Hierarchische Graphen zur Wegesuche

Von der Fakultät für Informatik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der
Naturwissenschaften genehmigte Abhandlung

von
Friedhelm Buchholz
aus Fritzlar

Hauptberichter: Prof. Dr. Volker Claus
Mitberichterin: Prof. Dr. Dorothea Wagner

Tag der mündlichen Prüfung: 28. Juli 2000

Stuttgart, 28. August 2000

Diese Arbeit entstand am Lehrstuhl Formale Konzepte der Informatik Fakultät der Universität Stuttgart. Ich danke meinen Kolleginnen und Kollegen der beiden Theorie-Abteilungen für die fruchtbaren Diskussionen und Anregungen zum Thema der Arbeit. Mein besonderer Dank gilt Herrn Prof. Dr. Volker Claus, der es mir als wissenschaftlichem Mitarbeiter ermöglichte, diese Arbeit zu schreiben, und mich stets inspiriert, ermutigt und unterstützt hat. Prof. Dr. Dorothea Wagner danke ich für die Übernahme des Mitberichts.
Weiterhin bedanke ich mich bei Dipl. Inform. Ottokar Kulendik und meinen Schwestern Andrea Buchholz und Birgit Anders für das sorgfältige Korrekturlesen des Manuskriptes.
Meiner Frau Ulrike danke ich für ihre große Unterstützung und die Anregungen beim Schreiben und Korrekturlesen des Manuskriptes.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Wegeprobleme sind ein zentrales Thema der Graphentheorie, die beispielsweise im Zusammenhang mit Verkehrsinformationssystemen, Tourenplanung, Routing und Design von Netzwerken, VLSI Design, verteiltem Rechnen und geometrischen Algorithmen eine wichtige Rolle spielen. Aufgrund der hohen Anzahl von Publikationen zu Wegeproblemen wurden 1984 von Deo und Pang und 1993 von Current und Marsh Klassifikationen der verschiedenen Problemstellungen vorgenommen [DP84] [CM93b].
In dieser Arbeit benutzen wir ein Zwei-Phasenmodell. In der Preprocessing-Phase wird aus einem Graphen ein Hierarchischer-Graph konstruiert und in der Online-Phase wird die Wegesuche (Entfernungsberechnung) mittels des Hierarchischen Graphen durchgeführt. Diese Vorgehensweise wurde insbesondere im Zusammenhang mit Verkehrsgraphen ([CF93] [Car96] [JP96] [SKC93] [LRC+92]) und parallelen Algorithmen ([Lin90] [Dji96]) angewendet (siehe Abbildung 1.1). Hierbei stellen sich Fragen nach der Laufzeit der Preprocessing- ($\mathcal{T}$) und der Online-Phase ($\mathcal{Q}$) und nach der Größe des Hierarchischen-Graphen HG ($\mathcal{S}$) in Relation zu dem gegebenen Graphen G.

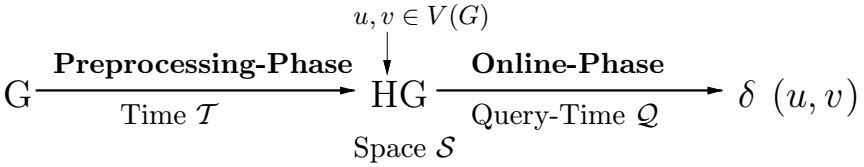

Abbildung 1.1: Zwei-Phasenmodell zur Berechnung der Entfernung $\delta(u, v)$.

Unser Ziel ist die Minimierung der Größen $\mathcal{T}, \mathcal{S}$ und $\mathcal{Q}$, wobei es nur sinnvoll ist, die Größen gemeinsam zu betrachten. Das Optimum besteht dann darin, dass $\mathcal{T}$ und $\mathcal{S}$ linear in der Größenordnung des gegebenen Graphen sind und $\mathcal{Q}$ von konstanter Größenordnung ist. Wir differenzieren außerdem nach der Struktur des gegebenen Graphen und nach der Qualität der berechneten

Entfernung in der Online-Phase. Sofern ein kürzester Weg oder eine kürzeste Entfernung gesucht ist, bezeichnen wir dies als SPSP (Single-Pair-Shortest-Path) oder SPSD (Single-Pair-Shortest-Distance). Sofern 'nur' Approximationen gesucht sind, fügen wir ein 'A' an (ASPSP, ASPSD). Wenn von einem Knoten zu allen anderen Knoten der kürzeste Weg oder die kürzeste Entfernung gesucht ist, schreiben wir SSSP oder SSSD (Single-Source-Shortest-Path, Single-Source-Shortest-Distance). Für alle Knotenpaare schreiben wir APSP (All-Pairs-Shortest-Path).
Für einige einfache Graphstrukturen geben wir in Kapitel 5 optimale Werte für $\mathcal{S}, \mathcal{T}$ und $\mathcal{Q}$ an.
In Kapitel 6 formalisieren wir eine Levelstrategie, die intuitiv von vielen Menschen bei der Wegesuche verfolgt wird. In einer näheren Umgebung vom Start und vom Ziel sucht man nach Auffahrtsmöglichkeiten auf ein 'schnelleres' Straßennetz (Bundesstraßen, Autobahnen) und sucht dann die kürzeste Verbindung in dem 'schnelleren' Straßennetz. Die Auffahrtsknoten verbinden wir in einem zweiten Level vollständig und untersuchen dann Fragen nach der Berechnungskomplexität von minimalen geeigneten Knotenmengen und von geeigneten Knotenmengen, so dass die Umgebungsgrößen der Knoten auf dem unteren Level minimal gewählt werden können. Beide Fragestellungen sind NP-vollständig. Für die erste geben wir eine logarithmische und für die zweite eine konstante Approximation an, die sich jeweils effizient berechnen lassen. Außerdem charakterisieren wir eine Lokalitätseigenschaft, die die Laufzeit in der Preprocessing-Phase in der Praxis wesentlich verringern wird. In Abschnitt 6.6 übertragen wir das Eignungskonzept auf k-Levelgraphen.
In Kapitel 7 wenden wir ein Separationskonzept rekursiv auf einen Graphen an und erzeugen sogenannte Trennende-Hierarchische-Graphen (THG). Weil diese chordal sind, untersuchen wir in den Abschnitten 7.5 und 7.6 chordale Graphen und Triangulationen als Eingabegraphen. Die Laufzeit $\mathcal{Q}$ wird bei der Verwendung von THG'en im Wesentlichen von der größten Separatormenge (Separatorweite) des THG bestimmt, die wir im Zusammenhang mit dem Parameter Baumweite diskutieren [RS86a]. Ein erstaunliches Resultat erzielen wir durch weitere Modifikationen des THG und der Online-Entfernungsberechnung für planare Graphen. Für diese lässt sich auf fast linearem Platz ein Dünner-Trennender-Hierarchischer-Graph aufbauen, so dass die Online-Entfernungsberechnung in $\tilde{O}(\sqrt{n})$ Operationen durchgeführt werden kann. Somit gilt $\mathcal{SQ} \in \tilde{O}(n^{1.5})$, was eine echte Verbesserung des Ergebnisses $\mathcal{SQ} \in \tilde{O}(n^{\frac{5}{3}})$ von Djidjev ist [Dji96]. Eine andere Modifizierung von THG'en führt zu Einfachen-THG'en, die sich für Approximationen der Online-Entfernungsberechnung eignen.
In Kapitel 2 führen wir die verwendeten Notationen ein, und in Kapitel 3 stellen wir die grundlegenden optimalen Verfahren vor, auf die wir in den anderen Kapiteln Bezug nehmen. Bekannte Modellierungstechniken für optimale und approximative Lösungen sind in Kapitel 4 dargestellt.

Kapitel 2

Definitionen

In diesem Kapitel führen wir die in dieser Arbeit benutzten Notationen ein. Die folgenden Abschnitte enthalten elementare Definitionen zu Mengen, Wörtern und Graphen, wie sie auch in der Literatur verwendet werden [Har69]. Spezielle Definitionen, die nur innerhalb einzelner Kapitel verwendet werden, führen wir an geeigneter Stelle ein.

2.1 Mengen, Zahlen, Abbildungen und Wörter

$\mathbb{N}$ sind die natürlichen Zahlen und $\mathbb{N}_0$ sind die natürlichen Zahlen einschließlich der Null. $\mathbb{Z}$ sind die ganzen Zahlen, $\mathbb{Q}$ die rationalen Zahlen und $\mathbb{R}$ die reellen Zahlen. $\mathbb{Z}^+$ sind die nichtnegativen ganzen Zahlen, $\mathbb{Q}^+$ die nichtnegativen rationalen Zahlen und $\mathbb{R}^+$ die nichtnegativen reellen Zahlen. Die obere und untere **Gaußklammer** ist für alle reellen Zahlen $x \in \mathbb{R}$ wie folgt definiert [Knu97]:

$$\begin{aligned} \lceil x \rceil &= \min\{n \in \mathbb{Z} \mid x \leq n\} \\ \lfloor x \rfloor &= \max\{n \in \mathbb{Z} \mid x \geq n\} \end{aligned}$$

Wir verwenden üblicherweise den **Logarithmus** zur Basis zwei, ohne die Zahl zwei explizit anzugeben, als $\log x$ für $x \in \mathbb{R}^+$. Den Logarithmus zur Basis e bezeichnen wir als $\ln x$.
Sei M eine endliche Menge, dann bezeichnet $|M| \in \mathbb{N}_0$ die Kardinalität von M und 2^M die **Potenzmenge** von M. Das kartesische Produkt von zwei Mengen A, B wird durch $A \times B = \{(a, b) \mid a \in A, b \in B\}$ definiert. $A \cup B$ bzw. $A \cap B$ bezeichnet die Vereinigung bzw. die Schnittmenge von A und B. Die Mengen $A_1, \ldots, A_k$ für $k \in \mathbb{N}$ heißen **Partition** der Menge M, wenn gilt:

$$\begin{aligned} A_i &\subseteq M \qquad \forall\, 1 \leq i \leq k \\ M &= \bigcup_{i=1}^{k} A_i \end{aligned}$$

$$A_i \cap A_j \quad = \quad \emptyset \qquad \forall\, 1 \leq i < j \leq k$$

Seien $f : A \to B$ und $g : C \to D$ Abbildungen von A nach B bzw. von C nach D. Die Abbildungen f und g stimmen auf der Menge $W \subseteq A \cap C$ genau dann überein, wenn für alle $w \in W$ gilt:

$$f(w) \quad = \quad g(w)$$

(Schreibweise $f \equiv_W g$). Falls f und g auf $W = A \cap C$ übereinstimmen, schreiben wir auch $f \equiv g$.
Die Abbildungen f und g seien total und auf $B \cup D$ sei eine totale Ordnungsrelation min definiert. Dann definieren wir das Minimum der beiden Abbildungen für alle $x \in A \cup C$ als:

$$h(x) \quad = \quad \begin{cases} \infty & : \quad f(x) \text{ undefiniert und } g(x) \text{ undefiniert} \\ f(x) & : \quad f(x) \text{ definiert und } g(x) \text{ undefiniert} \\ g(x) & : \quad f(x) \text{ undefiniert und } g(x) \text{ definiert} \\ min(f(x), g(x)) & : \quad f(x) \text{ definiert und } g(x) \text{ definiert} \end{cases} \tag{2.1}$$

Der **euklidische Abstand** $d^e : \mathbb{R}^4 \to \mathbb{R}^+$ ist für die Punktepaare $p = (p_x, p_y), q = (q_x, q_y) \in \mathbb{R}^2$ wie folgt definiert:

$$d^e(p, q) \quad = \quad \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Gegeben seien die Mengen $M_1, \ldots, M_k, M$, eine totale Ordnung auf M mit der Operation der Addition und eine Funktion $f : \bigcup_{i=1}^{k-1}(M_i \times M_{i+1}) \longrightarrow M$. Wir definieren folgende Schreibweise:

$$f(M_1, \ldots, M_k) \quad = \quad \min\{\sum_{j=1}^{k-1} f(m_j, m_{j+1}) \mid m_j \in M_j \text{ für } 1 \leq j \leq k\} \tag{2.2}$$

Der **euklidische Abstand auf zwei Mengen** von Punktepaaren $N, M \subset \mathbb{R} \times \mathbb{R}$ ergibt nach Gleichung 2.2 (für $k = 2$) beispielsweise:

$$d^e(N, M) \quad = \quad \min\{d^e(p, q) \mid p \in N, q \in M\}$$

Für eine Abbildung $f : M \to M_1 \times M_2 \times \cdots \times M_k$ bezeichnen wir $f_i : M \to M_i$ als die **Projektion** auf die i-te Komponente von f.
Eine endliche Menge Σ nennen wir auch ein Alphabet. Es sei Σ^* die Menge aller Wörter über dem Alphabet Σ einschließlich des leeren Wortes ϵ. Die Länge eines Wortes $u \in \Sigma^*$ bezeichnen wir mit $|u|$. $\Sigma^i \subseteq \Sigma^*$ sind alle Wörter $u \in \Sigma^*$ mit der Länge $i \in \mathbb{N}_0$. Dabei gilt: $\Sigma^0 = \{\epsilon\}$. $\Sigma^{i,j} := \{u \in \Sigma^* \mid i \leq |u| \leq j\}$ sind alle Wörter mit einer Länge zwischen i und j, wobei $i \leq j$ gelte. $Pr(u, i)$ bezeichnet den **Präfix** der Länge $0 \leq i \leq |u|$ vom Wort $u \in \Sigma^*$ und $Suf(u, i)$ den **Suffix** der Länge $0 \leq i \leq |u|$ vom Wort $u \in \Sigma^*$.

2.2 Graphen

Ein **ungerichteter Graph** $G = (V, E)$ besteht aus einer i. A. endlichen Knotenmenge V und einer endlichen Kantenmenge $E \subseteq \{\{u, v\} \mid u \in V, v \in V\}$. Aus Gründen der einfacheren Schreibweise bezeichnen wir die Knoten- und Kantenmenge von G auch als $V(G)$ und $E(G)$. Für die Anzahl der Knoten und die Anzahl der Kanten verwenden wir i. A. die Buchstaben $n = |V|$ und $m = |E|$. Die Menge aller ungerichteten Graphen bezeichnen wir mit $\mathcal{G}$. Eine Menge von Graphen $\mathcal{G}' \subseteq \mathcal{G}$ heißt **dünn**, wenn die Anzahl der Kanten höchstens linear mit der Anzahl der Knoten wächst, d.h. es gibt ein $k \in \mathbb{R}^+$, so dass für alle Graphen $G \in \mathcal{G}'$ gilt:

$$|E(G)| \quad \leq \quad k|V(G)|$$

Planare Graphen sind beispielsweise dünn ($k = 3$).
Die Kanten $e \in E$ mit $|e| = 1$ heißen Schlingen. Sofern wir es nicht ausdrücklich anmerken, verwenden wir nur schlingenfreie Graphen. Auf den Knoten $u \in V$ und auf Teilmengen $W \subseteq V$ definieren wir folgende **Nachbarschaftsrelationen**:

$$\begin{aligned} N(W) &= W \cup \bigcup_{u \in W} \{v \in V \mid \{u, v\} \in E\} \\ N^0(u) &= \{u\} \\ N^i(u) &= N(N^{i-1}(u)) \\ N^*(u) &= N^{|V|-1}(u) \end{aligned} \tag{2.3}$$

Man beachte, dass nach Gleichung 2.3 die Knoten W auch zu den Nachbarn $N(W)$ gehören. Es gilt $N^1(u) = N(u)$. Dabei bezeichnet $N^*(u)$ auch alle Knoten der **Zusammenhangskomponente** von u. Ein ungerichteter Graph G ist **zusammenhängend**, wenn für einen Knoten $u \in V(G)$ gilt: $N^*(u) = V(G)$. Ein ungerichteter Graph G heißt **vollständig**, wenn für alle Knotenpaare $u, v \in V(G)$ mit $u \neq v$ gilt: $\{u, v\} \in E(G)$. K_n ist der vollständige Graph und C_n der Kreisgraph mit $n \in \mathbb{N}$ Knoten. Die Graphen ohne den Knoten $u \in V(G)$ und ohne die Kante $e \in E(G)$ bezeichnen wir als:

$$\begin{aligned} G - u &:= (V - \{u\}, \{\{p, q\} \in E \mid p \neq u \wedge q \neq u\}) \\ G - e &:= (V, E - \{e\}) \end{aligned}$$

Für **gerichtete Graphen** definieren wir die Kanten als Teilmenge des kartesischen Produkts der Knotenmenge ($E \subseteq V \times V$). Wir übertragen die Nachbarschaftsrelationen wie folgt :

$$\begin{aligned} N^+(W) &= W \cup \bigcup_{u \in W} \{v \in V \mid (u, v) \in E\} \\ N^-(W) &= W \cup \bigcup_{u \in W} \{v \in V \mid (v, u) \in E\} \end{aligned}$$

$$\begin{aligned} N^0(u) &= \{u\} \\ N^i(u) &= N^+(N^{i-1}(u)) \\ N^*(u) &= N^{|V|-1}(u) \end{aligned}$$

Sämtliche folgende Notationen lassen sich leicht auf gerichtete Graphen übertragen. Da wir jedoch fast ausschließlich ungerichtete Graphen verwenden, verzichten wir hier auf weitere Ausführungen dazu.
Ein **Weg** $w = [u_0, u_1, \ldots, u_{k-1}]$ im Graphen G ist eine Knotenfolge $u_i \in V$ für $0 \leq i \leq k-1$, wobei die Kanten $\{u_i, u_{i+1}\}$ für $0 \leq i < k-1$ in der Kantenmenge $E(G)$ liegen. $V(w) = \{u_0, u_1, \ldots, u_{k-1}\}$ sind die Knoten und $E(w) = \{\{u_0, u_1\}, \ldots, \{u_{k-2}, u_{k-1}\}\}$ sind die Kanten des Weges w. $s(w) = u_0$ bezeichnet den Startknoten des Weges w und $e(w) = u_{k-1}$ den Endknoten. $|w| = k-1$ bezeichnet die Länge von w, d.h. die Anzahl von Kanten von w. Analog zu Wörtern bezeichnet $Pr(w, i) = [u_0, \ldots, u_{i-1}]$ den **Präfix** der Länge $i-1$ des Weges w, d.h. die ersten i Knoten von w. $Pr(w, u) = Pr(w, i+1)$ für $u = u_i$ und ein minimales i bezeichnet den Präfix bis zum ersten Auftreten von Knoten $u \in V(w)$ im Weg w. $Suf(w, i) = [u_{k-i}, \ldots, u_{k-1}]$ bezeichnet den Suffix der Länge i von w und $Suf(w, u) = Suf(w, i)$ mit $u_{k-i} = u$ und $k-i$ maximal den Suffix ab dem Knoten $u \in V(w)$. Der Weg w heißt **Teilweg** des Weges $w' = [v_0, \ldots, v_l]$, wenn es ein $i \geq 0$ mit $i + k - 1 \leq l$ gibt, für das gilt:

$$v_{i+j} = u_j \qquad \forall \quad 0 \leq j \leq k-1$$

Ein Teilweg w von w' heißt **echter Teilweg**, wenn die beiden Wege ungleich sind ($w \neq w'$). Die **Konkatenation von Wegen** definieren wir wie folgt: $w \circ w' = [u_0, u_1, \ldots, u_{k-1}, v_1, \ldots, v_l]$, wobei $u_{k-1} = v_0$ gelten muss.
Ein Weg w heißt **doppelpunktfrei**, wenn alle Knoten in w paarweise verschieden sind. Der Weg w heißt **Kreis**, wenn $s(w) = e(w)$ gilt und der Teilweg $Pr(w, |w|)$ doppelpunktfrei ist. Sei w ein Kreis, dann heißt eine Kante $\{u_i, u_j\} \in E(G)$ für $i \neq j$ eine **Sehne** von w, wenn gilt [Sim92]:

$$(i-1) \text{ modulo } k \neq j \quad \wedge \quad (i+1) \text{ modulo } k \neq j$$

Ein Graph $G = (V, E)$ heißt **chordal** genau dann, wenn jeder Kreis der Länge größer gleich vier mindestens eine Sehne enthält.
Jeder Kreis C_n ist für $n \geq 4$ nicht chordal. Vollständige Graphen K_n ($n \in N$) sind chordal. In Abbildung 2.1 ist ebenfalls ein chordaler Graph dargestellt. Chordale Graphen sind auch als Dreiecksgraphen oder triangulierte Graphen bekannt.
Alle Wege in einem Graphen G bezeichnen wir als P^G, alle Wege zwischen den Knoten $u, v \in V(G)$ als $P^G(u, v)$ und alle Kreise in G als K^G.

$$t(G) = \begin{cases} \infty & : \quad K^G = \emptyset \\ \min\{|w| \mid w \in K^G\} & : \quad \text{sonst} \end{cases} \tag{2.4}$$

heißt die **Taillenweite** von G und

$$C(G) = \begin{cases} 0 & : \quad K^G = \emptyset \\ \max\{|w| \mid w \in K^G\} & : \quad \text{sonst} \end{cases} \tag{2.5}$$

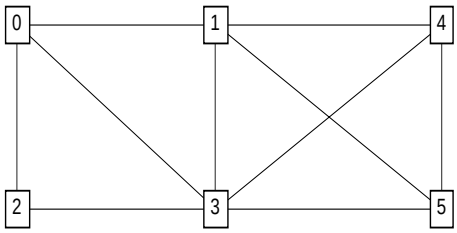

Abbildung 2.1: Ein chordaler Graph.

bezeichnet die **Länge des längsten Kreises von** G [WV74]. Ein Graph G besitzt genau dann einen **Hamiltonkreis**, wenn $C(G) = |V(G)|$ gilt. Für den Petersengraph P gilt: $C(P) = 9$ (siehe Abbildung 2.2).
Der **Abstand** zwischen zwei Knoten $u, v \in V(G)$ ist wie folgt definiert:

$$d(u,v) \quad := \quad \begin{cases} \infty & \text{falls} \quad P^G(u,v) = \emptyset \\ \min\{|w| \mid w \in P^G(u,v)\} & \text{sonst} \end{cases}$$

In dem Graphen $G = (V, E)$ **induziert** die Knotenmenge $A \subseteq V$ den Graphen

$$G[A] \quad = \quad (A, \{\{u,v\} \in E \mid u, v \in A\})$$

Ein Graph $G' = (V', E')$ ist ein **Teilgraph** des Graphen $G = (V, E)$, wenn gilt: $V' \subseteq V$ und $E' \subseteq E$ (Schreibweise $G' \subseteq G$). Die Knoten eines vollständigen Teilgraphen von G heißen **Clique** von G. Die **Cliquengröße** eines Graphen G ist definiert als die maximale Anzahl von Knoten eines vollständigen Teilgraphen von G:

$$\omega(G) \quad := \quad \max\{|V(G')| \mid G' \text{ ist vollständiger Teilgraph von } G\}$$

Ein Graph $G = (V, E)$ heißt **planar**, wenn sich der Graph G so in die Ebene zeichnen lässt, dass sich keine Kanten überkreuzen.

$$d(u) \quad = \quad |\{v \in V \mid \{u,v\} \in E\}|$$

heißt der **Grad eines Knotens** $u \in V(G)$ des ungerichteten Graphen G.

$$\Delta(G) \quad = \quad \max\{d(v) \mid v \in V(G)\}$$

heißt der G **Grad des Graphen** G.
Ein ungerichteter Graph heißt **k-zusammenhängend**, wenn es für jedes Knotenpaar mindestens k bis auf Anfangs- und Endknoten disjunkte Wege gibt. Ein zusammenhängender Graph ist 1-zusammenhängend. Der zusammenhängende ungerichtete Graph G heißt **Baum**, wenn die Anzahl der Knoten von G um eins

größer ist als die Anzahl der Kanten von G ($|V(G)| = |E(G)| + 1$). Ein gerichteter Graph T heißt Baum, wenn die ungerichtete Version von T ein Baum ist und es einen ausgezeichneten Knoten $r \in V(T)$ gibt, den wir als Wurzel von T bezeichnen und für den gilt: $N^*(r) = V(T)$. Die Knoten mit Grad eins heißen Blätter, und alle Knoten, die kein Blatt und keine Wurzel in einem Baum sind, bezeichnen wir als **innere Knoten**. Die **Höhe eines Baums** T ist die maximale Länge eines gerichteten Weges in T:

$$H(T) \quad := \quad \max\{|w| \mid w \in P^T(u,v), u, v \in V(T)\}$$

Die Höhe eines Baums mit nur einem Knoten ist folglich Null. Die Wurzel eines Baums ist in Tiefe Null und $N^i(r) - N^{i-1}(r)$ sind die Knoten in **Tiefe** $i \in \mathbb{N}$ **des Baumes** T. Den nächsten gemeinsamen Vorgänger im gerichteten Baum T definieren wir für die Knoten $u, v \in V(T)$ als (nearest-common-ancestor):

$$\begin{aligned} nca(u,v) = x \quad \Longleftrightarrow \quad & u \in N^*(x) \wedge v \in N^*(x) \wedge \\ & \forall y \in N^+(x) - \{x\} \quad (u \notin N^*(y) \vee v \notin N^*(y)) \end{aligned}$$

Der ungerichtete Graph $G = (V \cup U, E)$ mit $V \cap U = \emptyset$ heißt **bipartit**, wenn für alle Kanten $e \in E$ gilt: $|e \cap U| = 1$ und $|e \cap V| = 1$. Wir schreiben den Graphen dann auch als 3-Tupel $G = (V, U, E)$ und bezeichnen die beiden Knotenmengen des Graphen G mit $V(G)$ und $U(G)$. Den vollständigen bipartiten Graphen G mit $|V(G)| = n$ und $|U(G)| = r$ bezeichnen wir als $K_{n,r}$.
Die Vereinigung von Graphen G, G' ist über die Vereinigung der Knoten- und Kantenmengen definiert, die dabei nicht disjunkt sein müssen:

$$G \cup G' \quad = \quad (V(G) \cup V(G'), E(G) \cup E(G'))$$

Ein Graph heißt **regulär**, wenn alle Knotengrade gleich sind. Offensichtlich gilt für jeden regulären Graphen vom Grad $r \in \mathbb{N}$: $2m = nr$. Ein regulärer Graph heißt **kubisch**, wenn sein Grad drei ist. Ein Beispiel für einen kubischen Graphen ist der nicht planare Petersengraph [WB89] (siehe Abbildung 2.2).
Wir führen eine Familie von regulären Graphen mit Grad vier ein. Für ein $k \in \mathbb{N}$ heißt der Graph $G = (V, E)$ mit k^2 Knoten ein **geschlossener** $k \times k$**-Gittergraph**, wenn gilt:

$$\begin{aligned} V \quad &= \quad \{(i,j) \mid i, j \in \{1, 2, \ldots, k\}\} \\ E \quad &= \quad \{\{(i,j),(i',j')\} \mid (i = i' \wedge (|j - j'| = 1 \vee |j - j'| = k - 1) \\ & \quad \vee (j = j' \wedge (|i - i'| = 1 \vee |i - i'| = k - 1)\} \end{aligned}$$

Aufgrund der Symmetrieeigenschaften von geschlossenen Gittergraphen ist die Anzahl der Knoten innerhalb eines Abstands $0 \leq i \leq k$ für alle Knoten $u, v \in V(G)$ gleich, d.h. es gilt $|N^i(u)| = |N^i(v)|$. Wir definieren daher die Anzahl der Knoten in einem geschlossenenen Gittergraphen G bis zum Abstand i als:

$$A_G(i) \quad = \quad |N^i(u)| \tag{2.6}$$

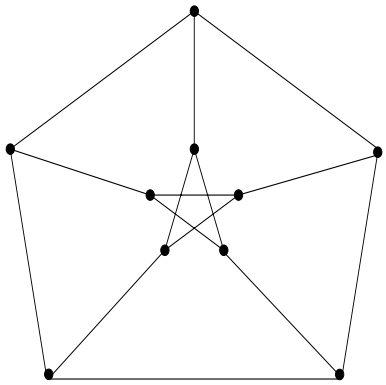

Abbildung 2.2: Der Petersengraph.

