq p$, womit die Behauptung gezeigt wäre. □

Theorem 4.1. *Für alle $t \geq 1$ existieren unendlich viele Graphen $G = (V, E)$ mit n Knoten, für die alle t-Spanner $\Omega(\frac{n^2}{t^2})$ Kanten benötigen.*

Beweis. Klar, denn ausgehend von Lemma 4.1 ist $n \geq t$, und damit ist die Behauptung bewiesen. □

5 Ein Algorithmus

Eine wichtige Frage im Bezug auf t-Spanner ist nicht nur die nach der Existenz oder nach unteren/oberen Schranken für die Knotenzahlen, sondern auch die nach einem konkreten Algorithmus, um einen t-Spanner zu finden. Im Folgenden wird ein solcher Algorithmus präsentiert. Im Wesentlichen ist dieser nur eine Abwandlung des Algorithmus von Kruskal zum Bestimmen minimaler Spannbäume (s. auch hier (CLRS07)). Allerdings ist in der Definition des t-Spanners nicht gefordert, dass dieser keine Kreise enthält - im Gegensatz zu einem minimalen Spannbaum. Der Algorithmus baut auf genau diesem Argument auf.
Konkret betrachten wir im Verlauf dieses Kapitels ungerichtete, zusammenhängende Graphen. Im Gegensatz zu den Kapiteln 3 und 4 sind hier allerdings positive Kantenbewertungen erlaubt; wir beschränken uns also nicht auf ein Kantengewicht von 1.
Zur Erinnerung der Algorithmus von Kruskal zum Bestimmen eines minimalen Spannbaumes:

Algorithmus 5.1. ***Gegeben:*** *Ein ungerichteter, zusammenhängender Graph $G = (V, E)$ sowie eine Kostenfunktion $c : E \to \mathbb{R}_0^+$, die jeder Kante ein (positives) Kantengewicht zuordnet.*

Gesucht: *Ein minimaler Spannbaum $B = (V, T)$, wobei $T \subseteq E$ ist.*

1: { *Sortiere alle Kanten $e_i, 1 \leq i \leq |E|$ nach aufsteigendem Kantengewicht (Also so dass gilt: $c(e_k) \leq c(e_{k+1})$ für $1 \leq k \leq |E| - 1$)*}
2: $T := \emptyset$;
3: ***for*** $i := 1$ *to* $|E|$ ***do***
4: ***if*** $(V, T \cup \{e_i\})$ *enthält keinen Kreis* ***then***
5: $T := T \cup \{e_i\}$;
6: ***end if***
7: ***end for***

Kruskals Algorithmus bildet jetzt die Basis für den Algorithmus zur Bestimmung eines t-Spanners für ungerichtete Graphen. Im Beweis zu Theorem 3.2 arbeiten wir mit Spannbäumen und Interclusterkanten. Es bietet sich also an, hier ebenfalls Spannbäume als Grundlage für einen Algorithmus zu verwenden. Dabei seien $u(e), v(e)$ für $e \in E$ die beiden Knoten, die durch die Kante e verbunden sind.
Damit kommen wir zum Algorithmus zur Bestimmung eines t-Spanners:

Algorithmus 5.2. ***Gegeben:*** *Ein ungerichteter, zusammenhängender Graph $G = (V, E)$ sowie eine Kostenfunktion $c : E \to \mathbb{R}_0^+$, die jeder Kante ein (positives) Kantengewicht zuordnet; sowie ein Parameter $t \in \mathbb{Z}^+$*

Gesucht: *Ein t-Spanner $B = (V, T)$, wobei $T \subseteq E$ ist.*

1: { *Sortiere alle Kanten $e_i, 1 \leq i \leq |E|$ nach aufsteigendem Kantengewicht (Also so dass gilt: $c(e_k) \leq c(e_{k+1})$ für $1 \leq k \leq |E| - 1$)*}
2: $T := \emptyset$;
3: ***for*** $i := 1$ *to* $|E|$ ***do***
4: ***if*** $B := (V, T \cup \{e_i\})$ *enthält keinen Kreis* ***then***
5: $T := T \cup \{e_i\}$;
6: ***else***
7: ***if*** $dist(u(e_i), v(e_i), B) \geq t \cdot c(e_i)$ ***then***
8: $T := T \cup \{e_i\}$;
9: ***end if***
10: ***end if***
11: ***end for***

Algorithmus 5.2 unterscheidet sich von dem Algorithmus von Kruskal im Wesentlichen dadurch, dass Kreise zugelassen sind. Die Kanten, die einen Kreis schließen, werden nur zu dem Teilgraphen B hinzugefügt, wenn der Abstand der beiden Knoten $u(e)$ und $v(e)$, zwischen denen die Kanten jeweils verlaufen, größer als t mal das Kantengewicht ist (Zeile 7 bis 9).
Wenn wir nun im Verlauf der Iterationen des Algorithmus eine Kante finden, die einen Kreis schließt, so ist durch die Vorsortierung in Zeile 1 garantiert, dass der Abstand aller Knoten, die auf dem Kreis liegen, jeweils zueinander minimal ist. Davon ausgenommen sind nur die beiden Start-/Endknoten des Kreises. Hier wird jetzt die t-Spanner Bedingung überprüft und falls der Abstand dieser beiden Knoten im Teilgraphen B zu groß ist (also die t-Spanner Bedingung nicht mehr erfüllt), so wird der Kreis geschlossen. Dadurch wird die t-Spanner Eigenschaft für den Teilgraphen B garantiert; denn der in B enthaltene minimale Spannbaum erfüllt für alle „inneren“ Knoten (d.h. die Knoten, die einen Knotengrad größer 1 haben) per Definition die t-Spanner Bedingung und für alle „Randknoten“ (d.h. alle Konten mit Knotengrad 1) wird die t-Spanner Bedingung durch die Abfrage in Zeile 7 gesichert.

Satz 5.1. *Der Algorithmus 5.2 kommt mit $O(|E|(\log|E| + |V|^2))$ Zeitschritten aus*

Beweis. Nach (WG00) kommt das Sortieren der Kanten mit $O(|E|\log|E|)$ Zeitschritten aus, und auch die Berechnung von T benötigt nicht mehr als $O(|E|)$ Zeitschritte. Für die Kreisüberprüfung genügen ebenfalls $O(\log|E|)$ Zeitschritte, siehe hier ebenfalls (WG00) (Beweis zu Satz 1.14).
Für die Überprüfung der Distanz im Teilgraphen B in Zeile 7 von Algorithmus 5.2 kann Dijkstra's kürzeste Wege Algorithmus verwendet werden. Dieser kommt, nach (WG00), Satz 1.17, mit $O(|V|^2)$ Zeitschritten aus. Da wir in der Schleife (Zeile 3 bis 10) alle $|E|$ Knoten durchlaufen, kommen wir für die Zeilen 3 bis 11 auf einen Gesamtaufwand von $O(|E|(\log|E| + |V|^2))$. Durch Zusammenfassen ergibt sich der Gesamtzeitaufwand von Algorithmus 5.2. □

Der Algorithmus 5.2 gehört zu den *Greedy*-Algorithmen, d.h. er entscheidet sich in jeder Iteration für den aktuell besten Schritt. Hier bedeutet das, die Kante mit dem (aktuell) kleinsten Kantengewicht zu verwenden.