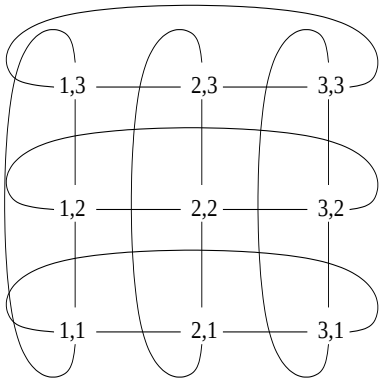

Abbildung 2.3: Ein geschlossener 3×3-Gittergraph.

Sei $k \in \mathbb{N}$ ungerade und G ein geschlossener $k \times k$-Gittergraph. Dann gilt für $0 \leq j < \frac{k}{2}$:

$$\begin{aligned} A_G(j) &= 1 + \sum_{i=1}^{j} 4i \\ &= 2j^2 + 2j + 1 \end{aligned} \tag{2.7}$$

Für $\frac{k}{2} < j \leq k$ gilt:

$$A_G(j) = 2A_G(\lfloor \frac{k}{2} \rfloor) - A_G(k-j) \tag{2.8}$$

2.3 Gewichtete Graphen

Sei (V, E) ein Graph. $G = (V, E, \delta)$ heißt **gewichteter Graph** mit Kantengewichtsfunktion $\delta : E \to \mathbb{R}^+$. Die Kantengewichtsfunktion wird wie folgt auf Wege $w = [u_1, u_2, \ldots, u_k]$ für $k \geq 2$ (für $k = 1$ ist $\delta(w) = 0$) und Knotenpaare $u, v \in V$ fortgesetzt:

$$\begin{aligned} \delta(w) &:= \sum_{i=1}^{k-1} \delta(\{u_i, u_{i+1}\}) \\ \delta(u,v) &:= \begin{cases} \infty & \text{falls} \quad P^G(u,v) = \emptyset \\ \min\{\delta(w) \mid w \in P^G(u,v)\} & \text{sonst} \end{cases} \end{aligned} \tag{2.9}$$

Die Menge aller kürzesten Wege in G bezeichnen wir mit SP^G und die Menge aller kürzesten Wege zwischen den Knoten $u, v \in V(G)$ als $SP^G(u, v)$. Sofern klar ist, auf welchen Graphen wir uns beziehen, schreiben wir auch einfach SP anstatt SP^G. Ein Weg w heißt eindeutiger kürzester Weg, wenn gilt: $\{w\} = SP(s(w), e(w))$. Ein eindeutiger kürzester Weg $w = [v_1, \ldots, v_r]$ heißt maximal, wenn es keinen eindeutigen kürzesten Weg $w' \in SP$ gibt, der w als echten Teilweg enthält. Man beachte, dass die Eigenschaft maximal nur für eindeutige Wege definiert ist.
In gewichteten Graphen G definieren wir zusätzlich zum Abstand zwischen Knotenpaaren einen **Abstand-δ** für ein Knotenpaar $u, v \in V(G)$ als die minimale Kantenanzahl für einen kürzesten Weg (und nicht für einen beliebigen Weg) zwischen u und v:

$$d_\delta(u, v) = \min\{|w| \mid w \in SP^G(u, v)\} \tag{2.10}$$

Ebenso definieren wir in gewichteten Graphen eine **Nachbarschaftsrelation-δ** für jeden Knoten $u \in V$ und $i \in \mathbb{N}_0$.

$$N_\delta^i(u) = \{v \in V \mid d_\delta(u, v) = i\} \tag{2.11}$$

Man beachte, dass sich die Abstände und Nachbarschaften in gewichteten Graphen von denen in ungewichteten Graphen unterscheiden. Für die Knoten a, b des Graphen in Abbildung 2.4 gilt:

$$\begin{aligned} d_\delta(a, b) &= 2 \\ d(a, b) &= 1 \end{aligned}$$

Wir erweitern die Nachbarschaftsrelation noch auf Knotenmengen $W \subseteq V$ für $1 \leq i \leq n$.

$$N_\delta(W) = W \cup \bigcup_{u \in W} \{v \in V \mid d_\delta(u, v) = 1\} \tag{2.12}$$

$$\begin{aligned} N_\delta^0(W) &= W \\ N_\delta^i(W) &= N_\delta(N_\delta^{i-1}(W)) \end{aligned} \tag{2.13}$$

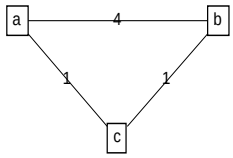

Abbildung 2.4: Beispiel für unterschiedliche Abstandsfunktionen.

Lemma 1 *Gegeben sei ein gewichteter Graph $G = (V, E, \delta)$. $N, N^i_\delta : 2^V \to 2^V$ seien die entsprechenden Nachbarschaftsrelationen. Dann gilt für alle Knotenmengen $W \subseteq V$ und alle $i \in \mathbb{N}_0$ stets:*

$$N^i_\delta(W) \quad \subseteq \quad N^i(W) \tag{2.14}$$

Beweis: Seien $d, d_\delta : V \times V \to \{0, 1, \ldots, n\}$ die Abstandsfunktionen in G. Es gilt für alle Knotenpaare $u, v \in V$ offensichtlich:

$$d(u, v) \quad \leq \quad d_\delta(u, v)$$

Daraus folgt die Aussage.

■

$$diam(G) \quad = \quad \max\{d(u, v) \mid u, v \in V\}$$

heißt **Diameter** von G und

$$diam^\delta(G) \quad = \quad \max\{\delta(u, v) \mid u, v \in V\}$$

heißt **Entfernungsdiameter** von G. Sei (V, E, δ) ein gewichteter Graph. $G = (V, E, \delta, \mu)$ heißt **Ortsgraph** mit Knotengewichtsfunktion $\mu : V \to \mathbb{R} \times \mathbb{R}$. Ein Ortsgraph $G = (V, E, \delta, \mu)$ heißt **euklidisch**, wenn die Kantengewichte mit dem euklidischen Abstand übereinstimmen, d.h. für alle Kanten $\{u, v\} \in E$ gilt:

$$\delta(\{u, v\}) \quad = \quad d^e(\mu(u), \mu(v))$$

Die Graphen $G = (V, E, \delta)$ und $G' = (V', E', \delta')$ heißen bzgl. der Knotenmenge $W \subseteq V \cap V'$ **längengleich** genau dann, wenn für alle Knoten $u, v \in W$ gilt:

$$\delta(u, v) \quad = \quad \delta'(u, v) \tag{2.15}$$

Wir schreiben dafür $G \equiv_W G'$. Im Fall von $W = V = V'$ heißen die beiden Graphen nur längengleich (Schreibweise $G \equiv G'$).
Die **Vereinigung gewichteter Graphen** G, G' ist definiert als:

$$G \cup G' \quad = \quad (V \cup V', E \cup E', \min(\delta, \delta')) \tag{2.16}$$

Für die Minimumbildung von Abbildungen siehe Gleichung 2.1.

2.4 Approximationen

Für ein Optimierungsproblem P sei $\mathcal{L}$ eine Menge von Lösungen und $f : \mathcal{L} \rightarrow \mathbb{R}^+ - \{0\}$ eine Bewertung der Lösungen. O.B.d.A. sei P ein Minimierungsproblem und $L^* \in \mathcal{L}$ eine optimale Lösung von P. Eine Lösung $L \in \mathcal{L}$ heißt α-Approximation für $\alpha \geq 1$, wenn gilt [GJ79]:

$$\frac{f(L)}{f(L^*)} \quad \leq \quad \alpha. \tag{2.17}$$

Ein Algorithmus heißt α-Approximation für P, wenn nur Lösungen berechnet werden, die α-Approximationen sind.
Wenn α konstant, d.h. unabhängig von der Problemgröße ist, sprechen wir auch von konstanten Approximationen.

2.5 Komplexitätsmaße

Wir messen den Laufzeit- und den Speicherbedarf der Algorithmen in Abhängigkeit der Größe (Knoten- und Kantenanzahl) der gegebenen Graphen. Nach Einführung der grundlegenden Problemstellung dieser Arbeit (siehe Abbildung 1.1) sei A ein Preprocessing-Algorithmus zur Konstruktion eines Hierarchischen-Graphen HG und B ein Online-Algorithmus zur Entfernungsberechnung mit HG. Für einen gegebenen Graphen mit n Knoten und m Kanten bezeichnet $\mathcal{S}(n, m)$ (Space) den Speicherplatz von HG, $\mathcal{T}(n, m)$ die Laufzeit von Algorithmus A und $\mathcal{Q}(n, m)$ die Laufzeit von Algorithmus B. Aus Gründen der Einfachheit schreiben wir im Weiteren nur S, T, Q. Wir benutzen für die Maße die O-Notation, wobei $\tilde{O}()$ bedeutet, dass die logarithmischen Faktoren entfallen [CLR96]. Wir benutzen ein Registermaschinenmodell, d.h. Zahlen beliebiger Größe können in einem Register gespeichert werden und die üblichen Rechenoperationen (Addition, Subtraktion, Multiplikation, Division) werden in konstanter Zeit ausgeführt.
P ist die Klasse der in polynomieller Zeit mit deterministischen Turingmaschinen lösbaren Probleme, während NP die Klasse der mit nichtdeterministischen Turingmaschinen in polynomieller Zeit lösbaren Probleme ist. Aus der Theorie dieser Komplexitätsklassen benutzen wir insbesondere die NP-Vollständigkeit und den Reduktionsbegriff (siehe [GJ79]).

Kapitel 3

Optimale Verfahren

3.1 Dijkstra

Das bekannteste Verfahren zur Kürzesten Wegesuche wurde von E. W. Dijkstra entwickelt und löst SSSP und SPSP für Graphen mit nichtnegativen Kantengewichten in $O(n^2 + m)$ Operationen [Dij59] (siehe Algorithmus *Dijkstra*). Dieses Verfahren wurde mehrfach modifiziert. Im Laufzeitverhalten ergab sich eine Verbesserung auf $O(m + n \log n)$ durch die Verwendung von Fibonacci Heaps für die Verwaltung der Randknotenmenge, die Fredman und Tarjan 1987 vorschlugen [FT87]. 1990 verbesserten Fredman und Willard dies auf $O(m + n\frac{\log n}{\log \log n})$ [FW90].
Wir geben den *Dijkstra* Algorithmus in der Version mit Start- und Zielknoten an, während in der ursprünglichen Fassung der Zielknoten entfällt. In der Bedingung für die while-Schleife ist dann lediglich der zweite Term wegzulassen.
Zur einfacheren Schreibweise definieren wir zu einem gewichteten Graphen $G = (V, E, \delta)$ die Relation $AR()$ und die Funktionen $Next()$ und $Update()$. Für eine Knotenmenge $S \subseteq V$ bezeichnet $AR(S) = N(S) - S$ den Außenrand von S. $u \in V$ sei ein beliebiger Knoten und $\delta' : V \to \mathbb{R}^+$ eine Abbildung auf den Knoten des Graphen, die im Algorithmus als Entfernungsannäherung benutzt wird.

Funktion 1 $Next(\delta', S)$

Gibt einen Knoten $y \in AR(S)$ an, für den gilt:

$$\delta'(y) \quad = \quad \min\{\delta'(x) \mid x \in AR(S)\}$$

Funktion 2 $Update(u, \delta', S)$

$S := S \cup \{u\}$;
for $x \in N(u) \cap (V - S)$ **do** $\delta'(x) := min(\delta'(x), \delta'(u) + \delta(\{u, x\}))$ **endfor** ;

Wir bezeichnen eine Knotenmenge $S \subseteq V$ bzgl. eines Knotens $s \in V$ als **äquidistant**, wenn für alle Knoten $u \in S$ und $v \in V - S$ gilt:

$$\delta(s,u) \quad \leq \quad \delta(s,v) \tag{3.1}$$

Der Algorithmus *Dijkstra* berechnet eine äquidistante Knotenmenge S.

Algorithmus 1 *Dijkstra*

Eingabe: $G = (V, E, \delta); s, t \in V$

Ausgabe: $\delta'(t) = \delta(s,t), \delta'(v) = \delta(s,v) \quad \forall v \in \{w \in V \mid \delta(s,w) < \delta(s,t)\}$

for $u \in V$ **do** $\delta'(u) := \infty$ **endfor** ;
for $u \in N(s)$ **do** $\delta'(u) := \delta(\{s,u\})$ **endfor** ;
$S := \{s\}$;
while $AR(S) \neq \emptyset$ **and** $t \notin S$ **do**
    $u := Next(\delta', S)$;
    $Update(u, \delta', S)$
endwhile ;

3.1.1 Korrektheit

Satz 1 *Der Algorithmus Dijkstra berechnet für Graphen G mit nichtnegativen Kantengewichten für jedes Knotenpaar s, t die kürzeste Entfernung, d.h. nach der Ausführung gilt: $\delta'(t) = \delta(s,t)$.*

Beweis: Wir zeigen durch Induktion über die Anzahl der Schleifendurchläufe, dass folgende Schleifeninvariante gilt:

$$\delta'(u) \quad = \quad \delta(s,u) \quad \forall u \in S \tag{3.2}$$

Zu Beginn ist $S = \{s\}$ und $\delta'(s) = 0$ und die Invariante ist somit gültig. Angenommen, es sei $u = Next(\delta', S)$ für einen Knoten $u \in V - S$ und es gebe einen kürzesten Weg $w = [s = x_0, \ldots, x_k = u] \in SP(s,u)$ mit $\delta(w) < \delta'(u)$. O.B.d.A. sei x_i der erste Knoten von w, der nicht zu S gehört. Nach Induktionsvoraussetzung gilt $\delta'(x_{i-1}) = \delta(s, x_{i-1})$. Aufgrund der Funktion $Update()$ gilt dann aber sogar $\delta'(x_i) = \delta(s, x_i)$. Wegen

$$\begin{aligned} \delta'(x_i) &= \delta(s,x_i) \\ &\leq \delta(w) \\ &< \delta'(u) \end{aligned}$$

wäre x_i anstatt u von der Funktion $Next()$ ausgewählt worden.

■

Die Invariante 3.2 im obigen Beweis lässt sich sogar noch verschärfen.

Lemma 2 *Seien $S \subseteq V$ die Knoten, für die die Entfernung bereits bestimmt ist. Die Entfernungsannäherung δ' stimmt bereits für die Außenrandknoten $v \in AR(S)$ mit der kürzesten Entfernung überein ($\delta'(v) = \delta(s,v)$), wenn es einen kürzesten Weg $w \in SP(s,v)$ mit $V(Pr(w, |w|-1)) \subseteq S$ gibt.*

3.1.2 Laufzeit

Es werden höchstens n Knoten in die Knotenmenge S aufgenommen. Bei jeder Knotenaufnahme wird das Minimum aus dem Rand entnommen und ein Update für die Nachbarknoten durchgeführt. Dabei wird jede Kante höchstens einmal betrachtet. Somit ist die Anzahl der Neuberechnungen von δ' im Inneren der Funktion $Update()$ auf $n+m$ begrenzt. Die Funktion $Next()$ wird höchstens $n-1$ mal aufgerufen und benötigt bei einfacher Implementierung durch eine Liste höchstens n Operationen. Somit ergibt sich eine Gesamtlaufzeit von $O(n^2+m)$ Operationen. Wird die Knotenmenge $AR(S)$ in einem gewöhnlichen Heap verwaltet, ergibt sich eine Laufzeit von $O((n+m)\log n)$ Operationen, weil nun jeder Aufruf von $Next()$ in konstanter Zeit ausgeführt wird und jede Neuberechnung von $\delta'()$ (Umorganisation des Heaps) im Inneren von $Update()$ $O(\log n)$ Operationen benötigt. Mit Fibonacci Heaps werden zum Löschen $O(\log n)$ Operationen benötigt und alle anderen Heap-Operationen können in konstanter Zeit ausgeführt werden. Daraus ergibt sich für den Algorithmus *Dijkstra* eine Laufzeit von $O(m + n\log n)$ [FW90].
Um die genaue Anzahl der Knoten zu analysieren, die der *Dijkstra* Algorithmus untersucht, definieren wir für einen Graphen G mit Start- und Zielknoten $s,t \in V(G)$:

$$V_G(s,t) \quad = \quad \{x \in V \mid \delta(s,x) \leq \delta(s,t)\} \tag{3.3}$$

Für einen Graphen G und ein Knotenpaar $s,t \in V(G)$ sei $S \subseteq V$ die maximale Knotenmenge, die eine Implementierung des *Dijkstra* Algorithmus am Ende berechnet. Wir definieren die Anzahl der insgesamt betrachteten Knoten als:

$$Dij_G(s,t) \quad = \quad |N(S)| \tag{3.4}$$

Es gilt dann offensichtlich für alle Knoten $s,t \in V(G)$:

$$Dij_G(s,t) \quad \leq \quad |N(V_G(s,t))| \tag{3.5}$$

Wenn die Kantengewichte alle ungleich Null sind, dann gilt auch eine untere Schranke:

$$|V_G(s,t)| \quad \leq \quad Dij_G(s,t)$$

Außerdem werden vom Algorithmus *Dijkstra* maximal alle Kanten in dem von $N(V_G(s,t))$ in G induzierten Graphen $G[N(V_G(s,t))]$ betrachtet.

3.2 Beidseitiger Dijkstra

Wir betrachten nun eine Modifikation des Algorithmus *Dijkstra*, die bei gleichmäßig zusammenhängenden Graphen Vorteile aufweist. Insbesondere ist die Anzahl der insgesamt untersuchten Knoten unter Umständen geringer. Im worstcase ergibt sich i. A. keine Verbesserung im Laufzeitverhalten (siehe Abschnitt 3.2.2).
Die Wegsuche wird nicht nur vom Startknoten aus durchgeführt, sondern auch vom Zielknoten aus. Wir verwenden anstatt einer Knotenmenge $S \subseteq V$ nun zwei Knotenmengen $S_s, S_t \subseteq V$ und zwei weitere Näherungsfunktionen $\delta'_s, \delta'_t : V \to \mathbb{R}^+$.

Algorithmus 2 *Dijkstra-beidseitig*

Eingabe: $G = (V, E, \delta); s, t \in V$

Ausgabe: $\delta'(m)$

$S_s := \{s\}$; $S_t := \{t\}$;
for all $x \in V$ **do** $\delta'_s(x) := \infty$; $\delta'_t(x) := \infty$; **endfor** ;
for all $x \in AR(S_s)$ **do** $\delta'_s(x) := \delta(\{x, s\})$; **endfor** ;
for all $x \in AR(S_t)$ **do** $\delta'_t(x) := \delta(\{x, t\})$; **endfor** ;
$\delta'_s(s) := 0$; $\delta'_t(t) := 0$;
$m := s$; $\delta'(m) := \delta'_t(s)$ *(willkürliche Auswahl,* $\delta'(m) := \delta'_s(t)$ *ebenso korrekt)*
while $S_s \cap S_t = \emptyset$ **do**
  $u := Next(\delta'_s, S_s)$; $v := Next(\delta'_t, S_t)$;
  if $\delta'_s(u) \leq \delta'_t(v)$
    then $Update(u, \delta'_s, S_s)$;
      if $\delta'_s(u) + \delta'_t(u) < \delta'(m)$
        then $m := u; \delta'(m) := \delta'_s(u) + \delta'_t(u)$;
      endif ;
    else $Update(v, \delta'_t, S_t)$;
      if $\delta'_s(v) + \delta'_t(v) < \delta'(m)$
        then $m := v; \delta'(m) := \delta'_s(v) + \delta'_t(v)$;
      endif ;
  endif ;
endwhile ;

3.2.1 Korrektheit

Satz 2 *Der Algorithmus Dijkstra-beidseitig berechnet die kürzeste Entfernung zwischen zwei Knoten* ($\delta'(m) = \delta(s, t)$).

Beweis: Offensichtlich werden im Algorithmus nur korrekte Weglängen bestimmt, d.h. es gilt immer $\delta'(m) \geq \delta(s, t)$. Nach einer analogen Begründung

wie im Korrektheitsbeweis des Algorithmus *Dijkstra* gilt für die Entfernungsannäherungen δ'_s, δ'_t und alle Knoten $u \in S_s$ und $v \in S_t$:

$$\begin{aligned} \delta'_s(u) &= \delta(s,u) \\ \delta'_t(v) &= \delta(t,v) \end{aligned}$$

O.B.d.A. endet der Algorithmus mit $S_s \cap S_t = \{y\}$ (siehe Abbildung 3.1).

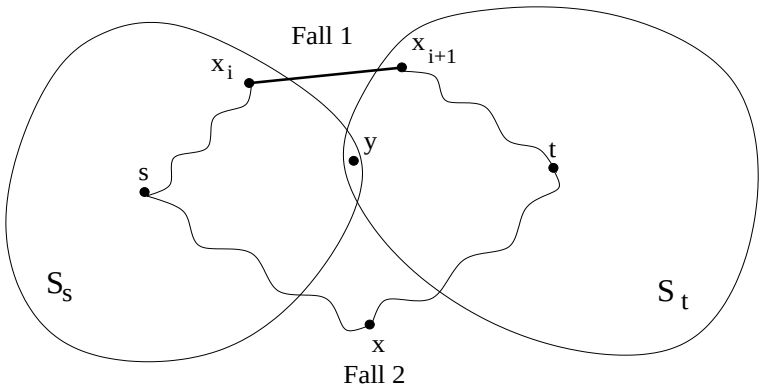

Abbildung 3.1: Veranschaulichung zum Korrektheitsbeweis von *Dijkstra-beidseitig*.

Fall 1: Sei $w \in SP(s,t)$ ein kürzester Weg mit $V(w) \subseteq S_s \cup S_t$. Weil S_s und S_t äquidistant sind, gibt es zwei aufeinander folgende Knoten auf dem Weg $w = [s, \ldots, x_i, x_{i+1}, \ldots, t]$, so dass der Präfix bis x_i in S_s und der Suffix ab x_{i+1} in S_t liegen, d.h. es gilt:

$$\begin{aligned} V(Pr(w, x_i)) &\subseteq S_s \\ V(Suf(w, x_{i+1})) &\subseteq S_t \end{aligned}$$

Nach Lemma 2 folgt für die Entfernungsannäherungen

$$\begin{aligned} \delta'_s(x_{i+1}) &= \delta(s, x_{i+1}) \\ \delta'_t(x_i) &= \delta(t, x_i) \\ \delta'(m) &\leq \delta'_s(x_i) + \delta'_t(x_i) \\ &= \delta(w) \end{aligned}$$

Fall 2: Angenommen, auf allen kürzesten Wegen $w \in SP(s,t)$ gibt es einen Knoten $x \in V(w)$ mit $x \notin S_s \cup S_t$. Weil die Knotenmengen

S_s und S_t für s bzw. t äquidistant sind, gilt aber $\delta'(m) \leq \delta(s,t)$. Folglich gibt es im Widerspruch zur Annahme einen kürzesten Weg $w' \in SP(s,t)$ mit $V(w') \subseteq S_s \cup S_t$.

■

Man beachte, dass der kürzeste Weg zwischen s und t nicht unbedingt über den Knoten $\{y\} = S_s \cap S_t$ verlaufen muss. Dies wird im Beispiel in Abbildung 3.2 dargestellt. Die nicht eingezeichneten Kantengewichte seien eins. *Dijkstra-beidseitig* berechnet die Knotenmengen $S_s = \{s, a, b, d\}, S_t = \{b, c, t\}$ und eine Weglänge $\delta(s,t) = 3,2$. Der kürzeste Weg zwischen s und t verläuft hier nicht über den Knoten $b = S_s \cap S_t$.

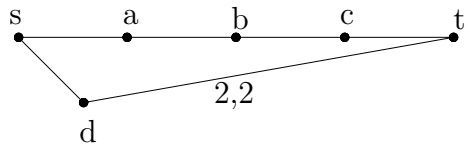

Abbildung 3.2: Beispielgraph für *Dijkstra-beidseitig*.

3.2.2 Laufzeit

In der O-Notation ändert sich das Laufzeitverhalten nicht, wenn der Algorithmus *Dijkstra* durch *Dijkstra-beidseitig* ersetzt wird. Es werden schließlich lediglich zwei Mengen S_s, S_t anstatt nur einer Knotenmenge S und drei Entfernungsannäherungen anstatt einer verwaltet. Um den Vorteil explizit anzugeben, betrachten wir geschlossene $k \times k$-Gittergraphen, die wir um die konstante Gewichtsfunktion $\delta \equiv 1$ erweitern (siehe Definition auf Seite 16). Die Anzahl der untersuchten Knoten von Algorithmus *Dijkstra-beidseitig* für ein Knotenpaar $u, v \in V(G)$ definieren wir analog zu $Dij_G()$ (siehe Gleichung 3.4) als

$$DijB_G(u,v) \quad = \quad |N(S_u)| + |N(S_v)|$$

Die Anzahl der verschiedenen untersuchten Knoten ist noch um eins kleiner, weil der Knoten $y = N(S_u) \cap N(S_v)$ zweimal aufgenommen wird. Sei $k \in \mathbb{N}$ ungerade und G ein geschlossener $k \times k$-Gittergraph. Dann gilt:

$$A_G(\delta(u,v)) \quad \leq \quad Dij_G(u,v) \quad \leq \quad A_G(\delta(u,v)+1)$$

$$DijB_G(u,v) \quad \leq \quad \begin{cases} 2A_G(\frac{\delta(u,v)}{2}+1) & \text{falls} \quad \delta(u,v) \text{ gerade ist} \\ 2A_G(\lceil\frac{\delta(u,v)}{2}\rceil+1) & \text{sonst} \end{cases}$$

Für den Fall $\delta(u,v) < \frac{k}{2}$ mit gerader Entfernung $\delta(u,v)$ ergibt sich nach Gleichung 2.7:

$$\begin{aligned} Dij_G(u,v) &\geq 2\delta^2(u,v) + 2\delta(u,v) + 1 \\ DijB_G(u,v) &\leq \frac{\delta^2(u,v)}{2} + 3\delta(u,v) + 5 \end{aligned}$$

Es sind also beim *Dijkstra-beidseitig* etwas mehr als die Hälfte aller Knoten zu untersuchen. Bei mehrdimensionalen k^r-Gittergraphen reduziert sich die Anzahl der untersuchten Knoten sogar auf einen Anteil von $\frac{1}{2^{r-1}}$ Knoten.
Allerdings zeigt sich dieses Verhalten nicht für alle Graphen. In Abbildung 3.3 ist ein Graph G dargestellt, bei dem es sich sogar umgekehrt verhält. Es werden beim Algorithmus *Dijkstra* $Dij_G(s,t) = 20$ Knoten untersucht und bei *Dijkstra-beidseitig* alle Knoten $Dij_G(s,t) = 29$.