Wichtig für die Anwendung dieses Algorithmus ist auch die Ausdünnung des zugrunde liegenden Graphen beim Erstellen eines t-Spanners. Für Graphen mit wenigen Kreisen ist die Wahrscheinlichkeit hoch, den kompletten Graphen wieder als t-Spanner zu erhalten, vor allem, wenn die Kantengewichte die gleiche Größenordnung besitzen.

6 Anwendungsbeispiel

t-Spanner haben in der Konstruktion oder Verbesserung von Netzwerken praktische Verwendung. Ein Anwendungsbeispiel, dass sich im Rahmen eines Projektseminars im Sommersemester 2010 mit dem Titel „Baustellenplanung mit der Deutschen Bahn AG“ ergeben hat, wollen wir jetzt im Folgenden betrachten. Dabei geht es um die Suche nach möglichen Neubaustrecken im Schienennetz der DB. Dieses besteht aus Bahnhöfen und zweigleisigen Strecken, die zwischen Bahnhöfen verlaufen. Natürlicherweise hat jede Strecke eine bestimmte Länge.
Wir können jetzt mithilfe von t-Spannern Vorschläge erarbeiten, zwischen welchen Bahnhöfen Neubaustrecken sinnvoll erscheinen. Dabei vergleichen wir im Wesentlichen ein gegebenes Schienennetz mit einem t-Spanner, und anhand der Differenz der Wege können dann Vorschläge für sinnvolle Strecken gegeben werden.
Dazu modellieren wir zuerst aus den gegebenen Daten das Schienennetz:

6.1 Modell

Vorgegeben ist eine modellhafte Testinstanz mit 19 Bahnhöfen und 34 Strecken. Jede Strecke liegt genau zwischen zwei Bahnhöfen, wobei teilweise zwischen zwei Bahnhöfen mehr als eine Strecke verlaufen kann. Das ist bedingt duch den Umstand, das neben reinen Personenverkehrsstrecken auch Güterverkehrs- und Mischstrecken existieren. Allerdings spielt dieser Fakt für die weitere Betrachtung keine Rolle.
In Tabelle 3 bzw. in Tabelle 4 sind die Bahnhöfe mit ID und Namen sowie die Strecken mit ID und Entfernung zu finden. Die Entfernungdaten stammen aus dem frei zugänglichen Programm „Google Earth“ und von „www.postleitzahlen.org“ und sind zunächst reine Luftlinienentfernungen zwischen den jeweiligen Orten. Dabei diente für das Ruhrgebiet die Stadt Essen als Referenz.

Damit können wir zu dem Modell kommen: Aus den gegebenen Daten ergibt sich ein ungerichteter, zusammenhängender, gewichteter Graph $G_M = (V_M, E_M)$, wobei $|V| = 19$ und $|E| = 28$ ist. Dabei fehlen genau die sechs Strecken, die in der Testinstanz zwischen denselben Bahnhöfen verlaufen. Diese Doppelung entsteht dadurch, dass hier von unterschiedlichen Streckenklassen ausgegangen wird, was in unserem Modell aber nicht weiter berücksichtigt wird. Die Kantengewichte entsprechen den Entfernungsangaben in km, multipliziert mit dem Faktor 1.3. Dieser Faktor spiegelt die durchschnittlichen Entfernungserhöhung durch die Streckenführung wider. Ferner soll auch $V = \{1, 2, \ldots, 20\} \setminus \{10\}$ und $E = \{1, 2, \ldots, 34\}$ sein; d.h. wir verwenden die gegebenen ID's als Knoten- bzw. Kantennummer. Dabei fällt die Knotennummer 10 aus der Knotenmenge heraus, da dieser Knoten (Basel) in der Testinstanz isoliert ist und nicht weiter betrachtet wird. Auch in Tabelle 3 ist der Knoten mit der ID 10 nicht mehr zu finden.

Weiterhin konstruieren wir analog zu oben aus Tabelle 5 einen ungerichteten, gewichteten Graphen $G_V = (V_V, E_V)$ mit $|V| = 19$ und $|E| = \binom{19}{2} = 171$. G_V ist damit ein vollständiger Graph auf den gegebenen Bahnhöfen.

Im weiteren Verlauf konstruieren wir auf G_V einen t-Spanner; wobei $t = 1.5$ gelten soll. Doch zunächst kommen wir zur Implementierung:

6.2 Implementierung

Um nun für das Modell eine Lösung zu finden, benötigen wir auf G_V einen t-Spanner. Um einen solchen konstruieren zu können, wird der Algorithmus 5.2 in Java implementiert. Dazu verwenden wir „Eclipse IDE for Java Developers, Build id: 20100218-1602“ und das Package „JGraphT 0.8.1“, das unter der GNU Lesser General Public License frei zur Verfügung steht und u.a. eine Implementierung von *Dijkstra*'s

kürzeste Wege Algorithmus (s.(CLRS07)) enthält.
Damit sieht Algorithmus 5.2 wie folgt aus:

Listing 1: calcTSpanner

```
/**
* Berechnet einen t-Spanner nach dem abgewandelten Kruskal-Algorithmus
* @param Graph aus dem Objekt MyGraph
* @param t
* @return t-Spanner
*/
public MyGraph calcTSpanner(MyGraph Graph, double t){

//Holt die Kantenliste des Graphen
ArrayList<MyEdge> kantenListe = (ArrayList<MyEdge>) Graph.getAllEdges();

//Sortiert die Kantenliste nach Entfernung
Collections.sort(kantenListe, new EdgeComparator());

MyGraph tempGraph = new MyGraph();
 for (Iterator<MyEdge> e = kantenListe.iterator(); e.hasNext(); ){
        MyEdge edge = e.next();

        //Falls ein Kreis entsteht...
        if(circle(new MyGraph(tempGraph.getAllEdgesSet()), edge)){

         //Falls der Knotenabstand der Kreisknoten zu groß ist...
         if(distance(edge.getStart(), edge.getEnd(),
                        new MyGraph(tempGraph.getAllEdgesSet()))
                        > t * edge.getDistance())
                        {
                        tempGraph.addEdge(edge);
                        }
        }
         if(!circle(new MyGraph(tempGraph.getAllEdgesSet()), edge)) {
                        tempGraph.addEdge(edge);
        }
 }
return tempGraph;
}
```

Anmerkung:
In der obigen Implementierung wird der Graph als *MyGraph*-Objekt und die Kanten als *MyEdge*-Objekte verwendet. Für deren Implementierung siehe Listing 2 bzw. Listing 3 im Anhang.
In *calcTSpanner* wird überprüft, ob der Graph zusammen mit der „aktuellen“ Kante einen Kreis bildet (durch Aufruf der Methode *circle*), und, wenn ja, ob die Abstandsbedingung erfüllt ist. Entsprechend werden die Kanten hinzugefügt oder entfernt. Abstände werden mit der Methode *distance* bestimmt.
Um zu bestimmen, ob ein ein Kreis durch Hinzufügen einer Kante entsteht, verwenden wir in der Methode *circle* den *DFS*-Algorithmus (s. (CLRS07)). Dieser kann in $O(\max(|V|,|E|))$ Zeitschritten ermitteln, ob in einem Graphen ein Kreis vorliegt oder nicht. Dazu muss *DFS* lediglich prüfen, ob ein Nachbarknoten schon besucht ist oder nicht, denn sobald *DFS* einen Knoten zum zweiten Mal besucht, exisistiert ein Kreis.