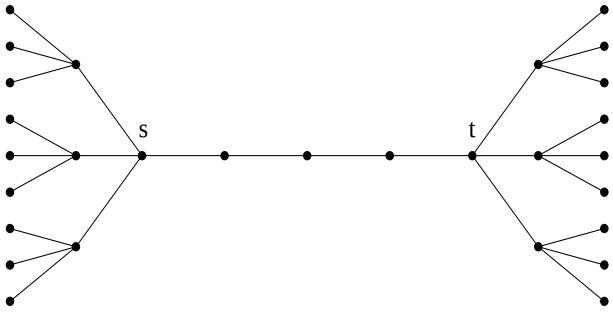

Abbildung 3.3: Gegenbeispiel für *Dijkstra-beidseitig*.

3.3 A^*

Eine andere Modifizierung des Algorithmus *Dijkstra* ist der A^* Algorithmus, bei dem eine geschätzte Entfernung zum Zielknoten verwendet wird. Für die Randknotenmenge wird dabei nicht nur die kürzeste Entfernung zum Startknoten, sondern auch die geschätzte Entfernung zum Zielknoten betrachtet. So verhindert man, dass zu lange in die 'falsche' Richtung gesucht wird [Pea84] [DP85].
Für einen Graphen $G = (V, E, \delta)$ benutzen wir die vollständige Funktion $h : V \times V \to \mathbb{R}^+$ als **Schätzfunktion**. Im A^* Algorithmus werden zwei weitere Entfernungsannäherungen $\delta', \delta'^* : V \to \mathbb{R}^+$ verwendet, wobei δ' die Entfernungsannäherung zum Startknoten und δ'^* die Summe aus der Entfernungsannäherung zum Startknoten und der geschätzten Entfernung zum Zielknoten angibt.
Die Funktion $Update()$ wird um die Entfernungsannäherung δ'^* zu $Update^*()$ erweitert.

Funktion 3 $Update^*(u, \delta'^*, \delta', S)$

$S := S \cup \{u\}$;
for $x \in N(u) \cap (V - S)$ **do** $\delta'(x) := min(\delta'(x), \delta'(u) + \delta(\{u, x\}))$;
$\delta'^*(x) := min(\delta'^*(x), \delta'(x) + h(x, t))$
endfor ;

Algorithmus 3 A^*

Eingabe: $G = (V, E, \delta); s, t \in V; h : V \times V \to \mathbb{R}^+$

Ausgabe: $\delta'(t)$

for $u \in V$ **do** $\delta'(u) := \infty$ **endfor** ;
for $u \in N(s)$ **do** $\delta'(u) := \delta(\{s, u\}); \delta'^*(u) := \delta(\{s, u\}) + h(u, t)$; **endfor** ;
$S := \{s\}$;
while $AR(S) \neq \emptyset$ **and** $t \notin S$ **do**
$u := Next(\delta'^*, S)$;
$Update^*(u, \delta'^*, \delta', S)$
endwhile ;

Die Schätzfunktion $h : V \times V \to \mathbb{R}^+$ heißt **zulässig**, wenn gilt:

$$h(u, v) \quad \leq \quad \delta(u, v) \quad \forall u, v \in V \tag{3.6}$$

3.3.1 Korrektheit

Satz 3 *[HNR68][Gel77]*
Der Algorithmus A^* berechnet für eine zulässige Schätzfunktion die kürzeste Entfernung.

Beweis: Gegeben sei der Graph $G = (V, E, \delta)$, die Schätzfunktion h und eine Knotenmenge $S \subseteq V$, die von A^* berechnet wurde. Offensichtlich berechnet A^* die Länge eines kürzesten Weges in dem von S in G induzierten Graphen $G[S]$. Somit gilt: $\delta'^*(t) = \delta'(t) \geq \delta(s, t)$.

Angenommen, es gibt einen kürzesten Weg $w = [s = x_0, \ldots, x_k = t] \in SP(s, t)$ mit $\delta(w) < \delta'^*(t)$. O.B.d.A. sei x_i der erste Knoten von w, der nicht zu S gehört. Dann gilt:

$$\begin{aligned} \delta'^*(x_i) &= \delta'(x_i) + h(x_i, t) \\ &= \delta(s, x_i) + h(x_i, t) \\ &\leq \delta(s, x_i) + \delta(x_i, t) \\ &= \delta(w) \\ &< \delta'^*(t) \end{aligned}$$

Folglich wäre x_i anstatt t von $Next()$ ausgewählt worden.

■

3.3.2 Laufzeiten

Der A^* Algorithmus ist nur dann gegenüber *Dijkstra* vorteilhaft, wenn die Schätzfunktion nahe an der tatsächlichen Entfernung liegt. Für die zulässige Schätzfunktion $h \equiv 0$ zeigen *Dijkstra* und A^* identisches Verhalten.
Die Anzahl der Knoten, die von A^* für ein Knotenpaar untersucht werden, sei analog zu *Dijkstra* definiert als $A^*(u, v) \in \mathbb{N}$. Judea Pearl hat die Anzahl der untersuchten Knoten von *Dijkstra* und A^* für Gittergraphen berechnet [Pea84]. Wir betrachten den unendlichen Gittergraphen $G = (\mathbb{Z} \times \mathbb{Z}, E, \delta \equiv 1)$ mit

$$E \quad = \quad \{\{(i,j),(i',j')\} \subset 2^{\mathbb{Z}\times\mathbb{Z}} \mid (i = i' \wedge |j - j'| = 1) \vee (|i - i'| = 1 \wedge j = j')\}$$

Dijkstra untersucht alle Knoten in einem Quadrat der Seitenlänge $n + m$. Dies ergibt folgende untersuchte Knotenanzahl (siehe auch Gleichung 2.7):

$$\begin{aligned} Dij((0,0),(n,m)) &= 1 + \sum_{i=1}^{n+m} 4i \\ &= 2(n+m)^2 + 2(n+m) + 1 \end{aligned}$$

Für A^* wird der Betragsabstand als Schätzfunktion verwendet.

$$h((i,j),(i',j')) \quad = \quad |i - i'| + |j - j'|$$

Es folgt nach den Untersuchungen von Pearl:

$$A^*((0,0),(n,n)) \quad \approx \quad 1.436\, n^2$$

Somit untersucht A^* weniger als 18 % der Knoten, die von *Dijkstra* untersucht werden.
Analog zur Modifizierung des Algorithmus *Dijkstra* zu *Dijkstra-beidseitig* lässt sich der A^* Algorithmus ebenfalls zu einer beidseitigen Variante modifizieren. Der Korrektheitsbeweis verläuft analog zu dem obigen Beweis.

Kapitel 4

Andere Modellierungsmethoden

In diesem Kapitel stellen wir einige Modellierungstechniken vor, die in den letzten Jahren entwickelt wurden und insbesondere für eine effiziente Wege- und Entfernungsberechnung interessant sind.

4.1 HiTi Graphen

S. Jung stellt 1995 in seiner Dissertation Hierarchische Multigraphen (HiTi Graphen) für eine effiziente kürzeste Wegesuche (SPSP) vor [Jun95] [JP96]. Ein Graph wird dabei in Teilgraphen zerlegt, und es werden zusätzliche Kanten eingefügt. Teilgraphbäume, die im Folgenden rekursiv definiert werden, stellen die Grundlage des Vorgehens dar.

Definition 1 *Gegeben sei ein Graph $G = (V, E, \delta)$. Der gerichtete Baum T mit Wurzel $r \in V(T)$ und der Abbildung $h : V(T) \rightarrow \{G[W] \mid W \subseteq V\}$ heißt* **Teilgraphbaum**, *wenn gilt:*

1. $h(r) = G$,
2. $V(h(x_1)), \ldots, V(h(x_k))$ *sind eine Partition von V für die direkten Nachfolger $x_1, \ldots, x_k \in N(r) - \{r\}$ von der Wurzel und*
3. *die Teilbäume von T mit Wurzel x_i sind Teilgraphbäume zum Graphen $h(x_i)$.*

Für jeden Knoten $x \in V(T)$ folgt nach der Definition:

$$h(x) \quad = \quad G[V(h(x))]$$

Man beachte, dass die Kanten, die zwischen verschiedenen Teilgraphen eines Levels verlaufen, ausgeblendet werden. Der Übergang vom Level i zum Level

$i + 1$ stellt sozusagen eine verfeinerte Partitionierung der Knoten des zugrunde liegenden Graphen dar, wobei weitere Kanten zwischen den Knotenmengen ausgeblendet werden. In Abbildung 4.1 ist eine Zerlegung eines Graphen mit Teilgraphbaum dargestellt.

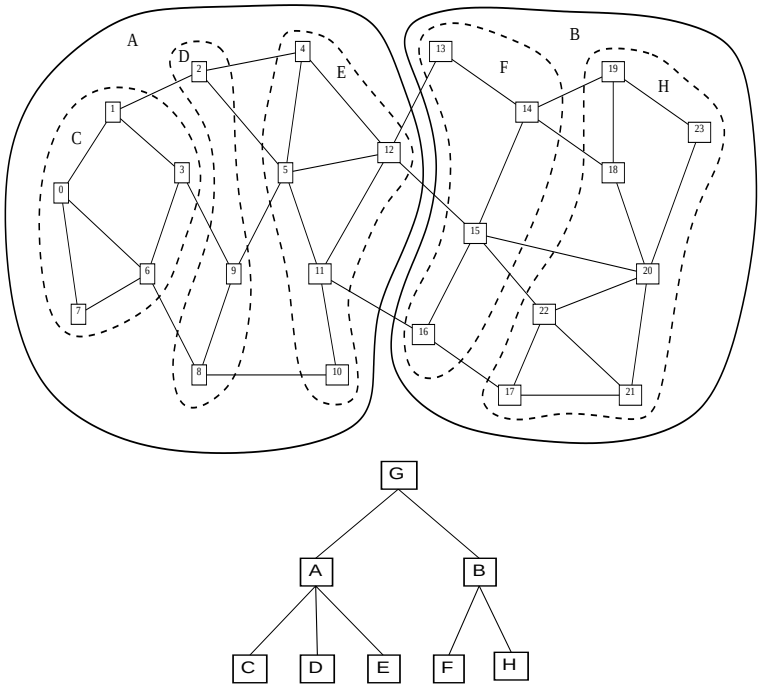

Abbildung 4.1: Beispielzerlegung eines Graphen G mit zugehörigem Teilgraphbaum.

Jung fügt dann zusätzliche Kanten zwischen den Randknoten der Teilgraphen ein, auf denen fast ausschließlich die Wegesuche verläuft. Für ein Knotenpaar $u, v \in V(G)$ beginnt die Wegesuche stets in den Blättern $x, y \in V(T)$ des zugehörigen Teilgraphbaums T mit $u \in V(h(x)), v \in V(h(y))$. Sei T' der Teilbaum von T mit Wurzel $nca(x, y)$. Bei der Berechnung der kürzesten Entfernung werden neben den Teilgraphen $h(x), h(y)$ von G nur die zusätzlichen Kanten untersucht, die von Randknoten von Teilgraphen von T' ausgehen. Anhand einiger Experimente mit 800×800-Gittergraphen mit vier Leveln wird empirisch die Effizienz des Verfahrens gegenüber dem A^* Algorithmus nachgewiesen. Fragen zu Konstruktionsverfahren von HiTi Graphen, so dass beispielsweise nur eine geringe Anzahl von Kanten einzufügen oder bei der Wegesuche nur eine geringe Anzahl von Kanten zu betrachten sind, werden von Jung nicht untersucht.

4.2 Kantenmonotone Levelgraphen

Adrijana Car stellt in ihrer Dissertation im Bereich Geoinformatik 1996 ein hierarchisches Graphmodell zur Berechnung von Näherungsentfernungen (ASPSP) vor, das wir hier als eine Spezialisierung der knotenmonotonen Levelgraphen aus Abschnitt 6.6 darstellen [Car96]. Bereits in Arbeiten von 1993 und 1994 veröffentlichen Car und Frank ähnliche Ideen [CF93], [Car93], [CF94]. In einem k-Levelgraphen $(G_0, \ldots, G_{k-1})$ ist jedes Level dabei ein Teilgraph des niedrigeren Levels (siehe kantenmonotone k-Levelgraphen in Definition 8 auf Seite 65).
Die Entfernungsberechnung für ein Knotenpaar soll möglichst in einem hohen Level und damit in einem 'kleinen' Graphen stattfinden. Als Übergangsknoten zwischen den Leveln werden die jeweils nächsten Knoten aus dem nächsthöheren Level benutzt. Beim Lesen der Arbeiten von Car beachte man, dass dort die Level in genau der umgekehrten Reihenfolge nummeriert werden.
Es wird der folgende Algorithmus zur Wegesuche benutzt, wobei wir annehmen, dass einer der beiden angefragten Knoten bereits im höchsten Level $k-1$ liegt.

Algorithmus 4 *Car Wegesuche*

Eingabe: *Ein kantenmonotoner k-Levelgraph* $(G_0, \ldots, G_{k-1})$ *und die Knoten* $u \in V_0$, $v \in V_{k-1}$.

Ausgabe: $\delta'(u, v)$

$i := \max\{j \in \{0, \ldots, k-1\} \mid u \in V_j\}$;
$\delta'(u, v) := 0$;
$y := u$;
while $i < k-1$ **do**
  $d := \min\{\delta_i(y, x) \mid x \in V_{i+1}\}$; Nur Level i wird betrachtet.
  Es sei $x \in V_{i+1}$ mit $\delta_i(y, x) = d$.
  $\delta'(u, v) := \delta'(u, v) + d$;
  $y := x$;
  $i := i + 1$;
endwhile ;
$\delta'(u, v) := \delta'(u, v) + \delta_{k-1}(y, v)$; Nur Level $k-1$ wird betrachtet.
return $\delta'(u, v)$;

Offensichtlich berechnet dieser Algorithmus i. A. nicht die kürzeste Entfernung. Car macht den Nutzen dieser Vorgehensweise jedoch anhand von zwei Beispielen mit 3-Levelgraphen (28 Knoten, 44 Kanten und 41 Knoten, 81 Kanten) plausibel. Welche Eigenschaften die kantenmonotonen Levelgraphen erfüllen müssen, damit die *Car Wegesuche* die kürzeste Entfernung berechnet und Modifikationen des Algorithmus, wie beispielsweise das Überspringen von Leveln beim Aufsteigen oder die Einbeziehung zusätzlicher 'nächster' Knoten beim Aufsteigen in ein höheres Level, werden von Car nicht untersucht.

4.3 t-Spanner

t-Spanner wurden in [PS89] für ungewichtete Graphen eingeführt und in [ADDJ90] auf gewichtete Graphen erweitert. Ein Teilgraph $G' = (V, E')$ des Graphen $G = (V, E)$ heißt t-Spanner, wenn für alle Knoten $u, v \in V$ gilt:

$$\frac{d_{G'}(u,v)}{d_G(u,v)} \quad \leq \quad t$$

Interessant ist die Berechnung von möglichst kleinen t-Spannern, d.h. mit wenig Kanten im Zusammenhang mit Verteilten Systemen und Kommunikationsnetzwerken.
$S_t(G)$ sei die minimale Anzahl von Kanten für einen t-Spanner von G. In [PS89] wurde gezeigt, dass die Berechnung von t-Spannern der Größe $S_t(G)$ auch für $t = 2$ NP-vollständig ist. Für die Berechnung von $(4k + 1)$-Spannern der Größe $O(n^{1+\frac{1}{k}})$ wurde ein Verfahren angegeben. In [ADDJ90] wurde dieses zu einem Verfahren verbessert, das $(2k + 1)$-Spanner der Größe $n\lceil n^{\frac{1}{k}} \rceil$ berechnet. In jedem Graphen lassen sich 3-Spanner der Größe $\tilde{O}(n^{\frac{3}{2}})$ in $\tilde{O}(m\sqrt{n})$ Operationen erzeugen [DHZ97].
Für bipartite Graphen gilt offensichtlich, dass nur die Graphen selbst ein t-Spanner für $t < 3$ sind (im ungerichteten Fall). Somit gibt es Fälle, in denen jeder 2-Spanner $\Theta(n^2)$ Kanten benötigt.
In [KP92] wird ein Verfahren zur Konstruktion von 2-Spannern der Größe $O(S_2(G) \log \frac{|E|}{|V|})$ vorgestellt. Dieses Verfahren konstruiert Spanner, in denen einzelne Knoten unter Umständen einen hohen Grad besitzen. Wenn man Knoten als Netzwerkkomponenten betrachtet, die Kommunikationsleistungen für ihre Nachbarknoten anbieten, ist man an einem ausgeglichenen Spanner interessiert. Die Berechnung von t-Spannern mit kleinstem Grad heißt LD-tSP (low degree t-spanner) und wird in [LR95] [LS93a] [LS93b] [CDNS92] [Soa94] untersucht. Kortsarz und Peleg zeigen, dass Approximationen von LD-2SP mindestens so schwer sind wie Approximationen von Set-Cover (siehe Seite 55), d.h. auch konstante Approximationen sind NP-vollständig [KP98]. Außerdem stellen sie ein probabilistisches Verfahren zur Approximation mit maximalem Fehler $O(\Delta^{\frac{1}{4}}(G))$ vor.

4.4 Emulatoren

Ein ungewichteter Graph $G' = (V, E')$ heißt ein k-Emulator des ungewichteten Graphen $G = (V, E)$, wenn für alle Knoten $u, v \in V$ gilt:

$$d_G(u,v) \quad \leq \quad d_{G'}(u,v) \quad \leq \quad d_G(u,v) + k$$

Emulatoren unterscheiden sich in drei wesentlichen Punkten von Spannern: Es handelt sich i. A. nicht um Teilgraphen von einem gegebenen Graphen, es wird ein additiver anstatt eines multiplikativen Fehlers betrachtet und es werden keine gewichteten Graphen untersucht.

Dor, Halperin und Zwick stellten 1997 Verfahren vor, die zu jedem ungewichteten Graphen in $\tilde{O}(n^{\frac{5}{2}})$ Operationen einen 2-Emulator der Größe $\tilde{O}(n^{\frac{3}{2}})$ und in $\tilde{O}(n^{\frac{7}{3}})$ Operationen einen 4-Emulator der Größe $\tilde{O}(n^{\frac{4}{3}})$ berechnen [DHZ97].

4.5 Stretch-Paths

Anstatt einen Graphen (t-Spanner) zu berechnen, so dass für alle Knoten die kürzeste Entfernung im t-Spanner höchstens um einen Faktor t (Stretch-Faktor) von der kürzesten Entfernung im Originalgraphen abweicht, kann AAPSP auch direkt durch die Berechnung von Stretch-Paths gelöst werden. Es werden dabei für alle Knoten $u, v \in V(G)$ Entfernungsannäherungen $\delta'(u, v)$ berechnet, die höchstens um einen **Stretch-Faktor** $t \in \mathbb{R}^+$ von der kürzesten Entfernung im Originalgraphen abweichen:

$$\delta(u, v) \quad \leq \quad \delta'(u, v) \quad \leq \quad t\delta(u, v) \tag{4.1}$$

Cohen stellt 1993 ein $\tilde{O}(n^{\frac{5}{2}})$-Verfahren vor, das AAPSP für einen Stretch-Faktor $t = 4 + \epsilon$ in gewichteten ungerichteten Graphen löst [Coh93]. 1997 benutzen Cohen und Zwick eine Partitionierungsmethode und den klassischen *Dijkstra* Algorithmus zur Lösung von AAPSP [CZ97]. Für einen Stretch-Faktor $t = 2$ entwickeln sie ein $\tilde{O}(n^{1.5}\sqrt{m})$-Verfahren, für $t = \frac{7}{3}$ ein $\tilde{O}(n^{\frac{7}{3}})$-Verfahren und für $t = 3$ ein $\tilde{O}(n^2)$-Verfahren. Die Algorithmen benutzen eine quadratische Datenstruktur($\Theta(n^2)$), die für alle Knotenpaare eine Entfernung $\tilde{\delta}(u, v)$ durch sukzessive Anwendungen des *Dijkstra* Algorithmus auf bestimmten Teilgraphen solange anpasst, bis Gleichung 4.1 erfüllt wird.

4.6 Neighborhood-Cover

Zur Berechnung von t-Spannern und Stretch-Paths werden verschiedene Covertechniken eingesetzt. Ein **Cover** C in einem Graphen G entspricht dabei einer Knotenüberdeckung von $V(G)$ durch eine Kollektion von Teilknotenmengen (Clustern) von $V(G)$, die zusammenhängende Teilgraphen von G induzieren. Das Ziel dabei ist, die Größe des Covers ($\sum_{S \in C} |S|$) und die Größe der einzelnen Cluster zu minimieren, so dass möglichst viele Knotenpaare innerhalb eines Clusters liegen. An Stelle der Größe der einzelnen Cluster wird auch der Diameter oder der maximale Grad in dem induzierten Graphen als Parameter benutzt [AP90]. Awerbuch u.a. definieren 1993 in [ABCP93] die Knotenmengen $S_1, \ldots, S_k \subseteq V(G)$ als (β, r)**-Neighborhood Cover** wie folgt ($\beta \in \mathbb{R}^+, r \in \mathbb{N}$):

1. Für jeden Knoten $u \in V(G)$ gibt es ein Cluster S_i mit $N^r(u) \subseteq S_i$ und
2. für den Diameter jedes Clusters $1 \leq i \leq k$ gilt: $diam(S_i) \leq O(\beta r)$.

Wenn die Parameter $\beta \in \mathbb{R}^+$ und $r \in \mathbb{N}$ Konstanten sind, kann der Diameter auch durch eine Konstante beschränkt werden, d.h. durch $O(1)$. Falls diese

Parameter jedoch abhängig von der Anzahl der Knoten sind, ist der Diameter nicht unbedingt durch eine Konstante beschränkt.
Effiziente Berechnungen von Covern wurden von Awerbuch und Peleg in [AP90] eingeführt und seitdem mehrfach verbessert [Coh93] [CZ97].
Mit Neigborhood-Covern der Größe $O(n \log n)$ lassen sich $O(\log n)$ Approximationen von SPSP in $O(\log n \log \log n)$ Operationen berechnen [ABCP93] [ABCP98].
Wir skizzieren grob die Vorgehensweise der Lösung von ASPSP mit Covertechniken anhand von Tree-Covern, die Awerbuch u.a. in [ABCP98] einführen. Wir geben nur den ungewichteten Fall an.

Definition 2 *[ABCP98] Gegeben sei ein Graph $G = (V, E)$ und die Parameter $\beta, r \geq 1$. Ein (β, r)-***Tree-Cover** *$\mathcal{F}_{\beta,r}$ besteht aus Teilgraphen von G, die Bäume sind, so dass gilt:*

1. *Jeder Baum $F \in \mathcal{F}_{\beta,r}$ besitzt höchstens die Tiefe $8\beta r$ und*
2. *für alle Knoten $u, v \in V$ gilt:*

$$d(u,v) \geq r \implies \exists_{F \in \mathcal{F}_{\beta,r}} \quad u, v \in F$$

Ein (β, r)-Tree-Cover heißt **sparse***, wenn jeder Knoten in höchstens $\beta n^{\frac{1}{\beta}}$ Bäumen liegt.*

Bei einem Tree-Cover handelt es sich um eine Abwandlung von Neighborhood-Covern, bei denen die Cluster Bäume sind. In einer Preprocessing-Phase werden $k = \lceil \log Diam(G) \rceil$ sparse Tree-Cover $\mathcal{F}_{\beta,2^i}$ in $O((m + n \log n)(\log n)k)$ Operationen berechnet $(1 \leq i \leq k)$. In der Query-Phase wird für ein gegebenes Knotenpaar $u, v \in V$ mit binärer Suche ein minimales $1 \leq J \leq k$ gesucht, so dass gilt: $u, v \in F \in \mathcal{F}_{\beta,2^J}$. Der Wert $16\beta 2^J$ ist eine 32β-Approximation für den Abstand $d(u, v)$, d.h. es handelt sich um eine $O(\log n)$-Approximation, wenn $\beta = \log n$ gesetzt wird. Die Query-Phase wird in diesem Fall in $O(\log n \log k) = O(\log n \log \log n)$ Operationen durchgeführt.

Kapitel 5

Baumstrukturen

Wir zeigen in diesem Kapitel, dass für Bäume und für Graphen ohne Kreise der Länge größer als drei gilt:

$$\begin{aligned} \mathcal{S}, \mathcal{T} &\in O(n) \\ \mathcal{Q} &\in O(1) \end{aligned}$$

Anschließend betrachten wir Zerlegungen von Graphen in ihre 2-zusammenhängenden Komponenten.

5.1 Kürzeste Wege in Bäumen

In Bäumen lassen sich Entfernungen in konstanter Zeit berechnen, wenn eine Preprocessing-Phase mit linearer Zeit auf linearem Platz vorangestellt wird.
Dazu transformieren wir einen ungerichteten Baum in einen gerichteten Baum $T = (V, E, \delta)$ mit gleicher Kantengewichtsfunktion δ und Wurzel $r \in V$, die beliebig gewählt werden kann. Für jeden Knoten $u \in V$ berechnen und speichern wir die Entfernung zur Wurzel: $\delta_r(u) = \delta(u, r)$.

Lemma 3 *Sei $T = (V, E, \delta)$ ein gerichteter Baum mit Wurzel $r \in V(T)$, Kantengewichtsfunktion δ und Knotengewichtsfunktion $\delta_r(u) = \delta(u, r)$, dann gilt für alle Knoten $u, v \in V(T)$:*

$$\delta(u, v) = \delta_r(u) + \delta_r(v) - 2\delta_r(nca(u, v))$$

Wenn der Baum in einer Preprocessing-Phase von linearer Zeit und linearem Platz vorbehandelt wird, lässt sich $nca()$ in konstanter Zeit berechnen [HT84]. Nach Lemma 3 lässt sich folglich die kürzeste Weglänge in konstanter Zeit berechnen, wobei wir ein Registermaschinenmodell und damit eine uniforme Zeitkomplexität verwenden [Weg93].
Ein kürzester Weg der Länge k kann in $O(k)$ Schritten ausgegeben werden. Dazu ist für jeden Knoten zusätzlich ein Zeiger auf den Vorgänger zu verwalten.

Theorem 1 *In Bäumen können Entfernungen in konstanter Zeit und kürzeste Wege in $O(k)$ Schritten berechnet werden, wenn eine Preprocessing-Phase von linearer Zeit und linearem Platz vorangestellt wird (k sei die Länge des kürzesten Weges).*

5.2 Kreisfreie Graphen

Wir betrachten nun Graphen, für die die Länge des längsten Kreises beschränkt ist.

$$\mathcal{G}_k \quad = \quad \{G \in \mathcal{G} \mid C(G) \leq k\}$$

$\mathcal{G}_0$ sind genau die Graphen ohne Kanten, $\mathcal{G}_1$ die Graphen, die außer Schlingen keine Kanten enthalten, und $\mathcal{G}_2$ die ungerichteten Bäume.
Offensichtlich gilt die folgende Inklusionseigenschaft: $\mathcal{G}_k \subset \mathcal{G}_{k+1}$ für $k \in \mathbb{N}_0$.

Satz 4 *Für jeden Graphen $G \in \mathcal{G}_3$ gibt es einen längengleichen Baum bzgl. $V(G)$.*

Beweis: Gegeben sei $G = (V, E, \delta) \in \mathcal{G}_3$. Transformiere G in den Graphen $G' = (V \cup V', E', \delta')$ wie folgt. Übernehme alle Kanten $e \in E$, die zu keinem Dreieck gehören, in E' und behalte die Kantengewichtsfunktion bei: $\delta'(e) = \delta(e)$. Für jedes Dreieck $[u, v, w]$ in G füge einen Knoten x_{uvw} in V' und die Kanten $\{u, x_{uvw}\}, \{v, x_{uvw}\}, \{w, x_{uvw}\}$ in E' mit folgenden Kantengewichten ein (s. Abbildung 5.1):

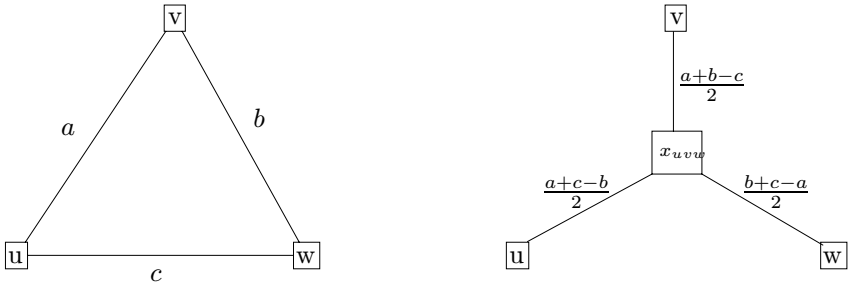

Abbildung 5.1: Entfernen eines Dreiecks.

$$\begin{aligned} \delta'(\{u, x_{uvw}\}) &= \frac{\delta(\{u,v\}) + \delta(\{u,w\}) - \delta(\{v,w\})}{2} \\ \delta'(\{v, x_{uvw}\}) &= \frac{\delta(\{u,v\}) + \delta(\{v,w\}) - \delta(\{u,w\})}{2} \end{aligned}$$

$$\delta'(\{w, x_{uvw}\}) \quad = \quad \frac{\delta(\{u,w\}) + \delta(\{v,w\}) - \delta(\{u,v\})}{2}$$

Da es in G zwischen zwei Endknoten eines Dreiecks keinen anderen doppelpunktfreien Weg als über das Dreieck selbst gibt, ist G' ein Baum. Die Längengleichheit ($\delta \equiv_V \delta'$) ergibt sich daraus, dass jede Kante aus E entweder auch in E' liegt oder einem Weg mit zwei Kanten und gleicher Länge in G' entspricht.

■

Korollar 1 *In den Graphen $\mathcal{G}_3$ gilt: $S, T \in O(n), Q \in O(1)$.*

Beweis: $Q \in O(1)$ folgt nach Theorem 1 und Satz 4.

Sei $h(n)$ die maximale Anzahl von Dreiecken in einem Graphen $G \in \mathcal{G}_3$ mit n Knoten. Weil sich zwei Dreiecke in höchstens einem Knoten schneiden, gilt für $n \in \mathbb{N}$:

$$\begin{aligned} h(n) &= h(n-2) + 1 \\ &= \lfloor \frac{n-1}{2} \rfloor \end{aligned}$$

Somit besitzt jeder Graph $G \in \mathcal{G}_3$ höchstens $n + \lfloor \frac{n-3}{2} \rfloor$ Kanten. Konstruiere in Linearzeit einen spannenden Baum T von G. Jede Kante $e \in E(G) - E(T)$ bildet ein Dreieck, das durch Einfügen eines neuen Knotens mit konstantem Aufwand (siehe Beweis zu Satz 4) entfernt wird.

■

Satz 5 *Für Graphen $G \in \mathcal{G}_k$ gibt es i.A. keinen längengleichen Graphen $G' \in \mathcal{G}_{k-1}$ für $k > 3$.*

Beweis: Wir zeigen, dass es keinen längengleichen Graphen aus $\mathcal{G}_{k-1}$ zum Kreisgraphen $C_k = (\{1, 2, \ldots, k\}, \{\{i, i+1\} \mid 1 \leq i < k\} \cup \{1, k\}, \delta \equiv 1) \in \mathcal{G}_k$ gibt. Sei G ein längengleicher Graph zu C_k. O.B.d.A. liegen die Knoten $1, \ldots, i$ auf einem doppelpunktfreien Weg $w' = [1, \ldots, 2, \ldots, i-1, \ldots, i]$ in G mit $\delta(w') = i - 1$. Aufgrund der Längengleichheit zu C_k muss der Knoten $i+1$ auf einer Verlängerung von w' liegen, d.h. $w'' = w' \circ [i, \ldots, i+1]$ ist ebenfalls doppelpunktfrei mit $\delta(w'') = i$. Somit liegen alle Knoten $1, 2, \ldots, k$ auf einem doppelpunktfreien Weg w in G. Wegen $\delta(1, k) = 1$ muss es bis auf Anfangs- und Endknoten einen zu w knotendisjunkten Weg in G geben. Daraus folgt $G \notin \mathcal{G}_{k-1}$.

■

Man beachte, dass Kreisfreiheit und Baumweite (siehe Definition 18) verschiedene Eigenschaften von Graphen sind. Beispielsweise hat der C_n für $n \geq 3$ die Baumweite $B(C_n) = 2$, allerdings gilt: $C_n \notin \mathcal{G}_{n-1}$.

5.3 2-Zusammenhangskomponenten

Ein Knoten $u \in V(G)$ eines Graphen G heißt **Artikulationsknoten**, wenn die Anzahl der Zusammenhangskomponenten von $G-u$ größer ist als die Anzahl der Zusammenhangskomponenten von G. Ein zusammenhängender Graph (oder eine Zusammenhangskomponente) heißt **2-zusammenhängend**, wenn es keinen Artikulationsknoten in dem Graphen (bzw. in der Zusammenhangskomponente) gibt. Eine 2-zusammenhängende Zusammenhangskomponente bezeichnen wir auch als 2-Zusammenhangskomponente [Har69].
Die 2-Zusammenhangskomponenten und sämtliche Artikulationsknoten eines Graphen ordnen wir im Folgenden in einem Baum an. Stiege bezeichnet diesen Baum mit einer etwas 'feineren' Struktur als Biblock Tree [Sti97] [Sti98]. Harary ordnet die 2-Zusammenhangskomponenten in einem sogenannten Gliedergraphen (ein Glied entspricht dabei genau einer 2-Zusammenhangskomponente) und die Artikulationsknoten in einem Artikulationsgraphen an [Har69].
Zusätzlich speichern wir in einer Preprocessing-Phase die Entfernungen von allen Knoten aus dem zugrunde liegenden Graphen zu einem ausgewählten Artikulationsknoten und zeigen so ein ähnliches Resultat wie Theorem 1.

Definition 3 *Gegeben sei ein zusammenhängender Graph $G = (V, E, \delta)$ mit den 2-Zusammenhangskomponenten $Z = \{G_1, \ldots, G_k\}$ und den Artikulationsknoten $A \subseteq V$. Der bipartite Graph $T = (A, Z, E)$ heißt* **2-Baum** *von G, wenn für alle Knoten $a \in A$ und $g \in Z$ gilt:*

$$\{a, g\} \in E \iff a \in V(g)$$

Lemma 4 *[Bra94] Jeder 2-Baum ist ein Baum.*

In Abbildung 5.2 ist ein Graph mit 2-Baum dargestellt.

Theorem 2 *Unter der Voraussetzung, dass die Entfernungen innerhalb einer 2-Zusammenhangskomponente eines Graphen in konstanter Zeit bestimmt werden können, genügt eine fast lineare Preprocessing-Phase, um jede Entfernung in konstanter Zeit zu berechnen.*

Beweis: Zu einem Graphen können in Linearzeit alle Artikulationsknoten und 2-Zusammenhangskomponenten und somit auch ein 2-Baum bestimmt werden [AHU74]. Sei $T = (A, Z, E)$ ein 2-Baum zum Graphen G. Wähle einen beliebigen Artikulationsknoten $r \in A$ als Wurzel von T und berechne mit dem *Dijkstra* Algorithmus in $O(m + n \log n)$ Operationen die Entfernungen $\delta_r(u) = \delta(u, r)$ für alle Knoten $u \in V(G)$ (SSSD).

Für die Knoten $u, v \in V(G)$, die in verschiedenen 2-Zusammenhangskomponenten $g, g' \in Z$ liegen ($u \in V(g), v \in V(g')$), unterscheiden wir die beiden folgenden Fälle (siehe Abbildung 5.3).

$nca(u, v) \in A$: Da alle Wege zwischen u und v über den Artikulationsknoten $nca(u, v)$ verlaufen, gilt:

$$\delta(u, v) = \delta_r(u) + \delta_r(v) - 2\delta_r(nca(u, v))$$

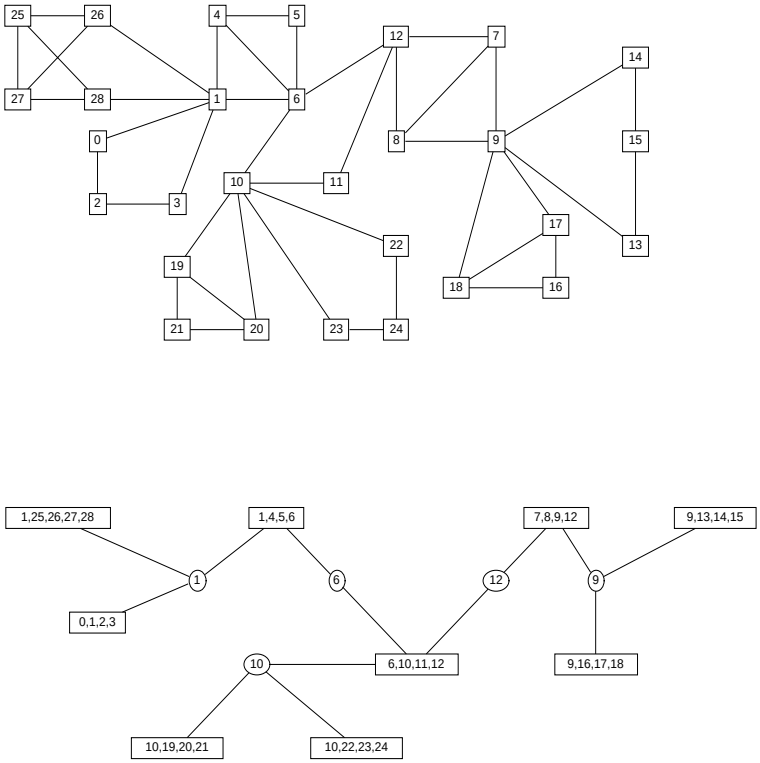

Abbildung 5.2: Graph mit 2-Baum.

$nca(u, v) \in Z$**:** Es seien $x, y \in N_T(nca(u, v))$ Artikulationsknoten mit $g \in N_T^*(x)$ und $g' \in N_T^*(y)$. Die Knoten x, y können in T in konstanter Zeit bestimmt werden. Dann folgt:

$$\delta(u, v) \quad = \quad \delta_r(u) - \delta_r(x) + \delta(x, y) + \delta_r(v) - \delta_r(y)$$

Unter der Voraussetzung, dass die Entfernungen innerhalb einer 2-Zusammenhangskomponente in konstanter Zeit bestimmbar sind, kann folglich die Entfernung für jedes Knotenpaar in konstanter Zeit bestimmt werden.

■

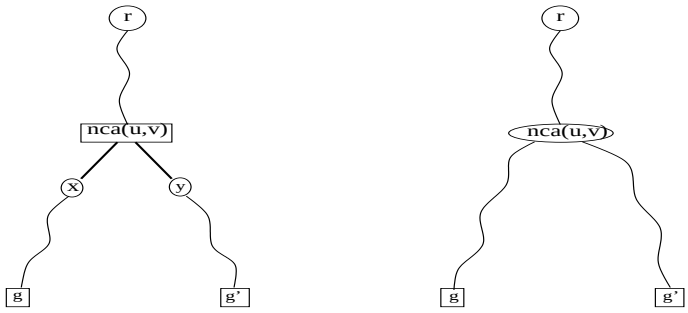

Abbildung 5.3: Links ist der Fall $nca(u, v) \in Z$ und rechts $nca(u, v) \in A$ dargestellt, wobei Artikulationsknoten Kreise sind.

Kapitel 6

Geeignete Knotenmengen und Levelgraphen

In diesem Kapitel stellen wir einen Modellierungsansatz vor, der auf folgender Annahme beruht:

> Ausgehend von einem Knoten u verläuft jeder längere kürzeste Weg nur über einige wenige sogenannte geeignete Knoten aus der näheren Umgebung von u, und sämtliche geeigneten Knoten aus den näheren Umgebungen von allen Knoten bilden eine 'kleine' Menge.

Für die geeigneten Knoten werden die kürzesten Entfernungen vollständig tabelliert. Für die restlichen Knoten werden nur Kanten zu geeigneten Knoten aus der näheren Umgebung eingefügt. Der geeignete hierarchische Graph besteht aus zwei Leveln. Im ersten Level befindet sich der Originalgraph, und im zweiten Level werden der vollständige Teilgraph und die zusätzlichen Kanten verwaltet. Die Entfernungsberechnung erfolgt für nah beieinander liegende Knoten im ersten und für die anderen Knotenpaare im zweiten Level. In Abschnitt 6.6 erweitern wir diesen Ansatz auf k-Levelgraphen.

6.1 Geeignete Knoten

Definition 4 *Gegeben seien ein gewichteter Graph $G = (V, E, \delta)$, ein Abstand $s \in \mathbb{N}_0$, ein Parameter $\xi \in \mathbb{R}^+$ (bezeichnet den Fehler) und eine Knotenmenge $S \subseteq V$. Ein Knotenpaar $u, v \in V$ heißt (S, s, ξ)-**geeignet**, wenn gilt:*

$$\delta(u, N^s(u) \cap S, N^s(v) \cap S, v) \quad \leq \quad \delta(u, v) + \xi \tag{6.1}$$

(Die Erweiterung von Funktionen wird im Kapitel 2 in Gleichung 2.2 definiert. Bei einelementigen Mengen entfallen die Mengenklammern.)
Wir bezeichnen den Parameter $s \in \mathbb{N}$ auch als den **Umgebungsabstand**. *Die Knotenmenge S heißt (s, ξ)-**geeignet**, wenn alle Knotenpaare $u, v \in V$ mit*

$d(u,v) > s$ *: (S,s,ξ)-geeignet sind. Die* **maximale Umgebungsgröße** *vom Umgebungsabstand s und der Knotenmenge S ist definiert als:*

$$\upsilon(s,S) \quad = \quad \max(1, \max\{|N^s(u) \cap S| \mid u \in V - S\}) \tag{6.2}$$

Man beachte, dass für die fortgesetzte Entfernungsfunktion δ für alle Knoten $u, v \in V$ und alle Knotenmengen $A, B, C, D, W \subseteq V$ mit $A \subseteq B, D \subseteq C$ gilt:

$$\begin{aligned} \delta(u, \emptyset, W, v) &= \infty \\ \delta(u, W, \emptyset, v) &= \infty \\ \delta(A, B, C, D) &= \delta(A, D) \end{aligned}$$

Für den Fall $u \in A, v \in B$ folgt nach der letzten Ungleichung:

$$\delta(u, A, B, v) \quad = \delta(u, v)$$

Jedes Knotenpaar, das nicht durch einen Weg verbunden ist ($P(u,v) = \emptyset$), ist nach Gleichung 6.1 für jede Knotenmenge S und jeden Wert $s \in \mathbb{N}$ $(S, s, 0)$-geeignet, weil die Entfernung in diesem Fall unendlich ist.
Die gesamte Knotenmenge $V(G)$ eines Graphen G ist beispielsweise $(0,0)$-geeignet, weil in diesem Fall für alle Knoten $u, v \in V(G)$ gilt:

$$\delta(u, u, v, v) \quad = \quad \delta(u, v).$$

Bei zusammenhängenden Graphen gilt sogar, dass keine echte Teilmenge der gesamten Knotenmenge $(0,0)$-geeignet ist. Die maximale Umgebungsgröße υ wird bei dem Aufwand für die Entfernungsberechnung relevant. Weil die Entfernungen für Knotenpaare aus S bereits in der Preprocessing-Phase berechnet worden sind, werden nur Umgebungsknoten für Knoten $u \in V - S$ betrachtet.

Lemma 5 *Wenn $s \in \mathbb{N}_0$ und $S \subseteq V$ so gewählt werden, dass für alle Knoten $u \in V$ gilt: $N^s(u) \cap S \neq \emptyset$, dann ist S stets $(s, 4\max\{\delta(u,S) \mid u \in V\})$-geeignet (siehe Abbildung 6.1).*

Zur Veranschaulichung obiger Definition betrachten wir ein $k \times k$-Gitter $G = (\{(x,y) \in \{0,1,\ldots,k-1\}^2\}, E, \delta \equiv 1)$ und die Knotenmenge

$$S(r,k) \quad = \quad \{(x,y) \in \{0,1,\ldots,k-1\}^2 \mid 0 = (x+y) \bmod r\} \tag{6.3}$$

Für $r = 1$ gilt: $S(1,k) = \{0,1,\ldots,k-1\}^2$ und für $r \geq 2k-1$ gilt: $S(r,k) = \{(0,0)\}$. Die Größe der Knotenmenge $S(r,k)$ lässt sich wie folgt abschätzen:

$$\begin{aligned} |S(r,k)| &\leq 2 \sum_{i=0}^{\frac{k-1}{r}-1} (i \cdot r + 1) + k \\ &= 2\frac{k-1}{r} + \frac{(k-1)^2}{r} - (k-1) + k \\ &= \frac{k^2-1}{r} + 1 \end{aligned} \tag{6.4}$$

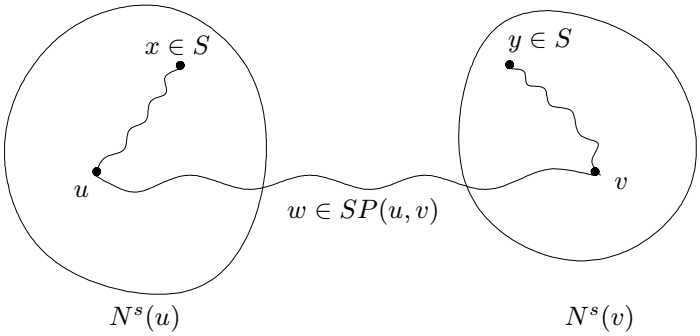

Abbildung 6.1: Maximaler Fehler mit geeigneten Knoten S.

$S(r,k)$ nimmt die maximale Größe für den Fall $0 = (k-1) \bmod r$ an.
In Abbildung 6.2 ist ein 10×10-Gitter dargestellt, wobei die markierten Knoten die Knotenmenge $S(3,10)$ bilden, die beispielsweise wegen den Knoten $(1,0)$ und $(4,0)$ nicht $(1,0)$-geeignet ist. Allerdings ist $S(3,10)$ $(2,0)$- und $(1,4)$-geeignet. Jede Knotenmenge $S(r,k)$ ist offensichtlich $(r,0)$-geeignet. Als maximale Umgebungsgrößen ergeben sich für das 10×10-Gitter $\upsilon(1,S(3,10)) = 2$, $\upsilon(2,S(3,10)) = 5$ und $\upsilon(3,S(3,10)) = 7$.

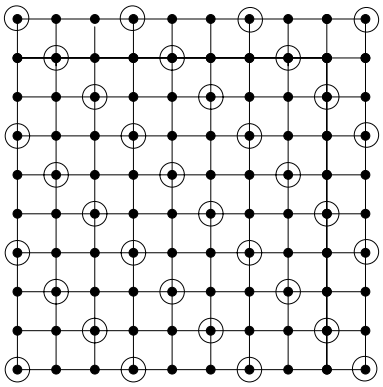

Abbildung 6.2: Ein 10×10-Gitter mit geeigneter Knotenmenge.

Definition 5 *Gegeben seien ein Graph $G = (V,E,\delta)$ und eine (s,ξ)-geeignete Knotenmenge $S \subseteq V$. Der Graph $G' = (V,E',\delta')$ heißt* (S,s,ξ)**-geeigneter Graph** *zu G mit*

$$\begin{aligned} E' &= \{\{u,v\} \subset S\} \cup \{\{u,v\} \subset V \mid d(u,v) \leq s, |\{u,v\} \cap S| = 1\} \\ \delta'(\{u,v\}) &= \delta(u,v) \qquad \forall \{u,v\} \in E' \end{aligned}$$

In einem (S, s, ξ)-geeigneten Graphen $G' = (V, E', \delta')$ zu $G = (V, E, \delta)$ gilt für alle Knotenpaare $u, v \in V$ mit Umgebungsabstand $d(u, v) > s$ (im Graph G):

$$\delta'(u, v) \quad \leq \quad \delta(u, v) + \xi$$

Die Entfernungsberechnung werden wir für die Knotenpaare mit kleinem Umgebungsabstand ($d(u, v) \leq s$) im 'Originalgraphen' und für die anderen Knotenpaare ($d(u, v) > s$) im geeigneten Graphen durchführen.

Satz 6 *Gegeben seien ein Graph $G = (V, E, \delta)$ und ein (S, s, ξ)-geeigneter Graph $G' = (V, E', \delta')$. Eine fast kürzeste Entfernung bis auf einen Fehler ξ lässt sich in $O((N_{\max}^{\lceil \frac{s}{2} \rceil + 1})^2 + (\upsilon(s, S))^2)$ Operationen berechnen. Dabei ist $N_{\max}^s = \max\{|N^s(u)| \mid u \in V\}$ und N^s die Nachbarschaftsrelation in dem ungewichteten Graphen (V, E).*

Beweis: Es sei die fast kürzeste Entfernung für das Knotenpaar $u, v \in V$ gesucht. Wir werten die beiden folgenden Fälle parallel zueinander aus.

- $d(u, v) \leq s$ **:** Der Algorithmus *Dijkstra-beidseitig* findet in G die kürzeste Entfernung, wobei die Knotenmengen S_u und S_v nur bis $N^{\lceil \frac{s}{2} \rceil}(u)$ und $N^{\lceil \frac{s}{2} \rceil}(v)$ aufgebaut werden müssen. N sei die Nachbarschaftsrelation in dem ungewichteten Graphen (V, E), die sich von der Nachbarschaftsrelation in G unterscheidet (siehe Lemma 1). Dies erfolgt in $O((N_{\max}^{\lceil \frac{s}{2} \rceil + 1})^2)$ Operationen (siehe Kapitel 3).
- $d(u, v) > s$ **:** Weil G' (S, s, ξ)-geeignet ist, gibt es für die Knoten u und v Kanten zu allen geeigneten Knoten aus $N^s(u)$ bzw. $N^s(v)$. Die fast kürzeste Entfernung bis auf einen Fehler ξ wird durch Auswertung von $\delta(u, N^s(u) \cap S, N^s(v) \cap S, v)$ in $|N^s(u)||N^s(v)|$ Operationen bestimmt (siehe Gleichung 6.1). Wegen $|N^s(u)|, |N^s(v)| \leq \upsilon(s, S)$ folgt der zweite Summand.

■

Es ist im ersten Fall notwendig, die Entfernungen für alle Knoten bis zum Umgebungsabstand $\lceil \frac{s}{2} \rceil$ im ungewichteten Graphen (V, E) zu bestimmen, weil zur Abstandsberechnung in dem gewichteten Graphen im worst-case der gesamte Graph untersucht werden muss. Zur Veranschaulichung von Satz 6 betrachten wir wiederum die $k \times k$-Gitter mit den $(r, 0)$-geeigneten Knotenmengen $S(r, k)$ (siehe Gleichung 6.3). Für die Größe von $S(r, k)$ und von der Umgebungsgröße $\upsilon(r, S(r, k))$ folgt nach den Gleichungen 6.4 und 6.2:

$$\begin{aligned} |S(r, k)| &\in O(\frac{k^2}{r}) \\ \upsilon(r, S(r, k)) &\leq 4r \\ &\in O(r) \end{aligned}$$

Damit ist folgender Platz für den $(S(r,k), r, 0)$-geeigneten Graphen $G' = (V, E', \delta')$ zu reservieren:

$$\begin{aligned} |E'| &\leq |S(r,k)|^2 + k^2 \upsilon(r, S(r)) \\ &\in O(\frac{k^4}{r^2}) + O(k^2 r) \end{aligned}$$

Die Entfernungsberechnung für ein Knotenpaar kann nach Satz 6 in einer Laufzeit von $O(r^4)$ ausgeführt werden, wobei $N^r_{\max} \in O(r^2)$ gilt (siehe Gleichung 2.7). $k \times k$-Gitter sind dünne Graphen, somit benötigt der *Dijkstra* Algorithmus für den Fall, dass die Knoten nah beieinander liegen ($d(u,v) \leq r$), höchstens den Aufwand $O(r^2 \log r)$. Für den Fall $k = r^2$ ergibt sich für ein $k \times k$-Gitter mit $n = k^2$ Knoten in der Terminologie $\mathcal{T}, \mathcal{S}, \mathcal{Q}$:

$$\begin{aligned} \mathcal{S} &\in O(k^3) \\ &= O(n^{1.5}) \\ \mathcal{Q} &\in O(k \log k) \\ &= O(\sqrt{n} \log n) \end{aligned}$$

Eine Minimierung des Speicherplatzes $\mathcal{S}$ ergibt sich für $k^{\frac{2}{3}} = r$:

$$\begin{aligned} \mathcal{S} &\in O(k^{\frac{8}{3}}) \\ &= O(n^{\frac{4}{3}}) \\ \mathcal{Q} &\in O(k^{\frac{4}{3}} \log k) \\ &= O(n^{\frac{2}{3}} \log n) \end{aligned}$$

6.2 Problemkomplexität

Wir definieren das Problem, eine geeignete Knotenmenge zu berechnen, für eine gegebene Fehlergröße $\xi \in \mathbb{R}^+$.