Damit steht nun der Algorithmus zur Bestimmung eines t-Spanners zur Verfügung. In der weiteren

Implementierung stellen wir jetzt noch Methoden zur Verfügung, die auf einem Graphen die Länge des kürzesten Weges mithilfe der *Dijkstra*-Methode aus dem *jGrapht* Paket berechnen und die zwei Listen von *Pair*-Objekten (siehe Listing 4) anhand des Abstandes zwischen den beiden Knoten, die ein *Pair* ausmachen, vergleichen können. Für die genau Implementierung siehe *CalcDistance* (Listing 5) bzw. *CompareWays* (Listing 6).

6.3 Lösung

Nach der Modellierung und Implementierung der benötigten Klassen und Methoden steht nun alles Nötige zur Verfügung, um eine Lösung des Problems, geeignete Vorschläge für Neubaustrecken zu geben, finden zu können.
Nach dem Einlesen der Daten erstellen wir mit 1 einen 1.5-Spanner auf dem Graphen G_V. Nach Korollar 3.1 hat G_V mit $|V| = 19$ einen $O(1.2788)$-Spanner mit $O(19)$ Kanten, falls alle Kantengewichte 1 sind.
In der Tat hat G_V, auch wenn die Kantengewichte hier verschieden von 1 sind, von den ursprünglichen 171 Knoten nur noch 30, ist also im Vergleich zu G_V dünn.
Weiter berechnen wir jetzt im 1.5-Spanner und in G_M für alle $\binom{19}{2} = 171$ Kombinationen von je zwei Bahnhöfen die Länge des kürzesten Weges zwischen den beiden (mithilfe von *CalcDistance*). Abschließend vergleichen wir mit *CompareWays* die jeweiligen Entfernungen aller Bahnhof-Kombinationen im 1.5-Spanner mit denen in G_M und geben am Schluss in absteigender Reihenfolge die Paare mit der größten Differenz der Distanzen zusammen mit dem kürzesten Weg im 1.5-Spanner aus. Das Ergebnis ist in Tabelle 8 zu finden.
Dabei ist der Weg als Folge von Kanten angegeben; d.h. das hier die Knotenpaare, zwischen denen eine Kante verläuft, als ganze Zahlen zwischen 0 und 19 angegeben sind. Aufgrund des Datensatzes der Testinstanz kommt der Knoten 9 (externe ID 10); Basel; hier nicht vor. Wegen der Fülle der Daten wird hier nur eine in Ganzzahlen codierte Form der Wege angegeben.
Bspw. bedeutet die erste Zeile von Tabelle 8 im Anhang, [Stuttgart Passau 261.82 [(16 : 17), (16 : 18)], dass der Weg von Stuttgart nach Passau in G_M 261.82 km länger ist als der entsprechende Weg im 1.5-Spanner. Es empfiehlt sich daher, die Strecke München-Passau (16 : 18) zu bauen, da die Strecke Stuttgart-München (16 : 17) schon in G_M enthalten ist. Damit können ebenfalls 261.82 km auf dem Weg von München nach Passau eingespart werden, vgl. zweite Zeile von Tabelle 8, [München Passau 261.82 [(16 : 18)] .
Zusammengenommen ergeben sich aus allen Einträgen in Tabelle 8 im Anhang mit einer positiven Distanz die in Tabelle 1 stehenden neun Vorschläge für Neubaustrecken.

Tabelle 1: Neubaustrecken

Kante	Start - Ende	Länge · 1.3
(16 : 18)	München - Passau	192.91
(13 : 15)	Fulda - Leipzig	274.05
(5: 15)	Hannover - Leipzig	278.46
(15 : 18)	Leipzig - Passau	415.13
(12 : 17)	Stuttgart - Würzburg	163.79
(5 : 7)	Hannover - Bremen	135.06
(2 : 4)	Rostock - Flensburg	248.52
(7 : 8)	Bremen - Rotterdam	417.00
(10 : 12)	Mannheim - Würzburg	144.39

Wenn wir jetzt den G_M, also das Streckennetz der Testinstanz, die wir erhalten haben, um diese neun Strecken erweitern, können wir insgesamt 6887.88 km im Streckennetz einsparen. Es genügt, diese neun

Strecken zu bauen, um für alle 53 Wege, die in Tabelle 8 eine positive Differenz zum t-Spanner haben, diese Differenz auf null zu bringen.
Damit haben wir, indem wir uns nur auf die Luftlinienentfernungen gestützt haben, mithilfe eines t-Spanners sinnvolle Vorschläge für Neubaustrecken erarbeiten können. Über die Möglichkeiten, die sich noch eröffnen könnten, gibt der nächste Abschnitt Auskunft.

6.4 Ausblick

In Kapitel 6.3 haben wir das Problem, geeignete Vorschläge für Neubaustrecken zu machen, mithilfe von Luftlinienentfernungen und dem Durchschnittsfaktor 1.3 gelöst. Dabei haben wir einen 1.5-Spanner verwendet. Jetzt stellt sich natürlicherweise die Frage, ob und wie sich hier noch Verbesserungsmöglichkeiten bieten.
Zunächst besteht die Möglichkeit, t zu ändern und so einen anderen t-Spanner zu erzeugen. Eng mit der Wahl von t verknüpft ist die „Umwegigkeit“ im Schienennetz. So bedeutet ein Faktor von 1.5, das im 1.5-Spanner jeder Weg maximal die 1.5-fache Länge im Vergleich zu dem direkten Weg hat (da wir bei G_V von einem vollständigen Graphen ausgegangen sind).
Bei $t = 1.2$ hat der t-Spanner 41 Kanten und es ergeben sich bei analogem Vorgehen die in Tabelle 2 angegebenen 15 Neubaustrecken.

Tabelle 2: Neubaustrecken

Kante	Start - Ende	Länge · 1.3
(3 : 18)	Cottbus - Passau	359.11
(16 : 18)	München - Passau	192.91
(12 : 15)	Würzburg - Leipzig	246.32
(12 : 17)	Würzburg - Stuttgart	163.79
(13 : 15)	Fulda - Leipzig	274.05
(6 : 13)	Ruhrgebiet - Fulda	210.70
(14 : 17)	Nürnberg - Stuttgart	156.72
(5 : 15)	Hannover - Leipzig	278.46
(5 : 7)	Hannover - Bremen	135.06
(15 : 18)	Leipzig - Passau	415.13
(13 : 19)	Fulda - Köln	195.84
(10 : 12)	Mannheim - Würzburg	144.39
(4 : 7)	Flensburg - Bremen	190.95
(7 : 8)	Bremen - Rotterdam	417.00
(2 : 4)	Rostock - Flensburg	248.52

Ferner stellt sich die Frage, inwieweit die Qualität eines Streckennetzes alleine durch die Streckenkilometer bestimmt werden kann. Andere Kantenbewertungen als alleine die Luftlinienentfernung stellen hier Möglichkeiten zur Verbesserung dar. So könnten z.B. geografische Faktoren in die Kantenbewertungen einfließen, so lassen sich bspw. Tunnel in bergigem Gebiet schwerer bauen als eine Strecke im Flachland.
Hier würde eine genauere Untersuchung der geografischen Gegebenheiten (in diesem Fall) Deutschlands für jede Verbindung in dem, dem t-Spanner zugrunde liegenden vollständigen Graphen G_V, besser abgestimmte Entfernungen ergeben. In obigem Beispiel diente als grobe Näherung dafür der Faktor 1.3, der für jede Strecke eine durchschnittliche Erhöhung der km im Vergleich zur Luftlinie bedeutete.
Auch regionale, (gesellschafts)politische und wirtschaftliche Aspekte können in die Kantenbewertung einbezogen werden. Jede neue Bahntrasse bedeutet einerseits eine regionale Verbesserung der Infra-

struktur für Städte und Gemeinden sowie Unternehmen; andereseits bietet sie auch Konfliktpotential im Hinblick z.B. auf Lärmbelästigung oder Naturschutz.
Alle diese Möglichkeiten, die Kantenbewertung ausgewogener zu machen, können eine genauere Untersuchung der in Kapitel 6 eingangs gestellten Frage ermöglichen.