Problem 1: ***Hitting Shortest Paths Set HSPS***(ξ)

Eingabe: *Graph* $G = (V, E, \delta)$, $s \in \mathbb{N}_0$, $l \in \mathbb{N}$

Frage: *Existiert eine* (s, ξ)*-geeignete Knotenmenge* $S \subseteq V$ *mit* $|S| \leq l$*?*

Das Problem HSPS(0) bezeichnen wir auch als HSPS.

Satz 7 *HSPS ist NP-vollständig.*

Beweis: HSPS liegt in NP, weil es weniger als 2^n Knotenmengen gibt, von denen wir eine raten und in kubischer Zeit (siehe Algorithmus *Eignungstest* auf Seite 50) überprüfen, ob sie $(s, 0)$-geeignet ist.

Die NP-Härte zeigen wir durch Reduktion von VC auf HSPS.

Problem 2: ***Vertex Cover VC***

Eingabe: *Graph $G = (V, E)$ und $k \in \mathbb{N}$*

Frage: *Gibt es eine Knotenüberdeckung $V' \subseteq V$ mit $|V'| \leq k$, so dass von jeder Kante $e \in E$ mindestens ein Knoten in V' liegt ($e \cap V' \neq \emptyset$)?*

Karp zeigte 1972 die NP-Vollständigkeit von Vertex Cover durch Reduktion von 3SAT [Kar72].

Vertex Cover bleibt NP-hart, wenn nur noch Graphen ohne Kreise der Länge kleiner gleich vier zugelassen sind. Dies kann beispielsweise dadurch erreicht werden, dass jede Kante durch drei Kanten ersetzt wird. Der ursprüngliche Graph mit m Kanten hat genau dann ein VC der Größe k, wenn der modifizierte Graph mit $3m$ Kanten ein VC der Größe $k + m$ besitzt.

Gegeben sei ein zusammenhängender Graph $G = (V, E, \delta \equiv 1)$ ohne Kreise der Länge kleiner gleich vier. Für jede Knotenmenge $S \subseteq V$ gilt:

$$S \text{ ist } (1,0) - \text{geeignet} \iff S \text{ ist ein } VC \text{ in } G$$

$\Longrightarrow$ Jede Kante $e = \{u, v\} \in E$ ist Anfangsstück eines kürzesten Weges, weil es keine Kreise der Länge kleiner gleich vier gibt. Weil S $(1,0)$-geeignet ist, folgt $e \cap S \neq \emptyset$. Man beachte, dass man auf die Eigenschaft des Fehlens von Kreisen der Länge kleiner gleich vier nicht verzichten kann, wie der Graph in Abbildung 6.3 zeigt. Die Knoten $\{a, b\}$ bilden zwar eine $(1,0)$-geeignete Knotenmenge, jedoch kein Vertex Cover.

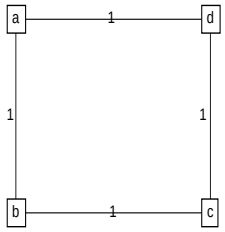

Abbildung 6.3: Ein 2×2-Gitter mit geeigneter Knotenmenge {a,b}.

$\Longleftarrow$ Jede Anfangs- und Endkante eines kürzesten Weges $w \in SP$ wird von mindestens einem Knoten aus S überdeckt. Somit ist das Knotenpaar $s(w), e(w)$ $(S, 1, 0)$-geeignet.

Somit gibt es in G genau dann ein VC der Größe $k \in \mathbb{N}$, wenn es eine $(1,0)$-geeignete Knotenmenge derselben Größe gibt.

■

Korollar 2 *HSPS(ξ) ist für jedes $\xi \in \mathbb{R}^+$ NP-vollständig.*

6.3 Algorithmen

Wir stellen in diesem Abschnitt einen Eignungstest für eine gegebene Knotenmenge und einen Graphen sowie einen Algorithmus zur Berechnung einer approximativen Lösung von HSPS vor, die höchstens um einen logarithmischen Faktor vom Optimum abweicht.

Definition 6 *Gegeben seien ein zusammenhängender Graph $G = (V, E, \delta)$ und ein Umgebungsabstand $s \in \mathbb{N}_0$. Die Menge zu* **beachtender Knotenpaare** *ist:*

$$M(s) \quad = \quad \{(x, y) \in V \times V \mid d(x, y) > s\}$$

und die Menge der vom Knoten $u \in V$ **beachteten Knotenpaare** *ist:*

$$M(u, s) \quad = \quad \{(x, y) \in M(s) \mid u \in \bigcup_{w \in SP(x,y)} V(w) \cap N^s(x)\}$$

Weil es für jedes Knotenpaar $(x, y) \in M(s)$ einen kürzesten Weg $w \in SP(x, y)$ mit mehr als s Kanten gibt (siehe Gleichung 2.10 in Kapitel 2), folgt für jeden Knoten $u \in V(Pr(w, s))$:

$$(x, y) \quad \in \quad M(u, s)$$

Für jedes $s \in \mathbb{N}_0$ gilt somit:

$$M(s) \quad = \quad \bigcup_{u \in V} M(u, s)$$

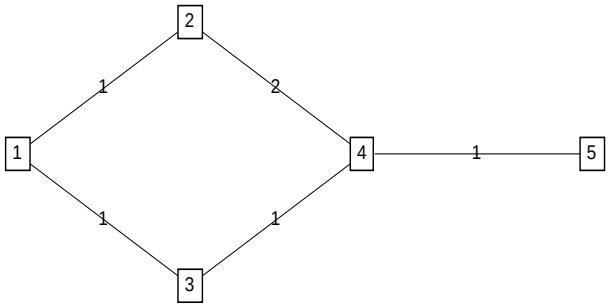

Abbildung 6.4: Beispielgraph zu Definition 6.

Man veranschaulicht sich die Definition 6 am Beispielgraphen in Abbildung 6.4, für den sich für $s = 1$ die folgenden zu beachtenden Knotenpaare ergeben:

$$M(1) \quad = \quad \{(1,4), (1,5), (2,3), (2,5), (3,2), (3,5), (4,1), (5,1), (5,2), (5,3)\}$$

6.3.1 Eignungstest

Algorithmus 5 *Eignungstest (für den Fehler $\xi = 0$)*

Eingabe: $G = (V, E, \delta)$, $S \subseteq V$, $G' = (V, E', \delta')$, $D \subseteq V \times V$

Ausgabe: {*true, false*}

```
for (u, v) ∈ D mit |{u, v} ∩ S| ≤ 1 do
    if δ(u, v) ≠ δ'(u, v) then return false; stop ; endif ;
endfor ;
return true;
```

Beim Aufruf von Algorithmus *Eignungstest* mit den Knotenpaaren $D = M(s)$ wird genau dann *true* zurückgegeben, wenn G' $(S, s, 0)$-geeignet ist. In der if-Bedingung müssen nur die Knotenpaare überprüft werden, bei denen nicht beide Knoten aus S sind, weil S in G' per Definition vollständig tabelliert ist. Hingegen ist es nicht ausreichend, nur Knotenpaare $(u, v) \in D$ mit $\{u, v\} \cap S = \emptyset$ zu überprüfen. Man macht sich dies an dem Graphen in Abbildung 6.5 mit der Menge $S = \{a\}$ und der Menge zu beachtender Knotenpaare $M(1) = \{(a, c), (c, a)\}$ deutlich. S ist in diesem Fall nicht $(1, 0)$-geeignet.

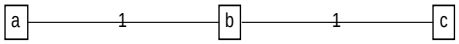

Abbildung 6.5: Beispiel zum Eignungstest.

Das Laufzeitverhalten vom *Eignungstest* ist im worst-case von kubischer Größenordnung. Zum einen werden die Entfernungen im 'Originalgraphen' G mit dem *Dijkstra* Algorithmus für alle Knoten in $O(nm + n^2 \log n)$ Operationen ermittelt. Zur Entfernungsberechnung in G' wird nur über die Randknoten von den Knoten $V - S$ minimiert. Dabei sei

$$d_S \quad = \quad \max\{d_{G'}(u) \mid u \in V - S\}$$

Man beachte, dass $d_S = v(s, S)$ für einen $(S, s, 0)$-geeigneten Graphen G' gilt. Daraus ergeben sich für jedes Knotenpaar maximal $O(d_S^2) \subseteq O(n^2)$ Vergleiche und somit insgesamt höchstens eine Laufzeit $O(n^3)$. Wir geben im Folgenden eine Graphfamilie G_n für gerade $n \in \mathbb{N}$ an, für die jede minimale geeignete Knotenmenge S und die Größe d_S stets von linearer Größenordnung sind.

$$\begin{aligned} G_n &= (V_n, E_n, \delta \equiv 1) \\ V_n &= \{a, b\} \cup \{(i, j) \in \mathbb{N} \times \mathbb{N} \mid 1 \le i \le 2, 1 \le j \le \frac{n}{2} - 1\} \\ E_n &= \{\{a, (1, j)\} \mid 1 \le j \le \frac{n}{2} - 1\} \cup \{\{b, (2, j)\} \mid 1 \le j \le \frac{n}{2} - 1\} \\ &\quad \cup \{\{(1, j), (2, j)\} \mid 1 \le j \le \frac{n}{2} - 1\} \end{aligned}$$

In Abbildung 6.6 ist der Graph G_{10} dargestellt.

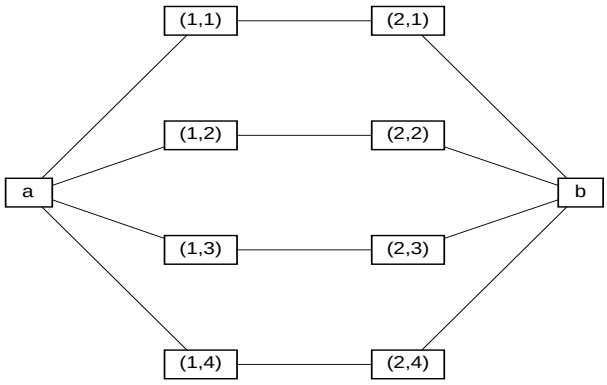

Abbildung 6.6: Der Graph G_{10}.

Satz 8 *Jede $(1,0)$-geeignete Knotenmenge im Graphen G_n für gerade $n \in \mathbb{N}$ hat mindestens die Größe $\frac{n}{2}$, und es gibt genau zwei minimale $(1,0)$-geeignete Knotenmengen S_1, S_2 im Graphen G_n für gerade $n \in \mathbb{N}$, die genau die Größe $\frac{n}{2}$ besitzen.*

$$\begin{aligned} S_1 &= \{a\} \cup \{(2,j) \mid 1 \leq j \leq \frac{n}{2} - 1\} \\ S_2 &= \{b\} \cup \{(1,j) \mid 1 \leq j \leq \frac{n}{2} - 1\} \end{aligned}$$

Weiterhin gilt:

$$\begin{aligned} d_{S_1} &= d_{S_2} \\ &= \frac{n}{2} - 1 \end{aligned}$$

Beweis: Die Menge zu beachtender Knotenpaare im Graphen G_n ist für den Umgebungsabstand $s = 1$:

$$M(1) = V_n \times V_n - \{(u,v) \in V_n \times V_n \mid \{u,v\} \in E_n\}$$

Wir zeigen, dass jede $(1,0)$-geeignete Knotenmenge S für jedes $1 \leq j \leq \frac{n}{2} - 1$ mindestens einen Knoten aus jeder der folgenden drei Mengen enthält:

$$\{a,(1,j)\}, \quad \{b,(2,j)\}, \quad \{(1,j),(2,j)\}$$

Angenommen, es sei $a \notin S \wedge (1,j) \notin S$, dann ist das Knotenpaar $(a,(2,j)) \in M(1)$ nicht $(S,1,0)$-geeignet. Analog gilt für $b \notin S \wedge (2,j) \notin S$, dass das Knotenpaar $(b,(1,j)) \in M(1)$ nicht $(S,1,0)$-geeignet ist, und für den Fall $(1,j) \notin S \wedge (2,j) \notin S$ sind die Knotenpaare $((1,j),b),((2,j),a) \in M(1)$ nicht $(S,1,0)$-geeignet.

Daraus ergibt sich, dass die o.g. Knotenmengen S_1 und S_2 die einzigen minimalen $(1,0)$-geeigneten Knotenmengen der Größe $\frac{n}{2}$ sind. Der maximale Grad eines Knoten $u \in V_n - S_i$ in dem $(S_i, 1, 0)$-geeigneten Graphen G^i für $1 \leq i \leq 2$ ist:

$$\begin{aligned} d_{S_1} &= d_{G^1}(b) \\ &= \frac{n}{2} - 1 \\ d_{S_2} &= d_{G^2}(a) \\ &= \frac{n}{2} - 1 \end{aligned}$$

■

Die Graphfamilie G_n ist außerdem ein Beispiel dafür, dass die $(1,0)$ geeigneten Knotenmengen genau ein VC sind (siehe Satz 7). Der Nutzen von geeigneten Graphen wird allerdings nicht erbracht, weil ein quadratischer Speicheraufwand und ein quadratischer Aufwand bei der Entfernungsberechnung erforderlich sind, wenn die geeigneten Knotenmengen und die Umgebungsgröße von linearer Größenordnung sind. Hingegen sind wir von der Annahme ausgegangen, dass die Umgebungsgröße und die geeignete Knotenmenge selbst klein sind.

6.3.2 Approximation der Größe der geeigneten Knotenmenge

Der folgende Greedy Algorithmus zur Berechnung einer approximativen Lösung von HSPS wählt bei jedem Schleifendurchlauf die größte beachtete Menge von Knotenpaaren aus und terminiert, wenn die gesamte Menge zu beachtender Knotenpaare berücksichtigt wurde.

Algorithmus 6 *Greedy-Geeignet*

Eingabe: *Eine Menge von Knotenpaaren $D \subseteq V \times V$ und für jeden Knoten $u \in V$ eine Menge von Knotenpaaren $D(u) \subseteq D$ mit $\bigcup_{u \in V} D(u) = D$.*

Ausgabe: *Eine Knotenmenge $S \subseteq V$*

$S := \emptyset$;
while $D \neq \emptyset$ **do**
  wähle $v \in V - S$ mit $|D(v)| = \max\{|D(u)| \mid u \in V - S\}$;
  $S := S \cup \{v\}$;
  $D := D - D(v)$;
  for $u \in V - S$ **do** $D(u) := D(u) - D(v)$ **endfor** ;
endwhile ;
return S;

Beim Aufruf von *Greedy-Geeignet* werden die zu beachtenden Knotenpaare $M(s)$ und die beachteten Knotenpaarmengen $M(u,s)$ für alle Knoten $u \in V$ übergeben. *Greedy-Geeignet* berechnet zum Graphen aus Abbildung 6.4 in zwei Durchläufen der while-Schleife die folgenden beachteten Knotenpaarmengen und terminiert mit der $(1,0)$-geeigneten Knotenmenge $\{1,4\}$.

It.	D(1) = M(1,1)	$D(2) = M(2,1)$	D(3) = M(3,1)	D(4) = M(4,1)	D(5) = M(5,1)	S
0.	{(1,4), (1,5), (2,3), (3,2)}	$\{(2,3),(2,5)\}$	{(1,4), (1,5), (3,2), (3,5), (4,1)}	{(2,5), (3,5), (4,1), (5,1), (5,2), (5,3)}	{(5,1), (5,2), (5,3)}	$\emptyset$
1.	{(1,4), (1,5), (2,3), (3,2)}	$\{(2,3)\}$	{(1,4), (1,5), (3,2)}	$\emptyset$	$\emptyset$	$\{4\}$
2.	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{1,4\}$

Tabelle 6.1: Die beachteten Knotenpaare zum Graphen aus Abbildung 6.4.

Satz 9 *Der Algorithmus Greedy-Geeignet läuft in $O(n^3)$ Operationen.*

Beweis: Die Menge $R = \{|D(u)| \mid u \in V - S\}$ wird in einem Heap verwaltet, so dass zur Maximumberechnung, zum Löschen des Maximums und zum Ändern eines Wertes $|D(u)|$ höchstens $O(\log n)$ Operationen notwendig sind. Die while-Schleife wird höchstens n mal durchlaufen, und bei jedem Durchlauf werden höchstens n Änderungen von Werten des Heaps vorgenommen. Somit erfolgt die Verwaltung von R in höchstens $O(n^2 \log n)$ Operationen.

Die Mengen von Knotenpaaren $D(u)$ werden für jeden Knoten $u \in V$ als quadratische Bitmatrix verwaltet. Das Entfernen eines Knotenpaares aus allen Knotenpaarmengen, das sind höchstens $|V - S|$, erfolgt dann in $O(|V - S|) \subseteq O(n)$ Operationen. Weil insgesamt höchstens $|D| \leq n^2$ Knotenpaare entfernt werden, ergibt sich für das Entfernen von Knotenpaaren eine Gesamtlaufzeit von $O(n^3)$.

■

Aufgrund der Größe der Eingabedaten im worst-case von $\Sigma_{u \in V}|D(u)| \in \Theta(n^3)$ ist die Aussage von Satz 9 optimal. Eine Verbesserung lässt sich unter der folgenden Annahme

$$\Sigma_{u \in V}|D(u)| \quad \leq \quad O(n^2) \tag{6.5}$$

erzielen.

Satz 10 *Der Algorithmus Greedy-Geeignet kann unter der Annahme 6.5 auf eine Laufzeit $O(n^2 \log n)$ reduziert werden.*

Beweis: Für jedes Knotenpaar $(x, y) \in D$ werden die Knoten $X_{x,y} = \{u \in V - S \mid (x, y) \in D(u)\}$ in einem Heap verwaltet, d.h. es sind maximal n^2 weitere Heaps zu verwalten. Die for-Schleife im Algorithmus *Greedy-Geeignet* wird durch folgenden Ausdruck ersetzt.

for $(x, y) \in D(v)$ **do**
  for $u \in X_{x,y}$ **do**
    $D(u) := D(u) - \{(x, y)\}$
  endfor ;
endfor ;

Es sind zwar für das Löschen eines Knotenpaares (x, y) aus allen Knotenpaarmengen $D(u)$ mit $u \in X_{x,y}$ evtl. $O(n \log n)$ Operationen und für das Löschen aller Knotenpaare aus $D(v)$ im worst-case $O(n^2 \log n)$ Operationen notwendig, allerdings werden nur die Knotenpaarmengen behandelt, die das zu löschende Knotenpaar auch enthalten. Es sind also insgesamt höchstens $O(\Sigma_{u \in V} |D(u)| \log n)$ Operationen für das Löschen von Knotenpaaren notwendig. Die Verwaltung der Größen der einzelnen Knotenpaarmengen (die Menge R) wird nach den Aussagen im Beweis zu dem vorigen Satz in $O(n^2 \log n)$ Operationen vorgenommen, und so ergibt sich unter der Annahme von Gleichung 6.5 die Gesamtlaufzeit $O(n^2 \log n)$.

■

Korollar 3 *Wenn für alle Knoten $u \in V$ die Knotenpaarmengen $D(u)$ von linearer Größenordnung sind, dann kann der Algorithmus Greedy-Geeignet in der Laufzeit $O(n^2 \log n)$ implementiert werden.*

Beweis: Aus der Annahme

$$|D(u)| \quad \in \quad O(n) \qquad \forall u \in V$$

folgt die Gleichung 6.5 und damit Satz 10.

■

Satz 11 *Der Algorithmus Greedy-Geeignet berechnet mit der o.g. Eingabe eine $(s, 0)$-geeignete Knotenmenge S.*

Beweis: Folgt aus der Schleifeninvarianten:

$$\bigcup_{u \in S} D(u) \quad = \quad M(s) - D \tag{6.6}$$

■

Satz 12 *Gegeben sei* $m_{\max} = \max\{|M(u,s)| \mid u \in V\}$. *Der Algorithmus Greedy-Geeignet berechnet eine approximative Lösung von HSPS, die höchstens um den Faktor* $H(m_{\max}) \leq 2(\ln 2)(\log_2 n) + 1$ *vom Optimum abweicht. Dabei sei* $H(d) = \sum_{i=1}^{d} \frac{1}{i} \leq (\ln d) + 1 = (\ln 2)(\log_2 d) + 1$ *die harmonische Reihe (siehe [BS81]).*

Beweis: Das Problem HSPS lässt sich auch als Set Cover formulieren.

Problem 3: ***Set Cover SC***

Eingabe: *Eine endliche Menge* X*, eine Menge* C *von Teilmengen aus* X *und eine Zahl* $l \in \mathbb{N}$.

Frage: *Gibt es eine Menge* $C' \subseteq C$ *mit* $|C'| \leq l$*, die* X *überdeckt* $(\bigcup_{c \in C'} c = X)$*?*

Chvátal, Johnson und Lovasz haben gezeigt, dass ein Greedy Algorithmus, ähnlich dem Algorithmus *Greedy-Geeignet*, eine Approximation für SC produziert, die höchstens um den Faktor $H(\max\{|c| \mid c \in C\})$ vom Optimum abweicht [Chv79] [Joh74] [Lov75]. Dabei korrespondieren die Grundmenge X mit den zu beachtenden Knotenpaaren $M(s)$ und die Kollektion C mit den beachteten Knotenpaarmengen $M(u,s)$ für $u \in V$. Somit ergibt sich eine maximale Abweichung von $H(m_{\max}) \leq (\ln 2)(\log_2 m_{\max})+1$. Wegen $m_{\max} \leq n^2$ folgt die Aussage.

■

Wir zeigen anhand eines Beispiels, dass die Aussage von Satz 12 bis auf konstante Faktoren optimal ist, d.h. die vom Algorithmus *Greedy-Geeignet* berechnete Approximation kann tatsächlich um einen logarithmischen Faktor vom Optimum abweichen. Dazu definieren wir eine Familie von Graphen G_k für $k \in \mathbb{N}_0$ mit $3 \cdot 2^k + k + 4 \in O(2^k)$ Knoten. Die kleinste geeignete Knotenmenge B besteht nur aus drei Knoten, während *Greedy-Geeignet* eine geeignete Knotenmenge A der Größe $k+1$ berechnet.

$$
\begin{aligned}
G_k &= (W \cup A \cup B, E_1 \cup E_2 \cup E_3, \delta) \\
W &= \{(i,j) \in \mathbb{N}_0 \times \mathbb{N}_0 \mid 1 \leq i \leq 3, 1 \leq j \leq 2^k\} \\
A &= \{a_0, a_1, \ldots, a_k\} \\
B &= \{b_1, b_2, b_3\} \\
E_1 &= \{\{(i,j), b_i\} \mid 1 \leq i \leq 3, 1 \leq j \leq 2^k\} \\
E_2 &= \{\{(i,j), a_r\} \mid 1 \leq i \leq 3, 0 \leq r \leq k, 2^{r-1} < j \leq 2^r\} \\
E_3 &= \{\{x,y\} \mid x,y \in A \cup B, x \neq y\} \\
\delta_{|E_1 \cup E_2} &\equiv 1 \\
\delta_{|E_3} &\equiv 0
\end{aligned}
$$

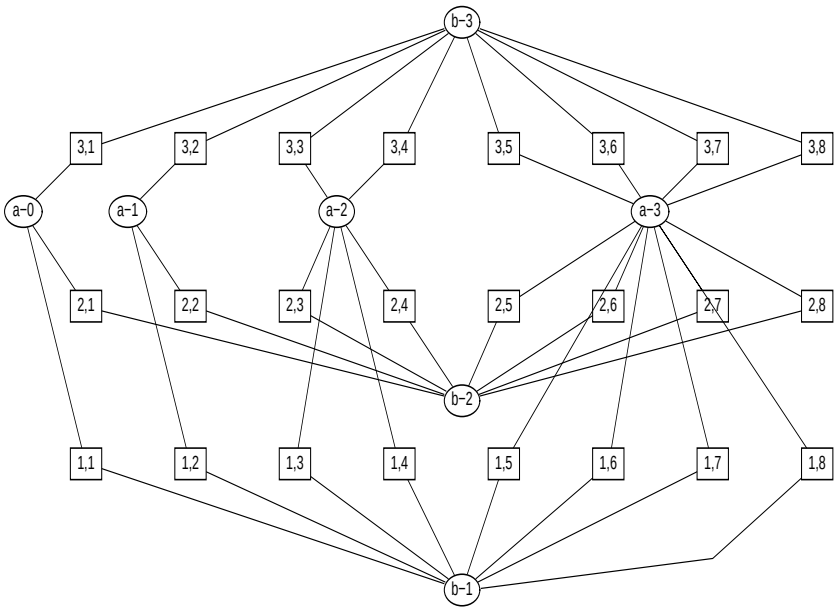

Abbildung 6.7: Graph G_3 ohne die Kanten E_3.

In Abbildung 6.7 ist der Graph G_3 ohne die Kanten E_3 dargestellt. Der Graph G_k ist offensichtlich für jedes $k \in \mathbb{N}_0$ zusammenhängend, besitzt einen Diameter von drei und einen Entfernungsdiameter von zwei ($diam(G_k) = 3, diam^{\delta}(G_k) = 2$). Eine Teilknotenmenge $X \subseteq V(G_k)$ ist genau dann $(1,0)$-geeignet, wenn die Nachbarknoten von X die Knotenmenge des Graphen überdecken ($N(X) = V(G_k)$). Somit sind die Knotenmengen A und B jeweils $(1,0)$-geeignet, wobei B die kleinste $(1,0)$-geeignete Knotenmenge ist. Es sind die Knotenpaare

$$M(1) \quad = \quad W \times W - \{(x,x) \mid x \in W\}$$

zu beachten und jeder Knoten $u \in A \cup B$ beachtet die Knoten

$$M(u,1) \quad = \quad N(u) \times W - \{(x,x) \in N(u) \times N(u)\}$$

Folglich wählt der Algorithmus *Greedy-Geeignet* beim Aufruf mit den zu **beachtenden** Knotenpaaren $M(1)$ den Knoten $a_k \in A$ mit größtem Grad $d(a_k) = 3 \cdot 2^{k-1} + k + 3$ in G_k für $k \geq 1$ und den Knoten $a_0 \in A$ mit Grad $d(a_0) = 6$ in G_0 aus. In der nächsten Iteration sind nur noch die Knotenpaare

$$\begin{aligned} M'(1) &= M(1) - M(u,1) \\ &= \{\{(i,j),y\} \mid 1 \leq i \leq 3, 1 \leq j \leq 2^{k-1}, y \in W, y \neq (i,j)\} \end{aligned}$$

zu beachten, und von jedem Knoten $u \in A \cup B$ werden die Knoten

$$M'(u,1) \quad = \quad M(u,1) \cap M'(1)$$

beachtet.

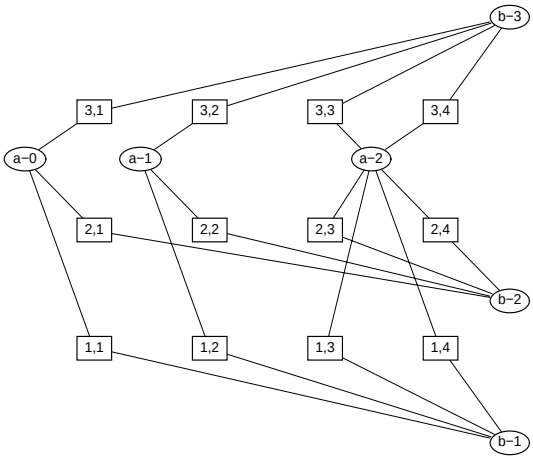

Abbildung 6.8: Graph $G_3 - N(a_3)$ ohne die Kanten E_3.

Somit wird der Knoten $a_{k-1} \in A$ mit größtem Grad $d(a_{k-1}) = 3 \cdot 2^{k-2} + k + 2$ in $G_k - N(a_k)$ ausgewählt (siehe Graph $G_3 - N(a_3)$ in Abbildung 6.8). *Greedy-Geeignet* wählt also die Knoten $a_k, a_{k-1}, \ldots, a_1, a_0$ in dieser Reihenfolge aus (lediglich die Auswahl der letzten beiden Knoten kann auch in umgekehrter Reihenfolge vorgenommen werden). A ist wegen $\frac{k+1}{3} \leq \frac{log(n)}{3}$ eine $\frac{log(n)}{3}$-Approximation für die Größe der geeigneten Knotenmenge.
1995 hat Halldórsson durch lokale Verbesserungen des Greedy Algorithmus eine Verbesserung der oberen Schranke $H(n)$ um einen konstanten Faktor ($\frac{11}{42}$) angegeben [Hal95], und Slavík hat gezeigt, dass sogar die obere Schranke $\ln n - f(n)$ für eine Funktion $f(n) \in \Theta(\ln \ln n)$ gilt [Sla95]. Lund und Yannakakis haben unter der Voraussetzung, dass $DTIME(n^{ln^{O(1)}n}) \neq NTIME(n^{ln^{O(1)}n})$ gilt, gezeigt, dass es keinen effizienten Algorithmus gibt, der für SC eine $c \log n$-Approximation berechnet ($c < \frac{1}{4}$) [LY94]. Vermutlich gibt es also keinen effizienten Algorithmus, der eine wesentlich günstigere als eine logarithmische Approximation berechnet.

6.3.3 Approximation des Umgebungsabstands

Die beiden folgenden Algorithmen sind Modifikationen des Farthest Point Algorithmus, den auch Feder und Greene für Approximationen der Clustergröße verwendet haben [FG88].

Algorithmus 7 *Farthest Vertex I*

Eingabe: *Ein Graph* $G = (V, E, \delta)$ *und ein* $1 \leq k \leq n$.

Ausgabe: *Eine Knotenmenge* $S \subseteq V$ *und ein Umgebungsabstand* $1 \leq s' \leq n$.

$S := \emptyset$;
while $|S| < k$ **and** $SP^{G[V-S]} \neq \emptyset$ **do**
Sei $w \in SP^{G[V-S]}$ ein kürzester Weg mit
$|w| = \max\{d_\delta^{G[V-S]}(u,v) \mid u,v \in V-S\}$
und $u \in V(w)$ ein 'mittlerer' Knoten von w.
$s' := |w|$;
$S := S \cup \{u\}$;
endwhile ;
return S und s';

Die Laufzeit des Algorithmus *Farthest Vertex I* wird durch die Entfernungsberechnung für alle Knotenpaare im Inneren der while-Schleife bestimmt. Dabei wird für alle Knotenpaare die minimale Kantenanzahl aller kürzesten Wege in dem induzierten Graphen $G[V-S]$ berechnet. Dies erfolgt mit dem *Dijkstra* Algorithmus in $O(n \cdot m + n^2 \log n)$ Operationen (siehe Kapitel 3). Insgesamt ergibt sich somit eine Laufzeit von $O(k(n \cdot m + n^2 \log n)) \subseteq O(kn^3)$ Operationen.

Satz 13 *Der Algorithmus Farthest Vertex I berechnet eine $(s', 0)$-geeignete Knotenmenge $S \subseteq V(G)$ für den Graphen G mit $|S| = k$.*

Beweis: Angenommen, es sei $(u,v) \in M(s')$. Nach dem Algorithmus *Farthest Vertex I* gilt:

$$d_\delta(u,v) \quad \leq \quad s'$$

Offensichtlich gilt außerdem

$$d(u,v) \quad \leq \quad d_\delta(u,v)$$

Dies führt zu einem Widerspruch. Also ist $M(s') = \emptyset$ und somit gilt die Aussage.

■

Es ist für die Korrektheit von Satz 13 nicht entscheidend, welcher Knoten auf dem Weg w im Rumpf der while-Schleife ausgewählt wird. Wählt man jedoch den Anfangs- oder Endknoten von w, ist die maximale Länge (in der Anzahl von Kanten) in dem Restgraphen höchstens um eins gesunken. Wählt man den mittlersten Knoten, könnte sich diese Länge bereits halbiert haben. Man kann hier auch eine umfassendere Greedy-Strategie einsetzen, die den Knoten $u \in V - S$ wählt, der die maximale Länge von kürzesten Wegen im Restgraphen minimiert. Dazu ist der Rumpf der while-Schleife zu ersetzen.

Algorithmus 8 *Farthest Vertex II*

Eingabe: *Ein Graph $G = (V, E, \delta)$ und ein $1 \leq k \leq n$.*

Ausgabe: *Eine Knotenmenge $S \subseteq V$ und ein Umgebungsabstand $1 \leq s' \leq n$.*

$S := \emptyset$;
while $|S| < k$ **and** $SP^{G[V-S]} \neq \emptyset$ **do**
Es sei für alle Knoten $v \in V - S$:
$d_{max}(v) := \max\{d_\delta^{G[V-S-\{v\}]}(x,y) \mid x,y \in V - S - \{v\}\}$.
Wähle $u \in V - S$ mit
$d_{max}(u) = \max\{d_{max}(v) \mid v \in V - S\}$
$s' := d_{max}(u)$;
$S := S \cup \{u\}$;
endwhile ;
return S und s';

Die Laufzeit vergrößert sich für den Algorithmus *Farthest Vertex II* um den Faktor n, weil die Entfernungsberechnung APSD für jeden Graphen $G[V - S - \{v\}]$ für alle Knoten $v \in V - S$ durchgeführt werden muss. Somit ergibt sich eine Gesamtlaufzeit von $O(kn^4)$ Operationen im worst-case. Die Approximationsgüte hinsichtlich der Größe von S kann jedoch auch mit dem Algorithmus *Farthest Vertex II* nicht garantiert werden, d.h. es kann durchaus eine wesentlich kleinere $(s', 0)$-geeignete Knotenmenge geben. Die folgende Variante gleicht dies aus, allerdings unter Zulassung eines größeren zu berücksichtigenden Fehlers.

Algorithmus 9 *Farthest Vertex III*

Eingabe: *Ein Graph* $G = (V, E, \delta)$ *und ein* $1 \leq k \leq n$.

Ausgabe: *Eine Knotenmenge* $S \subseteq V$ *und ein Umgebungsabstand* $1 \leq s' \leq n$.

$S := \emptyset$;
while $|S| < k$ **and** $AR(S) \neq \emptyset$ **do**
Wähle einen Knoten $u \in V - S$ mit
$d(u, S) = \max\{d(v, S) \mid v \in V - S\}$
$s' := d(u, S)$;
$S := S \cup \{u\}$;
endwhile ;
return S und s';

Die Laufzeit des Algorithmus *Farthest Vertex III* hat sich im Gegensatz zu den beiden anderen Algorithmen dramatisch verringert. Die Entfernungen $d(u, S)$ für alle Knoten $u \in V - S$ können mit dem *Dijkstra* Algorithmus in $O(m + n \log n)$ Operationen berechnet werden, wobei als Startknoten die Knotenmenge S verwendet wird. Da die while-Schleife höchstens k mal durchlaufen wird, ist die Laufzeit von *Farthest Vertex III* höchstens $O(k(m + n \log n))$.

Satz 14 *Gegeben sei ein Graph* $G = (V, E, \delta)$ *mit maximalem Kantengewicht* $c = \max\{\delta(e) \mid e \in E\}$. *Der Algorithmus Farthest Vertex III berechnet eine* $(s', 4s'c)$*-geeignete Knotenmenge* S *mit höchstens* k *Knoten.*
Farthest Vertex III ist fast eine 2-Approximation für den Umgebungsparameter, d.h. es gibt für kein $\xi \in \mathbb{R}^+$ *eine* $(\lceil \frac{s'}{2} \rceil - 1, \xi)$*-geeignete Knotenmenge mit weniger als* k *Knoten.*

Beweis: Sei $w \in SP^G$ und $|w| > s'$. In der s' Nachbarschaft von $s(w)$ und $e(w)$ liegen Knoten aus S, d.h. es gilt:

$$\begin{aligned} N^{s'}(s(w)) \cap S &\neq \emptyset \quad \text{und} \\ N^{s'}(e(w)) \cap S &\neq \emptyset\,. \end{aligned}$$

Somit ist die berechnete Entfernung, wenn der Weg über die Knoten aus S führt, nach Lemma 5 höchstens um $4s'$ Kantengewichte zu groß.

Die zweite Aussage folgt, weil alle Knotenpaare aus S mindestens s' Kanten voneinander entfernt sind, d.h. für alle Knoten $u, v \in S$ gilt:

$$d(u,v) \quad \geq \quad s'$$

Somit gibt es keinen Knoten $u \in V$, der im Abstand $\lceil \frac{s'}{2} \rceil - 1$ von zwei Knoten aus S liegt. Jede $(\lceil \frac{s'}{2} \rceil - 1, \xi)$-geeignete Knotenmenge besteht also aus mindestens k Knoten.

■

6.4 Verallgemeinerungen von HSPS

Eine Verallgemeinerung von SC ist das folgende Problem.

Problem 4: ***Partial-Set-Cover PSC***

Eingabe: *Eine endliche Menge X, eine Kollektion C von Teilmengen aus X, ein Parameter $l \in \mathbb{N}$ und ein Anteil $p \in [0,1]$.*

Frage: *Gibt es eine Kollektion $C' \subseteq C$ mit $|C'| \leq l$, die X mindestens mit dem Anteil p überdeckt $(|\bigcup_{c \in C'} c| \geq p|X|)$?*

Offensichtlich ist SC ein Spezialfall von PSC mit dem Parameter $p = 1$. Slavík zeigt 1995, dass ein modifizierter Greedy-Algorithmus PSC bis zu einer unteren Schranke $\log n - \Theta(\log \log n)$ löst [Sla95].
HSPS lässt sich analog zu SC verallgemeinern.

Problem 5: ***Partial-Hitting-Shortest-Paths-Set PHSPS***

Eingabe: *Graph $G = (V, E, \delta)$, $s \in \mathbb{N}_0$, $l \in \mathbb{N}$ und $p \in [0,1]$.*

Frage: *Existiert eine Knotenmenge $S \subseteq V$ mit $|S| \leq l$, so dass mindestens ein Anteil von p Knotenpaaren, die einen größeren Abstand als s besitzen, $(s,0)$-geeignet sind $(p|M(s)| \leq |\bigcup_{u \in S} M(u,s)|)$?*

PHSPS lässt sich durch eine kleine Modifikation des Algorithmus *Greedy-Geeignet* approximieren.

Algorithmus 10 *Greedy-Teilgeeignet*

Eingabe: *Eine Menge von Knotenpaaren $D \subseteq V \times V$, für jeden Knoten $u \in V$ eine Menge von Knotenpaaren $D(u) \subseteq D$ und einen Parameter $p \in [0,1]$.*

Ausgabe: *Eine Knotenmenge $S \subseteq V \times V$*

$S := \emptyset$;
$x := |D|$;
while $|D| > (1-p)x$ **do**
  wähle $v \in V - S$ mit $|D(v)| = \max\{|D(u)| \mid u \in V - S\}$
  $S := S \cup \{v\}$;
  $D := D - D(v)$;
  for $u \in V - S$ **do** $D(u) := D(u) - D(v)$ **endfor** ;
endwhile ;
return S;

Korollar 4 *Wenn der Algorithmus Greedy-Teilgeeignet mit der zu beachtenden Knotenmenge $M(s)$, den beachteten Knotenpaarmengen $M(u,s)$ für alle Knoten $u \in V$ und einem Parameter p aufgerufen wird, berechnet er eine Knotenmenge S mit:*

$$p|M(s)| \quad \leq |\bigcup_{u \in S} M(u,s)|$$

Beweis: Die Schleifeninvariante (Gleichung 6.6) gilt hier ebenso, weil der Rumpf der while-Schleife mit dem Rumpf der while-Schleife des Algorithmus *Greedy-Geeignet* identisch ist. Mit obiger Abbruchbedingung gilt nach Beendigung der while-Schleife $|D| \leq (1-p)|M(s)|$, und somit folgt die Aussage.

■

6.5 Lokalitätseigenschaft

Theorem 3 *Gegeben seien ein Graph $G = (V, E, \delta)$ und ein Umgebungsabstand $s \in \mathbb{N}_0$. Eine Knotenmenge $S \subseteq V$ ist genau dann $(s,0)$-geeignet, wenn alle Knotenpaare $u, v \in V$ mit Abstand $d(u,v) = s+1$ $(S,s,0)$-geeignet sind.*

Beweis: Ist S $(s,0)$-geeignet, so sind auch alle Knotenpaare $u, v \in V$ mit Abstand $d(u,v) = s+1$: $(S,s,0)$-geeignet.

Für die andere Richtung zeigen wir, dass für eine nicht $(s,0)$-geeigente Knotenmenge $S \subseteq V$ mindestens ein nicht $(S,s,0)$-geeignetes Knotenpaar $u, v \in V$ mit Abstand $d(u,v) = s+1$ existiert. Sei $(u,v) \in V \times V$ ein nicht $(S,s,0)$-geeignetes Knotenpaar mit minimalem Abstand, der größer als s ist, d.h. es gilt:

$$d(u,v) \quad = \quad \min\{d(x,y) \mid d(x,y) > s \wedge (x,y) \text{ ist nicht } (S,s,0) - \text{geeignet}\} \tag{6.7}$$

Im Widerspruch zur Aussage sei $d(u,v) > s+1$ und

$$w = [u = x_1, x_2, \ldots, x_r = v] \quad \in \quad SP(u,v) \tag{6.8}$$

sei ein kürzester Weg mit

$$\begin{aligned} d(u,v) &= r-1 \\ &> s+1 \end{aligned} \tag{6.9}$$

Das Knotenpaar (x_2, x_r) besitzt nach Gleichung 6.9 den Abstand $d(x_2,x_r) = r-2 > s$ und ist somit nach Gleichung 6.7 $(S,s,0)$-geeignet, d.h. es gibt einen Knoten $x' \in N^s(x_2) \cap S$, so dass gilt:

$$\begin{aligned} \delta(x_2, N^s(x_2) \cap S, N^s(x_r) \cap S, x_r) &\leq \delta(x_2, x', N^s(x_r) \cap S, x_r) \\ &= \delta(x_2, x_r) \end{aligned} \tag{6.10}$$

Somit folgt: $d(x_1, x') \leq s+1$.

Wir unterscheiden nun zwei Fälle, die beide zum Widerspruch führen (siehe Abbildung 6.9).

Fall 1: $x' \in N^s(x_1)$**.** Dann folgt:

$$\begin{aligned} \delta(x_1, N^s(x_1) \cap S, N^s(x_r) \cap S, x_r) &\leq \delta(x_1, x_2, x', N^s(x_r) \cap S, x_r) \\ &\underbrace{=}_{Gl.\ 6.10} \delta(x_1, x_2, x_r) \\ &\underbrace{=}_{Gl.\ 6.8} \delta(x_1, x_r) \end{aligned}$$

Somit wäre das Knotenpaar (x_1, x_r) im Widerspruch zur Annahme $(S,s,0)$-geeignet.

Fall 2: $x' \notin N^s(x_1)$**.** Dann ist der Abstand $d(x_1,x') = s+1$ und somit das Knotenpaar (x_1,x') nach Annahme $(S,s,0)$-geeignet. Wir wählen folgende abkürzende Schreibweise:

$$c \quad := \quad \delta(x_1, N^s(x_1) \cap S, N^s(x') \cap S, x', N^s(x_r) \cap S, x_r)$$

Dann gilt:

$$\begin{aligned} \delta(x_1, N^s(x_1) \cap S, N^s(x_r) \cap S, x_r) &\leq c \\ &= \delta(x_1, x', N^s(x_r) \cap S, x_r) \\ &\leq \delta(x_1, x_2, x', N^s(x_r) \cap S, x_r) \\ &\underbrace{=}_{Gl.\ 6.10} \delta(x_1, x_2, x_r) \\ &\underbrace{=}_{Gl.\ 6.8} \delta(x_1, x_r) \end{aligned}$$

Folglich wäre auch in diesem Fall das Knotenpaar (x_1, x_r) im Widerspruch zur Annahme $(S,s,0)$-geeignet.

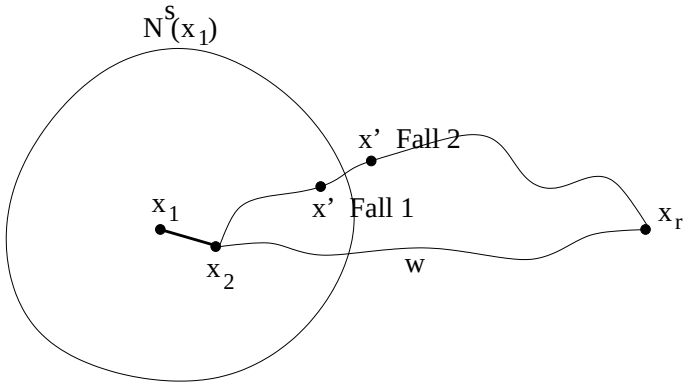

Abbildung 6.9: Veranschaulichung zum Beweis von Theorem 3.

■

Zur Veranschaulichung von Satz 3 betrachten wir wiederum das Gitter in Abbildung 6.2. Die markierte Knotenmenge $S(3, 10)$ ist nicht $(1, 0)$-geeignet. Folglich gibt es Knoten mit Abstand zwei, die nicht $(S(3, 10), 1, 0)$-geeignet sind. Dies sind beispielsweise die Knoten $(0, 0)$ und $(0, 2)$.
Die Lokalitätseigenschaft lässt sich nicht auf (s, ξ)-geeignete Knotenmengen für $\xi > 0$ verallgemeinern. In Abbildung 6.10 ist ein Graph mit konstanter Kantengewichtsfunktion dargestellt. Die markierten Knoten bilden die Knotenmenge S. Alle Knotenpaare mit größerem Abstand als eins sind $(S, 1, 1)$-geeignet, bis auf das Knotenpaar $1, 11$. Daher bilden die markierten Knoten keine $(1, 1)$-geeignete Knotenmenge.

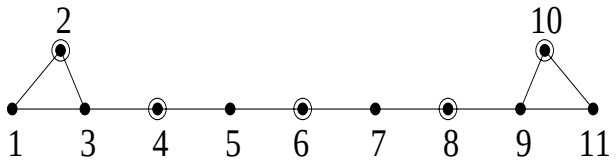

Abbildung 6.10: $\{2, 4, 6, 8, 10\}$ ist nicht $(1, 0)$-geeignet.

Im nächsten Abschnitt nutzen wir die Lokalitätseigenschaft aus, um einen effizienteren Eignungstest einzuführen. Außerdem geben wir einen effizienteren Algorithmus zur Berechnung von geeigneten Knotenmengen an.

6.5.1 Modifikationen der Algorithmen

Wir definieren die Menge zu beachtender und beachteter Knotenpaare aufgrund der Lokalitätseigenschaft erneut (siehe Definition 6).

$$
\begin{aligned}
M^l(s) &= \{(u,v) \in V \times V \mid d(u,v) = s+1\} \\
M^l(u,s) &= \{(x,y) \in M^l(s) \mid u \in \bigcup_{w \in SP(x,y)} V(w) \cap N^s(x)\}
\end{aligned}
$$

Korollar 5 *Wenn der Algorithmus Eignungstest mit der Knotenmenge $D = M^l(s)$, den Graphen $G = (V, E, \delta)$, $G' = (V, E', \delta')$ und einer Knotenmenge $S \subseteq V$ aufgerufen wird, so wird für die Knotenmenge S und den Graphen entschieden, ob sie $(s,0)$-geeignet bzw. $(S,s,0)$-geeignet sind.*
Wenn der Algorithmus Greedy-Geeignet mit den Knotenmengen $D = M^l(s)$ und $D(u) = M^l(u,s)$ für alle Knoten $u \in V$ aufgerufen wird, dann wird eine $(s,0)$-geeignete Knotenmenge berechnet.

Beweis: Folgt aus Theorem 3 und den Beweisen der Algorithmen.

■

Wenn man für eine Graphfamilie annimmt, dass die Mengen von Knotenpaaren $M^l(u,s)$ für alle Knoten $u \in V$ durch eine Konstante beschränkt sind (für kleine Umgebungen ist dies naheliegend), so ergibt sich für den Algorithmus *Greedy-Geeignet* eine fast lineare Laufzeit und folgende Verbesserung der Approximationsgüte.

Korollar 6 *Es gelte für alle Knoten $u \in V(G)$: $M^l(u,s) \leq k$ für ein $k \in \mathbb{N}$. Dann berechnet Greedy-geeignet eine $\log k + 1$-Approximation für die Größe der geeigneten Knotenmenge. Sofern dies für eine Familie von Graphen gilt, wird folglich eine konstante Approximation berechnet.*

Beweis: Folgt aus Satz 12.

■

Man beachte, dass sich die Lokalitätseigenschaft nicht auf teilgeeignete Knotenmengen übertragen lässt. Der Algorithmus *Greedy-Teilgeeignet* löst für den Aufruf mit den Knotenpaarmengen $D = M^l(s)$ und $D(u) = M^l(u,s)$ nicht das Problem PHSPS, weil sich der Anteilparameter p auf die gesamte Knotenpaarmenge $M(s)$ bezieht.

6.6 Levelgraphen

In diesem Abschnitt bilden wir einen Straßengraphen in einen k-Levelgraphen ab, wobei das Level Null eines k-Levelgraphen genau dem Straßengraphen entspricht und die höheren Level eine Ausdünnung niedrigerer Level darstellen. Die Knotenmengen auf höheren Leveln sind immer Teilmengen von Knoten aus

niedrigeren Leveln. Allerdings kann es auf höheren Leveln durchaus Kanten geben, die auf unteren Leveln nicht vorhanden sind. Für die Kantengewichtsfunktionen der einzelnen Level gilt jedoch, dass die Entfernungen auf unteren Leveln immer eine untere Schranke für die Entfernungen auf höheren Leveln sind. Die Einbeziehung höherer Level soll aufgrund kleinerer Graphen zu einer effizienteren Entfernungsberechnung führen. Im Weiteren besteht dann die Aufgabe darin, zu einem Straßengraphen einen 'minimalen' k-Levelgraphen zu konstruieren, so dass mit einem einfachen Levelalgorithmus der (fast) kürzeste Weg berechnet werden kann.
Weil Kanten in verschiedenen Leveln existieren können, verwenden wir Multigraphen.

Definition 7 *Ein Graph $G = (V, E, \delta)$ mit Knotenmenge V, Kantenmenge $E = \{\{u, v, i\} \mid u, v \in V, i \in \mathbb{N}_0\}$ und Kantengewichtsfunktion $\delta : E \to \mathbb{R}^+$ heißt* **Multigraph**.

Definition 8 *$G = (V, E, \delta, l)$ heißt* **k-Levelgraph**, *wenn (V, E, δ) ein Multigraph ist und $l : E \to \{0, 1, \ldots, k-1\}$ eine Levelfunktion mit folgender Eigenschaft für alle Kanten $\{u, v, i\}, \{u, v, j\} \in E$:*

$$l(\{u, v, i\}) = i \tag{6.11}$$
$$\delta(\{u, v, i\}) = \delta(\{u, v, j\}) \tag{6.12}$$

Die Kanten, Knoten und den Graphen von Level $i \in \{0, 1, \ldots, k-1\}$ bezeichnen wir mit

$$\begin{aligned} E_i &= \{e \in E \mid l(e) = i\} \\ V_i &= \{u \in V \mid \exists e \in E_i : u \in e\} \\ G_i &= (V_i, E_i, \delta_i \equiv \delta_{|E_i}) \end{aligned}$$

Die Kantenfunktion des Graphen G_i sei auf Knotenpaare aus V_i fortgesetzt, analog zur Fortsetzung der Kantengewichtsfunktion bei einfachen gewichteten Graphen (siehe Gleichung 2.9). Ein k-Levelgraph heißt **knotenmonoton**, *wenn für $0 \leq i < k-1$ gilt:*

$$\begin{aligned} V_{i+1} &\subseteq V_i \\ \delta_i(u, v) &= \delta_{i+1}(u, v) \qquad \forall u, v \in V_{i+1} \end{aligned} \tag{6.13}$$

Ein knotenmonotoner k-Levelgraph heißt **kantenmonoton**, *wenn für $0 \leq i < k-1$ gilt:*

$$E_{i+1} \subseteq E_i$$

Die Graphen der einzelnen Level eines k-Levelgraphen sind einfache Graphen, d.h. aufgrund von Gleichung 6.11 gibt es keine mehrfachen Kanten. Sofern das Level bekannt ist, lassen wir daher die dritte Komponente der Kanten weg. Die Kantengewichte jeder Multikante sind in jedem Level aufgrund von Gleichung 6.12 identisch.