Es besteht auch noch die Möglichkeit, hinsichtlich verschiedener Strecken- und Netzklassen wie Güter- oder Personennetz zu unterscheiden und für die einzelnen Klassen getrennt Neubauvorschläge zu finden. Eine Aufteilung in z.B. Güter- und Personengraph im Modell würde auch hier neue Möglichkeiten eröffnen.

Insgesamt lässt sich damit feststellen, dass die oben genannte Lösung für das Neubaustreckenproblem im Netz der Deutschen Bahn AG noch Verbesserungspotential in verschiedene Richtungen hat, als Beispiel aber für die Anwendung von t-Spannern in ungerichteten Graphen gut geeignet ist. Die Möglichkeiten, die Lösung zu verbessern, zeigen auf, wie hier mithilfe von t-Spannern noch vieles möglich ist, was man in der Praxis umzusetzten kann und wie man ein Werkzeug in der Hand hat, mit dem gute Lösungen für solche (und andere) Probleme gefunden werden können.

7 Abkürzungsverzeichnis

bsp.	beispielsweise
bzw.	beziehungsweise
Def.	Definition
DB	Deutsche Bahn AG
d.h.	das heißt
et al.	et alii - und andere
ff.	fortfolgend
ID	Identifikationsnummer
min.	mindestens
s.	siehe
sog.	sogenannte
u.a.	unter anderem
vgl.	vergleiche
z.B.	zum Beispiel

$\mathbb{R}$	Die Menge der reellen Zahlen
$\mathbb{Z}$	Die Menge der ganzen Zahlen

8 Anhang

Tabelle 3: Bahnhöfe

KnotenID	Name
1	Hamburg
2	Berlin
3	Rostock
4	Cottbus
5	Flensburg
6	Hannover
7	Ruhrgebiet
8	Bremen
9	Rotterdam
11	Mannheim
12	Frankfurt/Main
13	Würzburg
14	Fulda
15	Nürnberg
16	Leipzig
17	München
18	Stuttgart
19	Passau
20	Köln

Tabelle 4: Streckennetz

StreckenID	Entfernung	Startbahnhof	Endbahnhof
1	254.7	Hamburg	Berlin
2	152.07	Hamburg	Rostock
3	142.01	Hamburg	Flensburg
4	132.94	Hamburg	Hannover
5	94.6	Hamburg	Bremen
6	193.46	Berlin	Rostock
7	104.93	Berlin	Cottbus
8	250.19	Berlin	Hannover
9	145.63	Berlin	Leipzig
10	141.93	Cottbus	Leipzig
11	211.39	Hannover	Ruhrgebiet
12	202.04	Hannover	Fulda
13	202.04	Hannover	Fulda
14	220.12	Ruhrgebiet	Bremen
15	183.17	Ruhrgebiet	Rotterdam
16	57.64	Ruhrgebiet	Köln
17	57.64	Ruhrgebiet	Köln
18	71.61	Mannheim	Frankfurt/Main
19	272.51	Mannheim	München

Fortsetzung nächste Seite

Fortsetzung von letzter Seite			
StreckenID	Entfernung	Startbahnhof	Endbahnhof
20	95.77	Mannheim	Stuttgart
21	193.47	Mannheim	Köln
22	96.80	Frankfurt/Main	Würzburg
23	84.81	Frankfurt/Main	Fulda
24	152.50	Frankfurt/Main	Köln
25	86.90	Würzburg	Fulda
26	86.90	Würzburg	Fulda
27	89.90	Würzburg	Nürnberg
28	89.90	Würzburg	Nürnberg
29	232.83	Nürnberg	Leipzig
30	232.83	Nürnberg	Leipzig
31	149.16	Nürnberg	München
32	149.16	Nürnberg	München
33	200.63	Nürnberg	Passau
34	189.79	München	Stuttgart

Tabelle 5: Vollständige Entfernungen

StreckenID	Entfernung	Startbahnhof	Endbahnhof
1	254.7	Hamburg	Berlin
2	152.07	Hamburg	Rostock
3	353.09	Hamburg	Cottbus
4	142.01	Hamburg	Flensburg
5	132.94	Hamburg	Hannover
6	308.67	Hamburg	Ruhrgebiet
7	94.6	Hamburg	Bremen
8	412.01	Hamburg	Rotterdam
9	463.94	Hamburg	Mannheim
10	392.32	Hamburg	Frankfurt/Main
11	418.36	Hamburg	Würzburg
12	334.36	Hamburg	Fulda
13	462.08	Hamburg	Nürnberg
14	291.54	Hamburg	Leipzig
15	610.61	Hamburg	München
16	534.91	Hamburg	Stuttgart
17	603.35	Hamburg	Passau
18	357.4	Hamburg	Köln
19	193.46	Berlin	Rostock
20	104.93	Berlin	Cottbus
21	363.03	Berlin	Flensburg
22	250.19	Berlin	Hannover
23	452.79	Berlin	Ruhrgebiet
24	317.27	Berlin	Bremen
25	613.33	Berlin	Rotterdam
			Fortsetzung nächste Seite

Fortsetzung von letzter Seite

StreckenID	Entfernung	Startbahnhof	Endbahnhof
26	482.14	Berlin	Mannheim
27	422.55	Berlin	Frankfurt/Main
28	387.78	Berlin	Würzburg
29	338.31	Berlin	Fulda
30	378.23	Berlin	Nürnberg
31	145.63	Berlin	Leipzig
32	502.34	Berlin	München
33	511.95	Berlin	Stuttgart
34	437.41	Berlin	Passau
35	477.67	Berlin	Köln
36	296.79	Rostock	Cottbus
37	191.17	Rostock	Flensburg
38	249.60	Rostock	Hannover
39	451.73	Rostock	Ruhrgebiet
40	246.25	Rostock	Bremen
41	561.54	Rostock	Rotterdam
42	569.67	Rostock	Mannheim
43	499.93	Rostock	Frankfurt/Main
44	500.76	Rostock	Würzburg
45	427.10	Rostock	Fulda
46	520.61	Rostock	Nürnberg
47	302.54	Rostock	Leipzig
48	660.86	Rostock	München
49	625.42	Rostock	Stuttgart
50	618.69	Rostock	Passau
51	495.48	Rostock	Köln
52	467.40	Cottbus	Flensburg
53	322.49	Cottbus	Hannover
54	506.27	Cottbus	Ruhrgebiet
55	404.74	Cottbus	Bremen
56	678.21	Cottbus	Rotterdam
57	485.11	Cottbus	Mannheim
58	436.16	Cottbus	Frankfurt/Main
59	379.38	Cottbus	Würzburg
60	352.36	Cottbus	Fulda
61	345.72	Cottbus	Nürnberg
62	141.93	Cottbus	Leipzig
63	447.62	Cottbus	München
64	495.03	Cottbus	Stuttgart
65	359.11	Cottbus	Passau
66	520.82	Cottbus	Köln
67	269.36	Flensburg	Hannover
68	404.16	Flensburg	Ruhrgebiet
69	190.95	Flensburg	Bremen
70	458.03	Flensburg	Rotterdam

Fortsetzung nächste Seite

Fortsetzung von letzter Seite

StreckenID	Entfernung	Startbahnhof	Endbahnhof
71	592.46	Flensburg	Mannheim
72	521.38	Flensburg	Frankfurt/Main
73	556.52	Flensburg	Würzburg
74	470.98	Flensburg	Fulda
75	603.65	Flensburg	Nürnberg
76	427.89	Flensburg	Leipzig
77	752.38	Flensburg	München
78	669.24	Flensburg	Stuttgart
79	743.49	Flensburg	Passau
80	459.35	Flensburg	Köln
81	211.39	Hannover	Ruhrgebiet
82	103.85	Hannover	Bremen
83	363.35	Hannover	Rotterdam
84	331.86	Hannover	Mannheim
85	260.32	Hannover	Frankfurt/Main
86	287.17	Hannover	Würzburg
87	202.04	Hannover	Fulda
88	338.10	Hannover	Nürnberg
89	214.20	Hannover	Leipzig
90	487.25	Hannover	München
91	402.07	Hannover	Stuttgart
92	497.87	Hannover	Passau
93	248.64	Hannover	Köln
94	220.12	Ruhrgebiet	Bremen
95	183.17	Ruhrgebiet	Rotterdam
96	240.99	Ruhrgebiet	Mannheim
97	188.96	Ruhrgebiet	Frankfurt/Main
98	276.70	Ruhrgebiet	Würzburg
99	210.70	Ruhrgebiet	Fulda
100	363.04	Ruhrgebiet	Nürnberg
101	372.26	Ruhrgebiet	Leipzig
102	491.82	Ruhrgebiet	München
103	335.97	Ruhrgebiet	Stuttgart
104	561.00	Ruhrgebiet	Passau
105	57.64	Ruhrgebiet	Köln
106	320.77	Bremen	Rotterdam
107	403.25	Bremen	Mannheim
108	332.84	Bremen	Frankfurt/Main
109	377.90	Bremen	Würzburg
110	291.02	Bremen	Fulda
111	437.06	Bremen	Nürnberg
112	312.52	Bremen	Leipzig
113	585.69	Bremen	München
114	483.70	Bremen	Stuttgart
115	601.59	Bremen	Passau

Fortsetzung nächste Seite

Fortsetzung von letzter Seite

StreckenID	Entfernung	Startbahnhof	Endbahnhof
116	272.47	Bremen	Köln
117	391.35	Rotterdam	Mannheim
118	357.02	Rotterdam	Frankfurt/Main
119	450.65	Rotterdam	Würzburg
120	393.49	Rotterdam	Fulda
121	541.14	Rotterdam	Nürnberg
122	549.97	Rotterdam	Leipzig
123	659.48	Rotterdam	München
124	483.78	Rotterdam	Stuttgart
125	740.23	Rotterdam	Passau
126	204.46	Rotterdam	Köln
127	71.61	Mannheim	Frankfurt/Main
128	111.07	Mannheim	Würzburg
129	145.98	Mannheim	Fulda
130	188.19	Mannheim	Nürnberg
131	347.31	Mannheim	Leipzig
132	272.51	Mannheim	München
133	95.77	Mannheim	Stuttgart
134	379.35	Mannheim	Passau
135	193.47	Mannheim	Köln
136	96.80	Frankfurt/Main	Würzburg
137	84.81	Frankfurt/Main	Fulda
138	186.68	Frankfurt/Main	Nürnberg
139	295.08	Frankfurt/Main	Leipzig
140	303.83	Frankfurt/Main	München
141	154.40	Frankfurt/Main	Stuttgart
142	387.15	Frankfurt/Main	Passau
143	152.50	Frankfurt/Main	Köln
144	86.90	Würzburg	Fulda
145	89.90	Würzburg	Nürnberg
146	246.32	Würzburg	Leipzig
147	218.21	Würzburg	München
148	125.99	Würzburg	Stuttgart
149	290.48	Würzburg	Passau
150	247.14	Würzburg	Köln
151	158.29	Fulda	Nürnberg
152	210.82	Fulda	Leipzig
153	300.80	Fulda	München
154	201.32	Fulda	Stuttgart
155	351.47	Fulda	Passau
156	195.84	Fulda	Köln
157	232.83	Nürnberg	Leipzig
158	149.16	Nürnberg	München
159	156.72	Nürnberg	Stuttgart
160	200.63	Nürnberg	Passau

Fortsetzung nächste Seite

Fortsetzung von letzter Seite			
StreckenID	Entfernung	Startbahnhof	Endbahnhof
161	336.44	Nürnberg	Köln
162	362.26	Leipzig	München
163	368.37	Leipzig	Stuttgart
164	319.33	Leipzig	Passau
165	381.94	Leipzig	Köln
166	189.79	München	Stuttgart
167	148.39	München	Passau
168	455.41	München	Köln
169	316.34	Stuttgart	Passau
170	289.24	Stuttgart	Köln
171	535.25	Passau	Köln

Tabelle 6: Ergebnis

Start	Ende	Differenz	Weg
Stuttgart	Passau	261,82	[(16 : 17), (16 : 18)]
München	Passau	261,82	[(16 : 18)]
Cottbus	Fulda	258,45	[(3 : 15), (13 : 15)]
Fulda	Leipzig	258,45	[(13 : 15)]
Rotterdam	Leipzig	236,11	[(6 : 8), (5 : 6), (5 : 15)]
Ruhrgebiet	Leipzig	236,11	[(5 : 6), (5 : 15)]
Hannover	Leipzig	236,11	[(5 : 15)]
Bremen	Leipzig	229,94	[(5 : 7), (5 : 15)]
Passau	Bremen	220,52	[(5 : 7), (5 : 15), (15 : 18)]
Stuttgart	Cottbus	192,49	[(3 : 15), (13 : 15), (12 : 13), (12 : 17)]
Stuttgart	Leipzig	192,49	[(13 : 15), (12 : 13), (12 : 17)]
Stuttgart	Würzburg	179,65	[(12 : 17)]
Stuttgart	Berlin	175,6	[(1 : 15), (13 : 15), (12 : 13), (12 : 17)]
Cottbus	Mannheim	161,07	[(3 : 15), (13 : 15), (11 : 13), (10 : 11)]
Köln	Leipzig	161,07	[(13 : 15), (11 : 13), (11 : 19)]
Cottbus	Frankfurt/Main	161,07	[(3 : 15), (13 : 15), (11 : 13)]
Mannheim	Leipzig	161,07	[(10 : 11), (11 : 13), (13 : 15)]
Frankfurt/Main	Leipzig	161,07	[(11 : 13), (13 : 15)]
Bremen	Nürnberg	160,8	[(5 : 7), (5 : 13), (12 : 13), (12 : 14)]
Bremen	Würzburg	160,8	[(5 : 7), (5 : 13), (12 : 13)]
Bremen	Fulda	160,8	[(5 : 7), (5 : 13)]
Hannover	Bremen	160,8	[(5 : 7)]
Stuttgart	Nürnberg	159,98	[(12 : 14), (12 : 17)]
Rostock	Passau	148,37	[(1 : 2), (1 : 15), (15 : 18)]
Berlin	Passau	148,37	[(1 : 15), (15 : 18)]
Passau	Cottbus	148,37	[(3 : 15), (15 : 18)]
Passau	Leipzig	148,37	[(15 : 18)]
München	Bremen	145,46	[(5 : 7), (5 : 13), (12 : 13), (12 : 14), (14 : 16)]
Rostock	Flensburg	133,78	[(2 : 4)]
Fortsetzung nächste Seite			