Die Vereinigung einzelner Level eines k-Levelgraphen definieren wir analog zur Vereinigung von einfachen Graphen. Sei $A \subseteq \{0, 1, \ldots, k-1\}$ eine beliebige Teilmenge von Leveln, dann bezeichnet $G_A = (V_A, E_A, \delta_A)$ mit:

$$G_A \quad = \quad \bigcup_{p \in A} G_p$$

den zugehörigen einfachen Graphen (siehe Gleichung 2.16). Die Fortsetzung der Kantengewichtsfunktionen auf Knotenpaare verläuft wiederum analog zur Fortsetzung bei einfachen gewichteten Graphen (siehe Gleichung 2.9). Wir schreiben einen k-Levelgraphen auch als k-Tupel von einfachen Graphen $(G_0, \ldots, G_{k-1})$. Es gilt offensichtlich in jedem k-Levelgraphen stets $\delta(u, v) = \delta_{\{0,1,\ldots,k-1\}}(u, v)$. Man beachte, dass zwar ebenso für jede Levelmenge $A \subseteq \{0, 1, \ldots, k-1\}$ stets folgende Ungleichung gilt:

$$\delta_A(u, v) \quad \leq \quad \min\{\delta_i(u, v) \mid i \in A\}$$

Jedoch muss die Gleichheit i.A. nicht gelten. Ein kürzester Weg zwischen zwei Knoten kann durchaus abwechselnd in verschiedenen Leveln verlaufen. In knotenmonotonen Levelgraphen gilt jedoch die Gleichheit, denn aus Gleichung 6.13 folgt:

$$\delta_A(u, v) \quad = \quad \delta_{\min\{p \in A\}}(u, v)$$

Definition 9 *Ein knotenmonotoner k-Levelgraph $G = (V, E, \delta, l)$ heißt* **fast geeignet** *für die Parameter $(s, d, \xi) \in \mathbb{N}_0^{k-1} \times \mathbb{N}_0^{k-1} \times (\mathbb{R}^+)^{k-1}$ (Umgebungs-, Entfernungs- und Fehlervektor) oder auch (s, d, ξ)-***geeignet**, *wenn für alle $i \in \{0, 1, \ldots, k-2\}$ und alle Knotenpaare $u, v \in V(G)$ mit $d_0(u, v) > d_i$ zwei Knoten $x \in N_0^{s_i}(u) \cap V_{i+1}$ und $y \in N_0^{s_i}(v) \cap V_{i+1}$ existieren mit:*

$$\delta(u, x) + \delta_{i+1}(x, y) + \delta(y, v) \quad \leq \quad \delta(u, v) + \xi_i \tag{6.14}$$

G heißt **geeignet**, *wenn er fast geeignet für einen Fehler $\xi = 0$ ist.*

Hierbei ist $d_0 : V_0 \times V_0 \to \mathbb{N}_0$ eine Abstandsfunktion und $N_0 : V_0 \to 2^{V_0}$ eine Nachbarfunktion im Level Null. Die Umgebungs-, Entfernungs- und Fehlervektoren sollen streng monoton steigend sein, d.h. für $0 \leq i < k-1$ gilt stets: $s_i \leq s_{i+1}$, $d_i \leq d_{i+1}$ und $\xi_i \leq \xi_{i+1}$. Je weiter Knoten voneinander entfernt sind, umso größere Umgebungen müssen betrachtet werden, und ein umso größerer Fehler bei der Entfernungsberechnung ist zugelassen. Die Graphen in höheren Leveln enthalten weniger Knoten und erlauben daher eine effizientere Wegesuche.
Wir veranschaulichen die Definition an einem 6×6-Gitter $G = (V, E, \delta \equiv 1)$ (siehe Abbildung 6.11). Der Graph G' besteht aus den dick gezeichneten Kanten und G'' aus den gestrichelten Kanten in Abbildung 6.12. Der 2-Levelgraph (G, G') ist $(2, 0, 2)$-geeignet, während der 2-Levelgraph (G, G'') aufgrund von $\delta(u, v) = 6$ und $\delta(u, x) + \delta''(x, y) + \delta(y, v) = 14$ nicht einmal $(2, 0, 7)$-geeignet ist.
Folgendes Lemma beschreibt einen Zusammenhang zwischen geeigneten Graphen und geeigneten Levelgraphen.

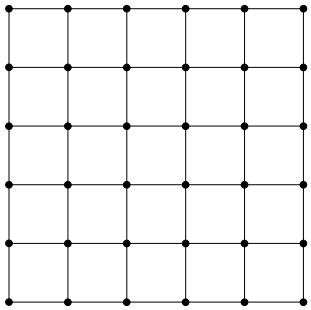

Abbildung 6.11: Ein 6×6-Gitter.

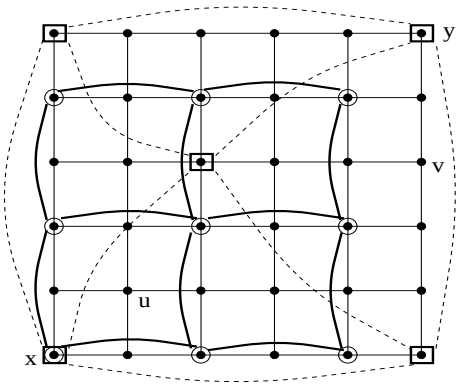

Abbildung 6.12: Zwei 2-Levelgraphen.

Lemma 6 *Gegeben seien ein Graph $G = (V, E, \delta)$, die Parameter $s \in \mathbb{N}_0, \xi \in \mathbb{R}^+$, eine Knotenmenge $S \subseteq V$ und ein vollständiger Graph $G' = (S, E', \delta')$ mit $\delta'(\{u, v\}) = \delta(u, v)$ für alle Knoten $u, v \in S$. Dann gilt:*

$$S \text{ ist } (s,\xi)\text{-geeignet} \iff (G, G') \text{ ist } (s, s, \xi)\text{-geeignet}$$

Beweis: Folgt aus den Definitionen 9 und 4.

■

Es gilt ebenso wie für geeignete Graphen eine Lokalitätseigenschaft.

Korollar 7 *Ein 2-Levelgraph (G, G') ist genau dann $(s, d, 0)$-geeignet, wenn für alle Knotenpaare $u, v \in V(G)$ mit Abstand $d(u, v) = d + 1$ gilt, dass sie $(V(G'), s, 0)$-geeignet sind.*

Beweis: Folgt analog zu Theorem 3 (auf Seite 61).

■

6.6.1 Entfernungsberechnung in Levelgraphen

Algorithmus 11 *Geeignete-Levelsuche*

Eingabe: *k-Levelgraph $G = (V, E, \delta, l)$ mit Umgebungs-, Entfernungs- und Fehlervektor $(s, d, \xi) \in \mathbb{N}_0^{k-1} \times \mathbb{N}_0^{k-1} \times (\mathbb{R}^+)^{k-1}$ und Knoten $u, v \in V$.*

Ausgabe: $\delta'(u, v)$

$i := 0;$
while $i < k - 1$ **and** $d_0(u, v) > d_i$ **do** $i := i + 1;$ **endwhile** ;
$i := i - 1;$
return $\min\{\delta(u, x) + \delta_{i+1}(x, y) + \delta(y, v) \mid x \in N_0^{s_i}(u) \cap V_{i+1}, y \in N_0^{s_i}(v) \cap V_{i+1}\};$

Der Algorithmus *Geeignete-Levelsuche* berechnet offensichtlich zu einem geeigneten Levelgraphen die kürzeste Entfernung ($\delta(u, v) = \delta'(u, v)$). Problematisch ist hierbei die Berechnung von $d_0(u, v)$, die die Berechnung von $\delta(u, v)$ involviert und nicht im voraus stattfinden kann. Anstatt $d_0(u, v)$ kann man eine Schätzfunktion $h : V \times V \to \mathbb{R}^+$ verwenden, die online ausgewertet wird. Die Schätzfunktion sollte möglichst große Werte liefern und es sollte gelten:

$$h(u, v) \leq d_0(u, v) \qquad \forall u, v \in V$$

Für Verkehrsgraphen benutzt man beispielsweise eine Schätzfunktion, die die Koordinaten der Knoten berücksichtigt.
Um die Effizienz des Verfahrens zu gewährleisten, werden die benötigten Längenabstände zu den Nachbarn im Level Null in einer Preprocessing-Phase berechnet

und in Tabellen bzw. Listen abgelegt. Dabei ist es entscheidend, dass in jedem Level nur wenig Knoten vorhanden sind, damit dieser Speicheraufwand gering bleibt.
Wir geben ein Beispiel für einen k-Levelgraphen an, für den wir keine Schätzfunktion verwenden, sondern den Algorithmus *Geeignete-Levelsuche* adaptieren. Der k-Levelgraph besteht aus den folgenden Hilfsgraphen:

$$
\begin{aligned}
H_j &= (V'_j, E'_j, \delta'_j \equiv 1) \qquad \text{für } 0 \leq j \leq k-1 \\
V'_0 &= \{0,1,\ldots,4^k\} \times \{0,1,\ldots,4^k\} \\
E'_0 &= \{\{(x,y),(x',y')\} \subset V'_0 \mid (|x-x'|=1 \wedge y=y') \vee (x=x' \wedge |y-y'|=1)\} \\
V'_j &= \{(x,y) \in V'_0 \mid 0 = x \bmod 4^j \wedge 0 = y \bmod 4^j\} \qquad \text{für } 0 < j \leq k-1 \\
E'_j &= \{\{(x,y),(x',y')\} \subset V'_j \mid (|x-x'|=4^j \wedge y=y') \vee (x=x' \wedge |y-y'|=4^j)\} \\
&\qquad \text{für } 0 < j \leq k-1
\end{aligned}
$$

Für $k = 2$ sind die Hilfsgraphen H_0, H_1 in Abbildung 6.13 dargestellt.

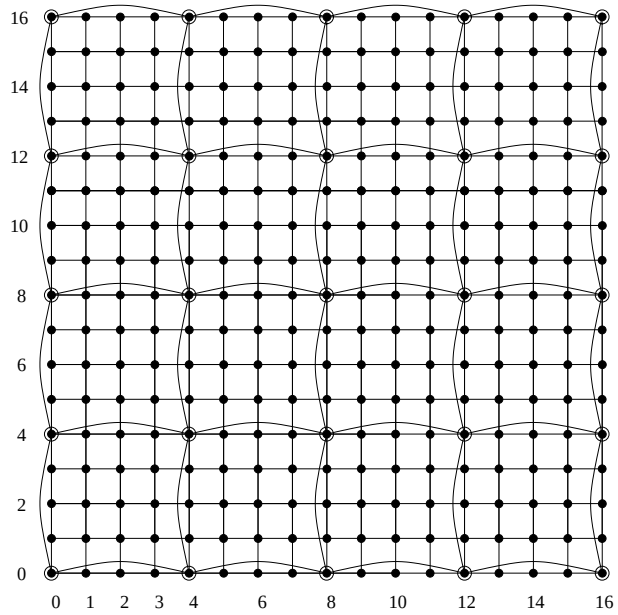

Abbildung 6.13: Hilfsgraphen H_0 und H_1.

Folgender kantenmonotoner k-Levelgraph $(G_0, G_1, \ldots, G_{k-1})$ mit

$$G_i = \bigcup_{j=i}^{k-1} H_j \tag{6.15}$$

für $0 \leq i \leq k-1$ heißt **k-Levelgittergraph**. In einem k-Levelgittergraphen gilt

für alle Knoten $u, v \in V_i$ eines Levels $0 < i \leq k-1$:

$$\delta_i(u,v) \quad = \quad \delta_0(u,v) \tag{6.16}$$

Es genügt also, für Knoten eines Levels i die Knoten des Levels i bei der Entfernungsberechnung zu betrachten.
Für die Formulierung einer weiteren wichtigen Eigenschaft von k-Levelgittergraphen definieren wir für jeden Knoten $u \in V_0$ und jedes Level $0 \leq i \leq k-1$ eine Teilmenge der Knoten des Levels i als Aufstiegsknoten. Die Knoten $A(u,i) \subseteq V_i$ heißen **Aufstiegsknoten**, wenn für alle Knoten $u' \in A(u,i)$ ein kürzester Weg $w \in SP(u,u')$ (bezogen auf den gesamten Graphen) existiert, der bis auf u' keinen Knoten des Levels i enthält:

$$V(w) - \{u'\} \cap V_i \quad = \quad \emptyset \tag{6.17}$$

Für jeden Knoten $u \in V_i$ ist $A(u,i) = \{u\}$. Weil V_0 sämtliche Knoten enthält, gilt also stets $A(u,0) = \{u\}$. In einem 2-Levelgittergraphen (siehe Abbildung 6.13) gilt beispielsweise:

$$\begin{aligned} A((11,10),1) &= \{(12,8),(12,12)\} \\ A((7,4),1) &= \{(8,4)\} \end{aligned}$$

Für jeden Knoten $u \in V_0 - V_1$ und jeden Knoten $v \in V_i$ gibt es einen Knoten aus $A(u,i)$, der auf einem kürzesten Weg zwischen u und v liegt. Diese Eigenschaft werden wir zur Entfernungsberechnung ausnutzen und sukzessive die Knotenmengen $A(u,0), A(u,1), \ldots$ und $A(v,0), A(v,1), \ldots$ bestimmen. Dabei besteht jede Menge von Aufstiegsknoten höchstens aus vier Knoten, wie man sich anhand des 2-Levelgitters in Abbildung 6.13 leicht deutlich macht. Weiterhin liegen für alle Knoten $u \in V_i - V_{i+1}$ die Knoten $A(u,i+1)$ höchstens im Abstand von vier vom Knoten u entfernt, d.h. es gilt:

$$A(u,i+1) \quad \subseteq \quad N_i^4(u) \tag{6.18}$$

Satz 15 *Ein k-Levelgittergraph ist $(s,d,0)$-geeignet für die Parameter*

$$\begin{aligned} s &= (4, 2\cdot 4, 3\cdot 4, \ldots, (k-1)4) \quad \textit{und} \\ d &= (2\cdot 4, 2\cdot 3\cdot 4, \ldots, 2\cdot(k-1)\cdot 4). \end{aligned}$$

Beweis: Für einen Weg $w \in P$ definieren wir das höchste Level als:

$$E_{max}(w) \quad = \quad \max\{j \in \{0,1,\ldots,k-1\} \mid E(w) \cap E_j \neq \emptyset\}$$

Für einen kürzesten Weg $w \in SP(u,v)$ für die Knoten $u, v \in V_0 - V_1$ sei $E_{max}(w) = j$. Dann gibt es einen kürzesten Weg aus $SP(u,v)$, der über die Knoten $A(u,1)$, $A(u,2)$, …, $A(u,j)$, $A(v,j)$, …, $A(v,2)$, $A(v,1)$ verläuft. Nach den obigen Ausführungen gibt es zwei Knoten $x \in A(u,j) \subseteq N_0^{4j}(u)$ und $y \in A(v,j) \subseteq N_0^{4j}(v)$ mit:

$$\delta(u,x) + \delta_j(x,y) + \delta(y,v) = \delta(u,v)$$

Es bleibt zu zeigen, dass tatsächlich für alle Knoten u, v mit $d_0(u,v) > 2 \cdot j \cdot 4$ ein kürzester Weg über den Graphen G_j führt. Wir führen den Nachweis nur für $j = 1$ anhand eines 2-Levelgittergraphen. Für größere Level ergibt sich die Aussage durch Induktion. Nach Theorem 3 sind zum Nachweis alle Knotenpaare im Abstand von neun Kanten zu überprüfen. Für diese Knoten trifft die Aussage zu. Für die meisten auch näher beieinander liegenden Knoten gilt die Aussage ebenso, jedoch nicht für alle. Beispielsweise nicht für $(2,2), (10,2)$:

$$\begin{aligned} \delta((2,2),(10,2)) &= 8 \\ &< \delta((2,2), N^4((2,2)) \cap V_1, N^4((10,2)) \cap V_1, (10,2)) \\ &= 9 \end{aligned}$$

■

Wir geben nun einen adaptierten Algorithmus zur Entfernungsberechnung in k-Levelgittergraphen an.

Algorithmus 12 *Stufenweise-Levelsuche*

Eingabe: *k-Levelgittergraph* $(G_0, G_1, \ldots, G_{k-1})$*, Knoten* $u, v \in V_0 - V_1$

Ausgabe: $\delta'(u,v)$

$\delta'_{-1}(u,v) := \infty$;
$\delta'_0(u,v) := |u_1 - v_1| + |u_2 - v_2|$;
$f(u,u) := 0; f(v,v) := 0$;
$i := 0$;
while $\delta'_i(u,v) < \delta'_{i-1}(u,v)$ **and** $i < k-1$ **do**
  for $u' \in A(u, i+1)$ **do**
    $f(u,u') := min\{f(u,u'') + \frac{|u'_1 - u''_1| + |u'_2 - u''_2|}{4^i} \mid u'' \in A(u,i)\}$
  endfor ;
  for $v' \in A(v, i+1)$ **do**
    $f(v,v') := min\{f(v,v'') + \frac{|v'_1 - v''_1| + |v'_2 - v''_2|}{4^i} \mid v'' \in A(v,i)\}$
  endfor ;
  $\delta'_{i+1}(u,v) := min\{\ f(u,u') + \frac{|u'_1 - v'_1| + |u'_2 - v'_2|}{4^{i+1}} + f(v',v)$
    $\mid u' \in A(u, i+1), v' A(v, i+1)\}$;
  $i := i + 1$;
endwhile ;
if $\delta'_i(u,v) < \delta'_{i-1}(u,v)$ **then return** $\delta'_i(u,v)$
  else return $\delta'_{i-1}(u,v)$
endif ;

Satz 16 *Der Algorithmus Stufenweise-Levelsuche berechnet in einem k-Levelgittergraphen die kürzeste Entfernung in höchstens* $3 \cdot 16 \cdot k$ *Operationen, d.h. es gilt* $\mathcal{Q} \in O(k)$.

Beweis: Angenommen, für ein Knotenpaar $u, v \in V_0 - V_1$ gibt es einen kürzesten Weg $w \in SP(u,v)$ mit $E_{max}(w) = j$, der über die Knoten $A(u,1)$, $A(u,2)$, ..., $A(u,j)$, $A(v,j)$, ..., $A(v,2)$, $A(v,1)$ verläuft. Die Entfernungen zu den Aufstiegsknotenmengen werden sukzessive durch die Funktion f bei jedem Schleifendurchlauf berechnet. Die Entfernung innerhalb des Levels j wird dann über den Betragsabstand (dividiert durch 4^j) zu den Entfernungen zu den Aufstiegsknoten addiert. Es bleibt zu zeigen, dass sich die Entfernung durch Aufstieg in höhere Level stets reduziert. Dies ist der Fall aufgrund der speziellen Struktur von k-Levelgittergraphen. Eine Kante des Levels i kann nur durch vier Kanten des Levels $i-1$ ersetzt werden.

■

k-Levelgittergraphen sind typische Graphen, für die das Eignungskonzept vorteilhafter gegenüber dem Separationskonzept ist, das in Kapitel 7 vorgestellt wird.

6.6.2 Problemkomplexität

Es stellen sich nun zum einen Fragen nach der Berechnung von 'kleinen' fast geeigneten Levelgraphen für vorgegebene Umgebungs-, Entfernungs- und Fehlervektoren und zum anderen Fragen nach der Berechnung von minimalen Umgebungs-, Entfernungs- und Fehlervektoren für vorgegebene Levelgraphen, so dass diese fast geeignet sind.

Korollar 8 *Gegeben seien ein k-Levelgraph G und die Parameter (s,d,ξ), $(s',d',\xi') \in \mathbb{N}_0^{k-1} \times \mathbb{N}_0^{k-1} \times (\mathbb{R}^+)^{k-1}$ mit $(s,d,\xi) \leq (s',d',\xi')$, dann gilt:*

$$G \text{ ist } (s,d,\xi)\text{-geeignet} \implies G \text{ ist } (s',d',\xi')\text{-geeignet}$$

Weil die Entfernungsberechnung im k-Levelgraphen letztlich nur in zwei Leveln stattfindet, definieren wir die Problemstellung nur für einen 2-Levelgraphen.

Problem 6: ***Minimales-Kanten-Skelett MES***

Eingabe: *Ein gewichteter Graph G, ein Umgebungs-, Entfernungs- und Fehlerparameter $s,d \in \mathbb{N}_0, \xi \in \mathbb{R}^+$ und ein Parameter $r \in \mathbb{N}_0$ (maximale Kantenzahl).*

Frage: *Gibt es einen Graphen G' (ein Skelett) mit höchstens r Kanten, so dass der 2-Levelgraph (G,G') (s,d,ξ)-geeignet ist?*

Ein 2-Multilevelgraph G, bei dem beide Level identisch sind, ist offensichtlich geeignet für alle Entfernungs- und Umgebungsparameter. Das zweite Level ist in diesem Fall nutzlos. Eine Lösung von MES für k-Multilevelgraphen erzeugt man, indem obige Problemstellung (k-1) mal gelöst wird. Wenn anstatt der Kantenanzahl die Knotenanzahl im zweiten Level betrachtet wird, bezeichnen wir die Problemstellung als **Minimales-Knoten-Skelett MVS**.

MES und MVS liegen in NP, weil eine nichtdeterministische Turingmaschine eine Kantenmenge bzw. eine Knotenmenge raten und mit polynomiellem Aufwand die Eignung des entstehenden 2-Levelgraphen nach Gleichung 6.14 überprüfen kann.

Korollar 9 *MVS ist NP-vollständig.*

Beweis: Folgt aus Lemma 6 und Satz 7 (HSPS ist NP-vollständig).

■

Satz 17 *MES ist NP-vollständig.*

Beweis: Wir führen eine polynomielle Reduktion des folgenden Problems durch.

Problem 7: ***Restricted-Centers-Clustering RCC***

Eingabe: *Eine Punktemenge $P \subset \mathbb{R}^2$ mit n Punkten, eine Anzahl von Clustern $k \in \{1, \ldots, n\}$ und eine Clustersize $D \in \mathbb{R}^+$.*

Frage: *Gibt es eine Clustercentermenge $C = \{c_1, \ldots, c_k\} \subseteq P$, so dass für alle Punkte $p \in P$ gilt:*

$$d^e(p, C) \quad \leq \quad D \; ?$$

Feder und Greene haben 1988 gezeigt, dass RCC NP-hart ist [FG88].

Wir konstruieren aus der Eingabe von RCC einen Graphen $G = (P, E, \delta)$ mit:

$$\begin{aligned} E &= \{\{u, v\} \subset P \mid d^e(u, v) \leq D\} \\ \delta(\{u, v\}) &= d^e(u, v) \qquad \forall \{u, v\} \in E \end{aligned}$$

O.B.d.A. setzen wir voraus, dass G zusammenhängend ist. Ansonsten betrachte nur eine Teilmenge von P. Außerdem sei $c = \max\{d^e(u, v) \mid u, v \in P\}$. Es gilt offensichtlich für alle Knoten $u \in V$ und alle Knotenmengen $V' \subseteq V$:

$$N(u) \cap V' \neq \emptyset \quad \Longleftrightarrow \quad d^e(u, V') \leq D \tag{6.19}$$

Es gibt genau dann eine Lösung $C \subset P$ mit höchstens $k \in \mathbb{N}$ Centerpunkten für RCC, wenn es einen $(1, 0, 2D + (k-1)c)$-geeigneten 2-Levelgraphen (G, G') mit höchstens $k - 1$ Kanten in G' gibt. Durch den Entfernungsparameter $d = 0$ wird die Eignung für alle Knotenpaare verlangt, und durch den großen zugelassenen Fehler ist es ausreichend, dass in dem zweiten Level lediglich ein Weg existiert.

$\Longrightarrow$ Sei $G' = (C, E', \delta')$ ein beliebig zusammenhängender Graph mit $k-1$ Kanten und euklidischer Gewichtsfunktion δ'. Für $|C| = k$ ist G' also ein Baum. Für alle Knoten $u, v \in P$ gibt es Knoten $x \in N(u) \cap C$ und $y \in N(v) \cap C$ mit:

$$\begin{aligned} \delta(u,x) + \delta'(x,y) + \delta(y,v) &\leq d^e(u,x) + (k-1)c + d^e(y,v) \\ &\leq 2D + (k-1)c \\ &\leq \delta(u,v) + 2D + (k-1)c \end{aligned}$$

Somit ist (G, G') $(1, 0, 2D + (k-1)c)$-geeignet.

$\Longleftarrow$ Angenommen, der Graph im zweiten Level ist nicht zusammenhängend, dann kann eine Zusammenhangskomponente entfernt werden, ohne die Eignungsbedingung zu verletzen. Entferne sukzessive Zusammenhangskomponenten, bis ein zusammenhängendes zweites Level übrig bleibt, dessen Knoten eine Lösung für RCC sind.

■

6.6.3 Approximationen

Gegeben seien ein Graph G und die Parameter $d, k \in \mathbb{N}_0$ und $\xi \in \mathbb{R}^+$. Das **Umgebungsminimum-V** definieren wir als:

$$\begin{aligned} s^V_{\min}(d, \xi, k) &= \min\{s \in \mathbb{N}_0 \mid \exists \text{ ein Skelett } G' \text{ mit höchstens } k \text{ Knoten,} \\ &\qquad \text{so dass } (G, G') \ (s, d, \xi)\text{-geeignet ist.}\} \end{aligned}$$

Sofern die Anzahl der Kanten des Skeletts durch k beschränkt ist, bezeichnen wir das Umgebungsminimum-E als $s^E_{\min}(d, \xi, k)$. Entfernungsminimum-X und Fehlerminimum-X seien für $X \in \{V, E\}$ analog definiert.
Die Approximationsproblemstellungen definieren wir wie folgt.

Problem 8: ***Umgebungs-Approximation-α-V*** ***SA-α-V***

Eingabe: *Ein Graph G mit Entfernungsparameter $d \in \mathbb{N}_+$, Fehler $\xi \in \mathbb{R}_+$ und einer maximalen Knotenzahl $k \in \mathbb{N}_0$.*

Frage: *Gibt es einen Graphen G' (ein Skelett) mit höchstens k Knoten, so dass der 2-Levelgraph (G, G') $(\alpha s_{\min}, d, \xi)$-geeignet ist?*

Wenn die Anzahl der Kanten des Skeletts durch k beschränkt ist, bezeichnen wir die Problemstellung als SA-α-E. Wiederum seien die Problemstellungen zur Approximation der Entfernung und des Fehlers analog definiert.

Korollar 10 *Umgebungs-Approximation-α-X, Entfernungs-Approximation-α-X und Fehler-Approximation-α-X sind für $X \in \{V, E\}$ und jedes konstante $\alpha \geq 1$ NP-vollständig.*

Beweis: Folgt aus den Beweiskonstruktionen der Sätze 7 und 17.