Fortsetzung von letzter Seite

Start	Ende	Differenz	Weg
Berlin	Mannheim	124,51	[(1 : 15), (13 : 15), (11 : 13), (10 : 11)]
Berlin	Frankfurt/Main	124,51	[(1 : 15), (13 : 15), (11 : 13)]
Berlin	Fulda	124,51	[(1 : 15), (13 : 15)]
Rostock	Rotterdam	107,28	[(0 : 2), (0 : 7), (7 : 8)]
Flensburg	Rotterdam	107,28	[(0 : 4), (0 : 7), (7 : 8)]
Hamburg	Rotterdam	107,28	[(0 : 7), (7 : 8)]
Bremen	Rotterdam	107,28	[(7 : 8)]
Mannheim	Nürnberg	74,54	[(10 : 12), (12 : 14)]
Mannheim	Würzburg	74,54	[(10 : 12)]
Stuttgart	Bremen	62,69	[(5 : 7), (5 : 13), (12 : 13), (12 : 17)]
Passau	Hannover	59,72	[(5 : 15), (15 : 18)]
Bremen	Frankfurt/Main	51,43	[(5 : 7), (5 : 13), (11 : 13)]
Stuttgart	Hannover	51,09	[(5 : 13), (12 : 13), (12 : 17)]
Stuttgart	Fulda	51,09	[(12 : 13), (12 : 17)]
Köln	Cottbus	44,32	[(3 : 15), (13 : 15), (11 : 13), (11 : 19)]
Berlin	Würzburg	32,51	[(1 : 15), (13 : 15), (12 : 13)]
Cottbus	Würzburg	32,51	[(3 : 15), (13 : 15), (12 : 13)]
Würzburg	Leipzig	32,51	[(12 : 13), (13 : 15)]
Passau	Mannheim	32,49	[(10 : 17), (16 : 17), (16 : 18)]
Flensburg	Leipzig	15,7	[(2 : 4), (1 : 2), (1 : 15)]
Cottbus	Flensburg	15,7	[(1 : 3), (1 : 2), (2 : 4)]
Berlin	Flensburg	15,7	[(1 : 2), (2 : 4)]
Bremen	Mannheim	11,6	[(5 : 7), (5 : 13), (11 : 13), (10 : 11)]
Passau	Flensburg	6,28	[(2 : 4), (1 : 2), (1 : 15), (15 : 18)]
München	Rostock	0	[(1 : 2), (1 : 15), (14 : 15), (14 : 16)]
Rotterdam	Nürnberg	0	[(6 : 8), (6 : 19), (11 : 19), (11 : 12), (12 : 14)]
Rostock	Nürnberg	0	[(1 : 2), (1 : 15), (14 : 15)]
München	Berlin	0	[(1 : 15), (14 : 15), (14 : 16)]
München	Hannover	0	[(5 : 13), (12 : 13), (12 : 14), (14 : 16)]
München	Cottbus	0	[(3 : 15), (14 : 15), (14 : 16)]
Rotterdam	Würzburg	0	[(6 : 8), (6 : 19), (11 : 19), (11 : 12)]
Rotterdam	Fulda	0	[(6 : 8), (6 : 19), (11 : 19), (11 : 13)]
Ruhrgebiet	Nürnberg	0	[(6 : 19), (11 : 19), (11 : 12), (12 : 14)]
Hannover	Rotterdam	0	[(5 : 6), (6 : 8)]
Rotterdam	Frankfurt/Main	0	[(6 : 8), (6 : 19), (11 : 19)]
München	Leipzig	0	[(14 : 15), (14 : 16)]
Berlin	Nürnberg	0	[(1 : 15), (14 : 15)]
Hannover	Nürnberg	0	[(5 : 13), (12 : 13), (12 : 14)]
Cottbus	Nürnberg	0	[(3 : 15), (14 : 15)]
Hannover	Mannheim	0	[(5 : 13), (11 : 13), (10 : 11)]
Rostock	Leipzig	0	[(1 : 2), (1 : 15)]
Köln	Nürnberg	0	[(12 : 14), (11 : 12), (11 : 19)]
München	Frankfurt/Main	0	[(11 : 12), (12 : 14), (14 : 16)]
München	Fulda	0	[(12 : 13), (12 : 14), (14 : 16)]
Ruhrgebiet	Würzburg	0	[(6 : 19), (11 : 19), (11 : 12)]

Fortsetzung nächste Seite

Fortsetzung von letzter Seite

Start	Ende	Differenz	Weg
Rostock	Cottbus	0	[(1 : 2), (1 : 3)]
Ruhrgebiet	Fulda	0	[(6 : 19), (11 : 19), (11 : 13)]
Hannover	Würzburg	0	[(5 : 13), (12 : 13)]
Hannover	Frankfurt/Main	0	[(5 : 13), (11 : 13)]
Köln	Hannover	0	[(5 : 6), (6 : 19)]
Köln	Würzburg	0	[(11 : 12), (11 : 19)]
Rostock	Bremen	0	[(0 : 2), (0 : 7)]
Köln	Rotterdam	0	[(6 : 8), (6 : 19)]
München	Würzburg	0	[(12 : 14), (14 : 16)]
Flensburg	Bremen	0	[(0 : 4), (0 : 7)]
Köln	Fulda	0	[(11 : 13), (11 : 19)]
Nürnberg	Leipzig	0	[(14 : 15)]
Hannover	Ruhrgebiet	0	[(5 : 6)]
Ruhrgebiet	Frankfurt/Main	0	[(6 : 19), (11 : 19)]
Hannover	Fulda	0	[(5 : 13)]
Berlin	Rostock	0	[(1 : 2)]
Stuttgart	München	0	[(16 : 17)]
Frankfurt/Main	Nürnberg	0	[(11 : 12), (12 : 14)]
Ruhrgebiet	Rotterdam	0	[(6 : 8)]
Fulda	Nürnberg	0	[(12 : 13), (12 : 14)]
Stuttgart	Frankfurt/Main	0	[(10 : 11), (10 : 17)]
Mannheim	Fulda	0	[(10 : 11), (11 : 13)]
Hamburg	Rostock	0	[(0 : 2)]
Köln	Frankfurt/Main	0	[(11 : 19)]
München	Nürnberg	0	[(14 : 16)]
Berlin	Leipzig	0	[(1 : 15)]
Hamburg	Flensburg	0	[(0 : 4)]
Cottbus	Leipzig	0	[(3 : 15)]
Berlin	Cottbus	0	[(1 : 3)]
Frankfurt/Main	Würzburg	0	[(11 : 12)]
Stuttgart	Mannheim	0	[(10 : 17)]
Hamburg	Bremen	0	[(0 : 7)]
Würzburg	Nürnberg	0	[(12 : 14)]
Würzburg	Fulda	0	[(12 : 13)]
Frankfurt/Main	Fulda	0	[(11 : 13)]
Mannheim	Frankfurt/Main	0	[(10 : 11)]
Köln	Ruhrgebiet	0	[(6 : 19)]
Cottbus	Rotterdam	-1,31	[(3 : 15), (5 : 15), (5 : 6), (6 : 8)]
Cottbus	Ruhrgebiet	-1,31	[(3 : 15), (5 : 15), (5 : 6)]
Cottbus	Hannover	-1,31	[(3 : 15), (5 : 15)]
Cottbus	Bremen	-7,47	[(3 : 15), (5 : 15), (5 : 7)]
Hamburg	Leipzig	-16,02	[(0 : 7), (5 : 7), (5 : 15)]
München	Mannheim	-16,97	[(10 : 17), (16 : 17)]
Hamburg	Passau	-25,44	[(0 : 7), (5 : 7), (5 : 15), (15 : 18)]
München	Rotterdam	-29,09	[(6 : 8), (6 : 19),

Fortsetzung nächste Seite

Fortsetzung von letzter Seite

Start	Ende	Differenz	Weg
			(11 : 19), (11 : 12), (12 : 14), (14 : 16)]
München	Ruhrgebiet	-29,09	[(6 : 19), (11 : 19), (11 : 12), (12 : 14), (14 : 16)]
München	Köln	-29,09	[(14 : 16), (12 : 14), (11 : 12), (11 : 19)]
Stuttgart	Rostock	-30,63	[(1 : 2), (1 : 15), (13 : 15), (12 : 13), (12 : 17)]
Stuttgart	Flensburg	-34,07	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 17)]
Stuttgart	Hamburg	-34,07	[(0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 17)]
Stuttgart	Rotterdam	-39,83	[(6 : 8), (6 : 19), (11 : 19), (10 : 11), (10 : 17)]
Rotterdam	Mannheim	-39,83	[(6 : 8), (6 : 19), (11 : 19), (10 : 11)]
Stuttgart	Ruhrgebiet	-39,83	[(6 : 19), (11 : 19), (10 : 11), (10 : 17)]
Stuttgart	Köln	-39,83	[(10 : 17), (10 : 11), (11 : 19)]
Ruhrgebiet	Mannheim	-39,83	[(6 : 19), (11 : 19), (10 : 11)]
Köln	Mannheim	-39,83	[(10 : 11), (11 : 19)]
Rostock	Mannheim	-81,72	[(1 : 2), (1 : 15), (13 : 15), (11 : 13), (10 : 11)]
Rostock	Würzburg	-81,72	[(1 : 2), (1 : 15), (13 : 15), (12 : 13)]
Rostock	Frankfurt/Main	-81,72	[(1 : 2), (1 : 15), (13 : 15), (11 : 13)]
Rostock	Fulda	-81,72	[(1 : 2), (1 : 15), (13 : 15)]
München	Flensburg	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 14), (14 : 16)]
Hamburg	München	-85,16	[(0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 14), (14 : 16)]
Flensburg	Nürnberg	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 14)]
Flensburg	Mannheim	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (11 : 13), (10 : 11)]
Flensburg	Würzburg	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (12 : 13)]
Flensburg	Frankfurt/Main	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13), (11 : 13)]
Hamburg	Nürnberg	-85,16	[(0 : 7), (5 : 7), (5 : 13), (12 : 13), (12 : 14)]
Hamburg	Mannheim	-85,16	[(0 : 7), (5 : 7), (5 : 13), (11 : 13), (10 : 11)]
Flensburg	Fulda	-85,16	[(0 : 4), (0 : 7), (5 : 7), (5 : 13)]
Hamburg	Würzburg	-85,16	[(0 : 7), (5 : 7), (5 : 13), (12 : 13)]
Hamburg	Frankfurt/Main	-85,16	[(0 : 7), (5 : 7), (5 : 13), (11 : 13)]
Hamburg	Fulda	-85,16	[(0 : 7), (5 : 7), (5 : 13)]
Rostock	Hannover	-85,16	[(0 : 2), (0 : 7), (5 : 7)]
Flensburg	Hannover	-85,16	[(0 : 4), (0 : 7), (5 : 7)]
Hamburg	Hannover	-85,16	[(0 : 7), (5 : 7)]
Berlin	Köln	-96,9	[(1 : 15), (13 : 15), (11 : 13), (11 : 19)]
Hamburg	Cottbus	-118,08	[(0 : 2), (1 : 2), (1 : 3)]
Berlin	Bremen	-118,08	[(1 : 2), (0 : 2), (0 : 7)]
Hamburg	Berlin	-118,08	[(0 : 2), (1 : 2)]
Köln	Rostock	-123,66	[(0 : 2), (0 : 7), (5 : 7), (5 : 6), (6 : 19)]
Köln	Flensburg	-123,66	[(0 : 4), (0 : 7), (5 : 7), (5 : 6), (6 : 19)]
Rostock	Ruhrgebiet	-123,66	[(0 : 2), (0 : 7), (5 : 7), (5 : 6)]
Flensburg	Ruhrgebiet	-123,66	[(0 : 4), (0 : 7), (5 : 7), (5 : 6)]
Hamburg	Köln	-123,66	[(0 : 7), (5 : 7), (5 : 6), (6 : 19)]
Hamburg	Ruhrgebiet	-123,66	[(0 : 7), (5 : 7), (5 : 6)]
Köln	Bremen	-123,66	[(5 : 7), (5 : 6), (6 : 19)]

Fortsetzung nächste Seite

Fortsetzung von letzter Seite			
Start	Ende	Differenz	Weg
Ruhrgebiet	Bremen	-123,66	[(5 : 6), (5 : 7)]
Passau	Rotterdam	-126	[(6 : 8), (6 : 19), (11 : 19), (11 : 12), (12 : 14), (14 : 16), (16 : 18)]
Passau	Ruhrgebiet	-126	[(6 : 19), (11 : 19), (11 : 12), (12 : 14), (14 : 16), (16 : 18)]
Köln	Passau	-126	[(16 : 18), (14 : 16), (12 : 14), (11 : 12), (11 : 19)]
Passau	Frankfurt/Main	-126	[(11 : 12), (12 : 14), (14 : 16), (16 : 18)]
Passau	Fulda	-126	[(12 : 13), (12 : 14), (14 : 16), (16 : 18)]
Passau	Würzburg	-126	[(12 : 14), (14 : 16), (16 : 18)]
Passau	Nürnberg	-126	[(14 : 16), (16 : 18)]
Berlin	Rotterdam	-142,53	[(1 : 15), (5 : 15), (5 : 6), (6 : 8)]
Berlin	Ruhrgebiet	-142,53	[(1 : 15), (5 : 15), (5 : 6)]
Berlin	Hannover	-142,53	[(1 : 15), (5 : 15)]

Listing 2: MyGraph

```
public class MyGraph{
//Kontenmenge
HashSet<Integer> vertices = new HashSet<Integer>();

//Kantenmenge
HashSet<MyEdge> edges = new HashSet<MyEdge>();

/*
* Konstruktor
*/
public MyGraph(Set<MyEdge> edges) {
 for (MyEdge e: edges){
  this.vertices.add(e.start);
  this.vertices.add(e.end);
  if (!this.edges.contains(new MyEdge(e.end, e.start, e.distance))){
  this.edges.add(e);
  }
}
}

public MyGraph(){
}

public boolean deleteVertex(int i){
return vertices.remove(i);
}

/*
* Getter Methoden
*/
public Set<MyEdge> getNeighborEdges(int node){
 Set<MyEdge> temp = new HashSet<MyEdge>();

 for (MyEdge e: this.edges){
```

```
        if(e.start == node || e.end == node){
                temp.add(e);
        }
}
return temp;
}

public Set<Integer> getNeighborNodes(int node){
Set<Integer> output = new HashSet<Integer>();
if (this.edges != null){
 for (MyEdge e: this.edges){
        if(e.start == node){
                output.add(e.end);
        }
        if(e.end == node){
                output.add(e.start);
        }
}
}
return output;
}

public Set<Integer> getAllNodes(){
 return this.vertices;
}

public List<MyEdge> getAllEdges(){
 return new ArrayList<MyEdge>(edges);
}

public Set<MyEdge> getAllEdgesSet(){
 return edges;
}

public boolean containsNode(Integer node){
 return (this.vertices.contains(node));
}

public boolean containsEdge(MyEdge e){
 return (this.edges.contains(e));
}

/*
* Kanten hinzufügen
*/
public void addEdge(MyEdge e){
 this.vertices.add(e.start);
 this.vertices.add(e.end);
 if (!this.edges.contains(new MyEdge(e.end, e.start, e.distance))){
        this.edges.add(e);
}
}

public void addEdges(Set<MyEdge> edges){
 for (MyEdge e:edges){
```

```
        this.vertices.add(e.start);
        this.vertices.add(e.end);
         if (!this.edges.contains(new MyEdge(e.end, e.start, e.distance))){
          this.edges.add(e);
         }
}
}

public void printGraph(){
 int j = 0;
 for (MyEdge edge: getAllEdges()){
  System.out.println("Kante von Knoten " + edge.getStart()
  + " nach Knoten " + edge.getEnd()
  + " mit einer Enfernung von " + edge.getDistance());
 j++;
}
}
}
```

Listing 3: MyEdge

```
int start, end;
double distance;

/*
* Konstruktor
*/
public MyEdge(int start, int end, double distance){
        this.start=start;
        this.end=end;
        this.distance=distance;
}

MyEdge(){
}


/*
* GetterMethoden
*/
public int getStart(){
        return this.start;
}

public int getEnd(){
        return this.end;
}

public double getDistance(){
        return this.distance;
}
}
```

Listing 4: Pair

```
public class Pair implements Comparable<Pair> {

private Set<Integer> vertices = new HashSet<Integer>();
public double distance;
public double difference;
public List<DefaultWeightedEdge> way;

public Pair(int v1, int v2, double dist, List<DefaultWeightedEdge> list){
 getVertices().add(v1);
 getVertices().add(v2);
 this.distance = dist;
 this.way = list;
}

@Override
public int compareTo(Pair pair0) {
 return -1 * (int) (this.distance - pair0.distance);
}

public boolean isEqual(Pair pair0){
 return this.vertices.equals(pair0.getVertices());
}

public void setVertices(Set<Integer> vertices) {
 this.vertices = vertices;
}

public Set<Integer> getVertices() {
 return vertices;
}
}
```

Listing 5: CalcDistance

```
/**
* Methode, die die Abstände zwischen je zwei Knoten mit Djikstra berechnet
* @param tempGraph
*/
private static LinkedList<Pair> CalcDistance(MyGraph tempGraph){
SimpleWeightedGraph<Integer, DefaultWeightedEdge> graph =
        new SimpleWeightedGraph<Integer,
                DefaultWeightedEdge>(DefaultWeightedEdge.class);

LinkedList<Pair> temp = new LinkedList<Pair>();
int nodes = tempGraph.getAllNodes().size();

//Fügt die Knoten hinzu
for (Integer vertex: tempGraph.getAllNodes()){
        if(vertex != 9){
                graph.addVertex(vertex);
        }
}
```

```
//Fügt die Kanten hinzu
for (MyEdge edge: tempGraph.getAllEdges()){
        graph.addEdge(edge.getStart(), edge.getEnd());
        graph.setEdgeWeight(graph.getEdge(edge.getStart(),
                edge.getEnd()), edge.getDistance());
}

for (int i = 0; i <= nodes; i++){
 for (int j = i + 1; j <= nodes; j++){
 graph.addVertex(i);
 graph.addVertex(j);
 DijkstraShortestPath<Integer, DefaultWeightedEdge> dijkstra =
        new DijkstraShortestPath<Integer, DefaultWeightedEdge>(graph, i, j);
 double path = dijkstra.getPathLength();
 if(i != 9 && j != 9){//Der Knoten mit der externen ID 10
  //ist in der Testinstanz ein isolierter Knoten
  //und wird nicht weiter betrachtet
  temp.add(new Pair(i, j, path, dijkstra.getPathEdgeList()));
 }
}
}
return temp;
}
```

Listing 6: CompareWays

```
/**
* Vergleicht alle Distanzen in einem Graph mit dennen im t-Spanner
* und gibt die geordneten Knotenpaare mit der größten Differenz
* zum t-Spanner zurück
* @param abständeVollständig
* @param abständeTestinstanz
*/
private static LinkedHashMap<Pair, Double> compareWays(
 LinkedList<Pair> abständeSpanner,
 LinkedList<Pair> abständeTestinstanz){

LinkedHashMap<Pair, Double> temp = new LinkedHashMap<Pair, Double>();
LinkedHashMap<Pair, Double> temp2 = new LinkedHashMap<Pair, Double>();

for (int i = 0; i < abständeTestinstanz.size(); i++){
 for (int j = 0; j < abständeSpanner.size(); j++){
        if(abständeTestinstanz.get(i).getVertices().
                equals(abständeSpanner.get(j).getVertices())){

        //Fügt den Unterschied der Wege im Spanner und
        //im Graphen hinzu sowie das Knotenpaar (aus dem Spanner)
  temp.put(abständeSpanner.get(j),
  abständeTestinstanz.get(i).distance - abständeSpanner.get(j).distance);
        }
}
}
return temp;
}
```

Literatur

[Awe85] Awerbuch, B.: Complexity of network synchronisation. In: *J ACM* 32 (1985), S. 804 – 823

[Bol78] Bollobás, B.: *Extremal Graph Theory*. NY : Academic Press, 1978

[CLRS07] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Algorithmen - Eine Einführung*. Oldenbourg Wissensch.Vlg, 2007 http://amazon.com/o/ASIN/3486582623/. – ISBN 9783486582628

[Die06] Diestel, Reinhard: *Graphentheorie (Springer-Lehrbuch Masterclass) (German Edition)*. 3., neu bearb. u. erw. Aufl. Springer, 2006 http://amazon.com/o/ASIN/3540213910/. – ISBN 9783540213918

[Gar03] Garey, David S. Michael R. Johnson J. Michael R. Johnson: *Computers and intractability a guide to the theory of NP-completeness*. New York: Freeman, 2003

[PS89] Peleg, David ; Schäffer, Alejandro A.: Graph Spanners. In: *Journal of Graph Theory* 13 (1989), S. 99 – 116

[Wet03] Wettinghaus, K.: *Graphentheorie*. Hanser Fachbuchverlag, 2003 http://amazon.com/o/ASIN/3446223436/. – ISBN 9783446223431

[WG00] Waldschmidt, H. ; Guntermann, K.: *Grundlagen der Informatik II*. 2000