■

6.7 Zusammenfassung und Erweiterungen

In den Abschnitten 6.1 und 6.2 haben wir geeignete Knotenmengen und geeignete Graphen (2-Levelgraphen) zur effizienten Entfernungsberechnung definiert und nachgewiesen, dass die Berechnung eines minimalen zweiten Levels NP-vollständig ist. In Abschnitt 6.3 haben wir verschiedene Heuristiken und eine logarithmische Approximation vorgestellt, die sich effizient berechnen lässt (siehe Satz 12). Anhand eines Beispiels haben wir gezeigt, dass die logarithmische Abweichung bei dem verwendeten Algorithmus auftreten kann. Durch die nachgewiesene Lokalitätseigenschaft in Abschnitt 6.5 kann der Aufwand vom Algorithmus *Eignungstest* und zur Berechnung eines 2-Levelgraphen wesentlich reduziert werden. Im letzten Abschnitt haben wir die Eignung auf k-Levelgraphen übertragen und noch einen Entfernungsparameter in das Eignungskonzept einbezogen. Wir haben außerdem einen speziellen k-Levelgraphen, einen k-Levelgittergraphen, vorgestellt, für den das Eignungskonzept besonders günstig ist. Die konstante Approximation sämtlicher Parameter weisen wir bereits für den Fall $k = 2$ nach.
Die vorgestellte zwei-Level Entfernungsberechnung lässt sich insbesondere bezüglich der Definition von geeigneten Knotenpaaren und Knotenpaarmengen variieren. Wir haben hier die Anzahl der Kanten zum nächsten Knoten als relevanten Parameter benutzt. Man kann auch die Anzahl der nächsten Knoten, die Entfernung oder den euklidischen Abstand (dies ist u.U. in Straßenverkehrsnetzen naheliegend) benutzen. Weitgehend lassen sich die Resultate in diesen Fällen übertragen, die Beweise sind dabei entsprechend anzupassen.
Eine weitere Verallgemeinerung besteht in der Auswahl der Kanten aus $\{\{x, y\} \mid x \in V - S, y \in S\}$ im zweiten Level. Anstatt hier einen globalen Parameter als Umgebungsgröße $s \in \mathbb{N}_0$ zu verwenden, kann auch eine Umgebungsfunktion $s : V \to \mathbb{N}$ verwendet werden, die die Umgebungsgröße abhängig von dem einzelnen Knoten bestimmt. Ein Knotenpaar $u, v \in V$ heißt dann **lokal-(S, s, ξ)-geeignet** (analog zu Definition 4), wenn gilt:

$$\delta(u, N^{s(u)}(u) \cap S, N^{s(v)}(v) \cap S, v) \quad \leq \quad \delta(u, v) + \xi \tag{6.20}$$

Sämtliche Resultate lassen sich auf lokal-geeignete Knotenmengen übertragen. Dabei entspricht die globale Umgebungsgröße einer konstanten Umgebungsfunktion.
Anstatt die Abstandsfunktion für die Auswahl der (Auffahrts-) Knoten im zweiten Level zu verwenden, können diese Knoten auch willkürlich ausgewählt werden. Wir bezeichnen dann ein Knotenpaar $u, v \in V$ als **lokal-(S, ξ)-geeignet** für die Auswahlfunktion $A : V \to 2^V$, wenn die folgende Bedingung erfüllt ist:

$$\delta(u, A(u) \cap S, A(v) \cap S, v) \quad \leq \quad \delta(u, v) + \xi \tag{6.21}$$

In diesem Fall ist allerdings die Abbruchbedingung bei der Wegesuche noch zu klären. Für die k-Levelgittergraphen hatten wir die Abbruchbedingung so gewählt, dass wir, solange es noch eine Verbesserung gab, um ein Level aufgestiegen sind (siehe Algorithmus *Stufenweise-Levelsuche*).

Kapitel 7

Rekursive Separationen

In diesem Kapitel separieren wir Graphen, so dass sämtliche Wege zwischen verschiedenen Knotenmengen über eine Separatormenge verlaufen. Es gibt zahlreiche Untersuchungen in der Graphentheorie, die sich mit diesem Thema unter verschiedenen Begriffen auseinandersetzen: Separatoren, Bisektoren, Artikulations- oder Zerfällungsknoten und Artikulationsmengen, Zusammenhangszahl, Clusterbildung, k-Partition [Har69] [Die96] [Bol78] [Klo94] [KLPS90]. Wir definieren daher die verwendeten Begriffe in diesem Kapitel, soweit sie nicht bereits in Kapitel 2 definiert wurden.
In Abschnitt 7.1 definieren wir Separatorbäume, die sich durch rekursive Separationen konstruieren lassen. Eine Erweiterung der Separatorbäume um eine Kantengewichtsfunktion führt in Abschnitt 7.2 zu Trennenden-Hierarchischen-Graphen, die in einer ähnlichen Variante bereits in [BR97] und in [Rie97] dargestellt sind. Für bestimmte nichtplanare Graphen geben wir in Abschnitt 7.3 die Werte $\mathcal{T}, \mathcal{S}, \mathcal{Q}$ für dieses Separationsverfahren an. In Abschnitt 7.4 führen wir die Separatorweite als einen charakteristischen Parameter eines Graphen ein und zeigen, dass sich Separatorweite und Baumweite höchstens um einen logarithmischen Faktor unterscheiden. Wir konstruieren eine Graphfamilie, für die Separatorweite und Baumweite tatsächlich um einen logarithmischen Faktor voneinander abweichen. Für chordale Graphen konstruieren wir in Abschnitt 7.5 günstige Separatorbäume, deren Separatoren die Cliquengröße des chordalen Graphen nicht überschreiten, d.h. Cliquengröße und Separatorweite chordaler Graphen sind identisch. Der Algorithmus von Tarjan und Yannakakis zur Berechnung einer perfekten Ordnung lässt sich zur Berechnung einer Triangulation eines Graphen erweitern [TY84]. In Abschnitt 7.6 zeigen wir konstruktiv, dass die so berechneten Triangulationen auch für dünne Graphen nicht minimal sind. In Abschnitt 7.7 konstruieren wir für planare Graphen Dünne-Trennende-Hierarchische-Graphen mit wenigen Kanten ($O(n \log n)$) und geben für diese Graphen einen effizienten Online-Algorithmus zur Entfernungsberechnung an ($O(\sqrt{n} \log^2 n)$). Für beliebige Graphen stellen wir in Abschnitt 7.8 einen Hierarchischen-Graphen fast linearer Größe vor, der sich zur Entfernungsapproximation eignet.

Definition 10 *Gegeben seien ein Graph $G = (V, E)$ und die Knotenmengen $Y, X_1, X_2, \ldots, X_k \subseteq V$. Die Knotenmenge Y* **separiert** *die Knotenmengen X_1, X_2 genau dann, wenn gilt:*

$$\forall_{u \in X_1} \forall_{v \in X_2} \forall_{w \in P(u,v)} \qquad V(w) \cap Y \neq \emptyset \tag{7.1}$$

Die Knotenmenge Y separiert die Knotenmengen $X_1, \ldots, X_k$, wenn die Knotenmengen $X_1, \ldots, X_k$ von Y jeweils paarweise separiert werden. Y heißt ein **minimaler Separator**, *wenn Y zwei Knoten $u, v \in V$ mit $u \neq v$ separiert, die nicht in Y sind, und es keine echte Teilmenge von Y gibt, die u und v separiert. Y heißt α-***Separator*** *für $X_1, \ldots, X_k$ $(0 \leq \alpha < 1)$, wenn für $1 \leq i \leq k$ gilt:*

$$|X_i| \quad \leq \quad \alpha n$$

Falls Y nur aus einem Knoten besteht, bezeichnen wir diesen Knoten auch als **Separatorknoten**.

Im Graphen in Abbildung 7.1 ist die Knotenmenge $\{11, 12\}$ ein $\frac{1}{2}$-Separator für die Knotenmengen $\{0, \ldots, 10\}, \{13, \ldots, 23\}$.

Definition 11 (α-β-$f(n)$-Separation [LT79]) *Gegeben seien ein Graph $G = (V, E)$, die Parameter $0 \leq \alpha < 1$, $\beta \in \mathbb{R}^+$ und eine Abbildung $f : \mathbb{N} \longrightarrow \mathbb{R}^+$. Eine Partition A, B, C der Knotenmenge V mit Separator C, die folgende Bedingungen erfüllt:*

- *C ist ein α-Separator für A und B und*
- *$|C| \leq \beta f(n)$*

heißt **α-β-$f(n)$-Separation**.

Das folgende Separatortheorem wurde von Lipton und Tarjan 1979 bewiesen und 1980 auf die Parameter $\alpha = \frac{1}{2}$ und $\beta = \frac{2\sqrt{2}}{1-\sqrt{\frac{2}{3}}} \leq 15.5$ erweitert [LT79] [LT80]. Für $\alpha = \frac{1}{2}$ hat Chung 1991 $\beta = 3\sqrt{6} \approx 7.35$ und für $\alpha = \frac{2}{3}$ haben Djidjev und Venkatesan 1997 $\beta = \sqrt{\frac{2}{3}} + \sqrt{\frac{4}{3}} \approx 1.97$ nachgewiesen [Chu91] [DV97].

Theorem 4 (Separatortheorem [LT79] [LT80]) *Für planare Graphen kann eine $\frac{2}{3}$-$2\sqrt{2}$-$\sqrt{n}$-Separation mit linearem Aufwand ($O(n)$) berechnet werden.*

Djidjev entwickelt in [Dji96] ein skalierbares Verfahren zur Online-Kürzesten-Wegesuche für planare Graphen. Er zeigt, dass für planare Graphen für jedes $M \in [n^{\frac{4}{3}}, n^{\frac{3}{2}}]$ eine Datenstruktur der Größe $O(M)$ in $O(n\sqrt{M})$ Operationen berechnet werden kann, so dass SPSP in $O(\frac{n}{\sqrt{M}} \log n)$ Operationen gelöst werden kann. Das Produkt SQ wird für $M = n^{\frac{4}{3}}$ minimiert, d.h. es gilt: $SQ \in O(n^{\frac{5}{3}} \log n)$. Wir zeigen in Abschnitt 7.7 mit einem modifizierten Verfahren $SQ \in O(n^{1.5} \log^3 n)$ für planare Graphen.

Djidjev kombiniert r-Divisionen, die von Frederickson in [Fre87] entwickelt wurden, und Separatorbäume, die Lingas in [Lin90] einführte, zu einer Datenstruktur. Zusammengefasst wiedergegeben wird innerhalb jeder Region einer r-Division ein Separatorbaum aufgebaut und zwischen allen Grenzknoten der Regionen ('boundary vertices' in [Fre87]) ein vollständiger Graph. Bei der SPSP für ein Knotenpaar u, v werden im ersten Schritt die Entfernungen zu den Grenzknoten $B(u), B(v)$ der jeweiligen Regionen bestimmt. In einem zweiten Schritt werden die Weglängen über die Grenzknotenmengen $B(u)$ und $B(v)$ in $O(|B(u)| \log |B(v)| + |B(v)| \log |B(u)|)$ Operationen minimiert. Man beachte, dass ein trivialer Ansatz dabei zu $O(|B(u)||B(v)|)$ Operationen führt.

7.1 Separatorbaum

Definition 12 *Gegeben sei ein Graph $G = (V, E)$. Ein gerichteter Baum $T = (Q, E_Q, l)$ mit Knotenmenge Q, Wurzel $q_0 \in Q$, Kantenmenge E_Q und Labelfunktion $l : Q \to 2^V \times 2^V$ heißt* **Separatorbaum**, *wenn für jeden Knoten $q \in Q$ mit Söhnen $q_1, \ldots, q_j \in Q$ für $j \geq 1$ gilt:*

1. *$l(q_0) = (V, S)$ für eine Knotenmenge $S \subseteq V$,*
2. *$l^2(q), l^1(q_1), \ldots, l^1(q_j)$ sind eine Partition von $l^1(q)$,*
3. *$\bigcup_{q \in Q} l^2(q) = V$ und*
4. *$l^2(q)$ separiert $l^1(q_1), \ldots, l^1(q_j)$ in dem induzierten Graphen $G[l^1(q)]$ von G.*

Wir führen einige weitere Bezeichnungen für die Knoten von Level $0 \leq i \leq H(T)$, die größte Separatormenge von Level $0 \leq i \leq H(T)$ und die größte Separatormenge im gesamten Separatorbaum T ein.

$$Q_i = \{q \in Q \mid N^i(q_0) - N^{i-1}(q_0)\} \tag{7.2}$$

$$l_{max,i} = \max\{|l^2(q)| \mid q \in Q_i\} \tag{7.3}$$

$$l_{max} = \max\{|l^2(q)| \mid q \in Q\} \tag{7.4}$$

T heißt ein **günstiger Separatorbaum**, *wenn die Bedingungen 2 und 4 durch folgende Bedingungen ersetzt werden:*

2'. *$l^2(q) \cup \bigcup_{i=1}^{j} l^1(q_i) = l^1(q)$ und*

4'. *$l^2(q)$ separiert $l^1(q_1), \ldots, l^1(q_j)$ in G.*

*T heißt α-β-$f(n)$-***Separatorbaum** *für $0 \leq \alpha < 1, \beta \in \mathbb{R}^+, f : \mathbb{N} \longrightarrow \mathbb{R}^+$, wenn für jeden Knoten $q \in Q_i$ für $0 \leq i \leq H(T)$ gilt:*

1. *$|l^1(q)| \leq \alpha^i n$ und*
2. *$|l^2(q)| \leq \beta f(|l^1(q)|)$*

Wenn die Knotenmengen $l^2(q)$ für eine Familie von Graphen sogar durch eine Konstante $k \in \mathbb{N}$ (unabhängig von n) beschränkt sind, d.h. es gilt $l_{max} \leq k$, dann sprechen wir auch von einem α-k-Separatorbaum.

Wir veranschaulichen uns die Definition an dem Separatorbaum in Abbildung 7.2 zum Graphen aus Abbildung 7.1. In den Knoten des Separatorbaums sind jeweils die Separatormengen ($l^2(q)$) dargestellt und unter dem Baum die induzierten Teilgraphen vom zugrunde liegenden Graphen. Man beachte, dass für die Blätter $q \in Q$ jedes Separatorbaums gilt:

$$l^1(q) \quad = \quad l^2(q). \tag{7.5}$$

In günstigen Separatorbäumen ist dies wegen Gleichung 2' in der obigen Definition nicht unbedingt erforderlich. Wir werden jedoch nur günstige Separatorbäume betrachten, die die Gleichung 7.5 erfüllen.

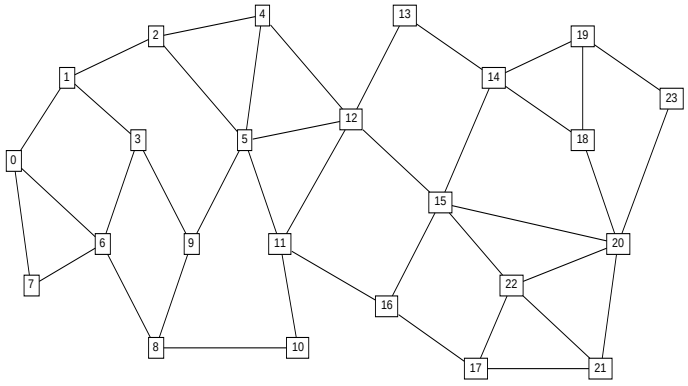

Abbildung 7.1: Beispielgraph zu Definition 12.

Die Größen der Knotenmengen l^1 halbieren sich jeweils von Level zu Level, und die größte Separatormenge l^2 hat die Größe drei, d.h. es handelt sich in Abbildung 7.2 um einen $\frac{1}{2}$-3-Separatorbaum.
Der Nutzen von Separatorbäumen ergibt sich aus dem folgenden Lemma.

Lemma 7 *Gegeben seien ein Graph $G = (V, E)$, ein Separatorbaum T und zwei Knoten $u, v \in V$. $q \in Q$ sei der Knoten in T mit maximaler Tiefe und $u, v \in l^1(q)$. P sei der Weg von der Wurzel q_0 zu q in T. Jeder Weg zwischen u und v in G führt über mindestens einen Knoten aus $\bigcup_{q' \in V(P)} l^2(q')$.*

Man beachte, dass aufgrund der Partitionsbedingung in Definition 12 die Knoten aus Separatormengen in tieferen Leveln eines Separatorbaums nicht vorkommen. In günstigen Separatorbäumen können diese Knoten in tieferen Leveln durchaus auftreten.

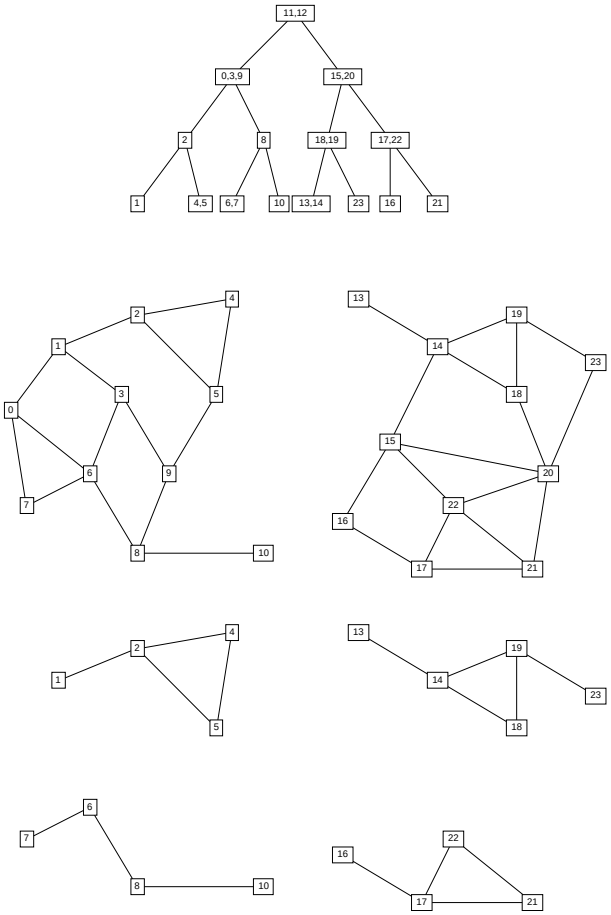

Abbildung 7.2: Separatorbaum mit induzierten Teilgraphen zu Abb. 7.1.

Korollar 11 *Nach Lemma 7 kann zu einem Graphen mit Separatorbaum $T = (Q, E_Q, l)$ wie folgt ein günstiger Separatorbaum $T' = (Q, E_Q, l')$ aufgebaut werden. Sei P_q der Pfad in T (oder T') vom Knoten q zur Wurzel. Setze l' wie folgt für alle $q \in Q$:*

$$l'(q) \quad := \quad (l^1(q) \cup \bigcup_{x \in P_q} l^2(x), \bigcup_{x \in P_q} l^2(x))$$

Diese Konstruktionsmethode haben wir auf den Separtorbaum aus Abbildung 7.2 angewendet und in Abbildung 7.3 den resultierenden Separatorbaum dargestellt. Man erkennt, dass der Separatorbaum dabei unnötig groß werden kann. In Abbildung 7.4 ist ein günstiger $\frac{1}{2}$-4-Separatorbaum zum Graphen aus Abbildung 7.1 dargestellt, aus dessen Separatormengen diese unnötigen Knoten entfernt

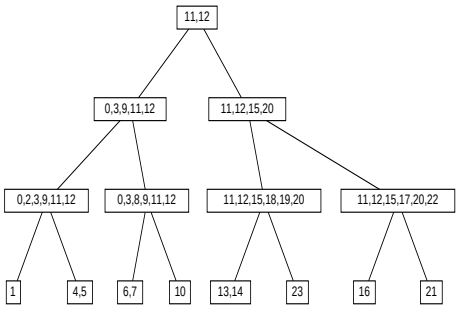

Abbildung 7.3: Günstiger Separatorbaum nach Konstruktion aus Korollar 11 zum Separatorbaum aus Abbildung 7.2.

wurden. Dennoch ist der günstige Separatorbaum größer als der herkömmliche Separatorbaum, weil schließlich die Separatormengen im gesamten Graphen und nicht nur auf induzierten Teilgraphen gültig sind. Günstige Separatorbäume erfordern also einen höheren Speicheraufwand, ermöglichen aber auch eine effizientere Entfernungsberechnung, wie im folgenden Abschnitt gezeigt wird (siehe Theoreme 6 und 7).

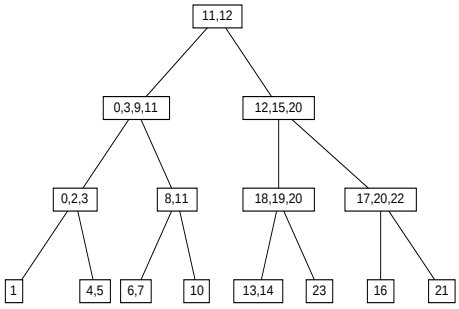

Abbildung 7.4: Günstiger $\frac{1}{2}$-4-Separatorbaum.

Da jede Separatormenge ($l^2(u)$) eines Knotens u die Knotenmengen der Söhne ($l^1(v)$) in G separiert, genügt es zur Bestimmung der kürzesten Entfernung, zwischen zwei Knoten genau einen Knoten des Separatorbaums und nicht alle Knoten auf einem Pfad zur Wurzel zu betrachten. Wir formulieren dies als Lemma analog zu Lemma 7.

Lemma 8 *Gegeben seien ein Graph $G = (V, E)$, ein günstiger Separatorbaum T und zwei Knoten $u, v \in V$. $q \in Q$ sei der Knoten in T mit maximaler Tiefe und $u, v \in l^1(q)$. Jeder Weg zwischen u und v in G führt über mindestens einen Knoten aus $l^2(q)$.*

Man beachte, dass die Höhe eines α-β-$f(n)$-Separatorbaums, dessen Labelfunktion nur auf nichtleere Knotenmengen verweist, höchstens $O(\log n)$ ist. Für planare Graphen setzen wir $f(n) = \sqrt{n}$, weil es für planare Graphen Separatoren der Größe $O(\sqrt{n})$ gibt (siehe Separatortheorem). Da die konkreten Werte von α, β bei unseren Untersuchungen keine Rolle spielen, nehmen wir im Weiteren an, dass die Parameter passend gewählt sind. Aus Gründen der Generalisierung benutzen wir die Funktion $f : \mathbb{R}^+ \to \mathbb{R}^+$ statt einer Wurzelfunktion, wobei wir bei den Komplexitätsabschätzungen folgende Eigenschaften der Funktion $f()$ voraussetzen, die beispielsweise von allen Funktionen $f(x) = x^c$ für $c \in \mathbb{R}^+$ erfüllt werden:

$$f(xy) \quad \leq \quad f(x)f(y) \quad \forall x, y \in \mathbb{R}^+ \tag{7.6}$$

$$0 \leq f(z) < 1 \quad \text{für} \quad 0 \leq z < 1 \tag{7.7}$$

Aus den Gleichungen 7.6 und 7.7 folgt, dass die Funktion $f()$ echt monoton wächst und $f(1) \geq 1$. Konstante Funktionen erfüllen diese Gleichungen nicht (außer für die Konstante $f(x) = 0$). Dies ist sinnvoll, weil dann Laufzeit und Speicherplatz nicht mehr in Abhängigkeit von $f()$ angegeben werden können, sondern zusätzlicher Organisationsoverhead nötig ist (siehe Theoreme 6 und 7).

Theorem 5 *Gegeben seien ein planarer Graph $G = (V, E)$ und passende Parameter α, β. Ein α-β-$\sqrt{n}$-Separatorbaum lässt sich in $O(n \log n)$ Operationen aufbauen.*

Beweis: Wir wenden das Separatortheorem rekursiv auf einen planaren Graphen an und konstruieren in $O(n \log n)$ Operationen einen binären Separatorbaum T. Für die Wurzel $q_0 \in V(T)$ setzen wir $l^1(q_0) = V$. Sei $q \in V(T)$ ein Knoten von T und A, B, C eine α-β-$\sqrt{|l^1(q)|}$-Separation mit Separator C von $G[l^1(q)]$, die nach dem Separatortheorem in linearer Laufzeit in $|l^1(q)|$ berechnet werden kann. Füge die Knoten q_1, q_2 als Nachfolger von q in T ein und setze:

$$\begin{aligned} l^2(q) &:= C \\ l^1(q_1) &:= A \\ l^1(q_2) &:= B \end{aligned}$$

Weil die Knotenmengen in einer Tiefe des Separatorbaums alle disjunkt sind, d.h. für die Knoten $q, q' \in Q$ $(q \neq q')$ einer Tiefe gilt

$$l^1(q) \cap l^1(q') \quad = \quad \emptyset \,,$$

folgt die Aussage.

■

7.2 Trennende-Hierarchische-Graphen

Wir erweitern in diesem Abschnitt unsere Betrachtungen von ungewichteten auf gewichtete Graphen und übertragen sämtliche bereits definierten Begriffe. Wir setzen dabei die Existenz eines $f(n)$-Separatorbaums für einen Graphen bzw. eine Familie von Graphen voraus.

Definition 13 (Trennender-Hierarchischer-Graph THG) *Gegeben seien ein Graph $G = (V, E, \delta)$ und ein Separatorbaum $T = (Q, E_Q, l)$. Jeden Knoten $q \in Q$ markieren wir mit einem Graphen $G_q(l^1(q), [l^1(q), l^2(q)], \hat{\delta}_q)$. δ'_q sei die Kantenfunktion in dem von $l^1(q)$ in G induzierten Graphen $G[l^1(q)]$ und für alle Kanten $\{u, v\} \in [l^1(q), l^2(q)]$ gilt:*

$$\hat{\delta}_q(u, v) \quad = \quad \delta'_q(u, v)$$

*Wenn der zugrunde liegende Separatorbaum T ein α-β-$f(n)$-Separatorbaum ist, bezeichnen wir den Trennenden-Hierarchischen-Graphen als α-β-$f(n)$-**THG** zu G.*

Trennende-Hierarchische-Graphen ergeben sich eindeutig aus den Separatorbäumen. Zum Separatorbaum T aus Abbildung 7.2 haben wir in Abbildung 7.5 die Graphen $G_q(l^1(q), [l^1(q), l^2(q)], \hat{\delta}_q)$ dargestellt, mit denen die Knoten außer der Wurzel und den Blättern markiert werden.
Wir werden einen THG auch als flachen Graphen $G' = (V, E', \hat{\delta})$ auffassen, wobei E' genau die Vereinigung aller Kanten aus dem THG enthält, und die Kantengewichtsfunktion wird ebenso aus den Graphen $\{G_q \mid q \in Q\}$ übernommen.

$$E' \quad = \quad \bigcup_{q \in Q} [l^1(q), l^2(q)]$$

Wenn wir von den Kanten des THG sprechen, meinen wir genau die Kanten E'. Zur Lösung von SPSP im Graphen $G = (V, E, \delta)$ für ein Knotenpaar $u, v \in V$ bestimmen wir den Knoten $q \in Q$ mit maximaler Tiefe im Separatorbaum T, so dass $u, v \in l^1(q)$ gilt. P sei der Weg von q zur Wurzel von T. Nach Lemma 7 führen alle Wege aus $P^G(u, v)$ über einen Knoten von $\bigcup_{v \in V(P)} l^2(v)$. Es gibt dann einen Knoten $q' \in V(P)$, so dass ein kürzester Weg $w \in SP^G(u, v)$ vollständig in $G[l^1(q')]$ liegt und somit über einen Knoten aus $l^2(q')$ verläuft. Also gilt:

$$\delta(u, v) \quad = \quad \min\{\hat{\delta}_v(u, x) + \hat{\delta}_v(x, v) \mid x \in \bigcup_{v \in V(P)} l^2(v)\} \tag{7.8}$$

Die Kantengewichtsfunktionen $\hat{\delta}_q$ der markierten Graphen G_q eines THG tabellieren wir für alle Knoten $q \in V(T)$ des zugehörigen Separatorbaums T. Somit kann in konstanter Zeit auf diese Kantengewichte zugegriffen werden. Genau genommen hätten wir logarithmischen Aufwand zu berücksichtigen, wenn die

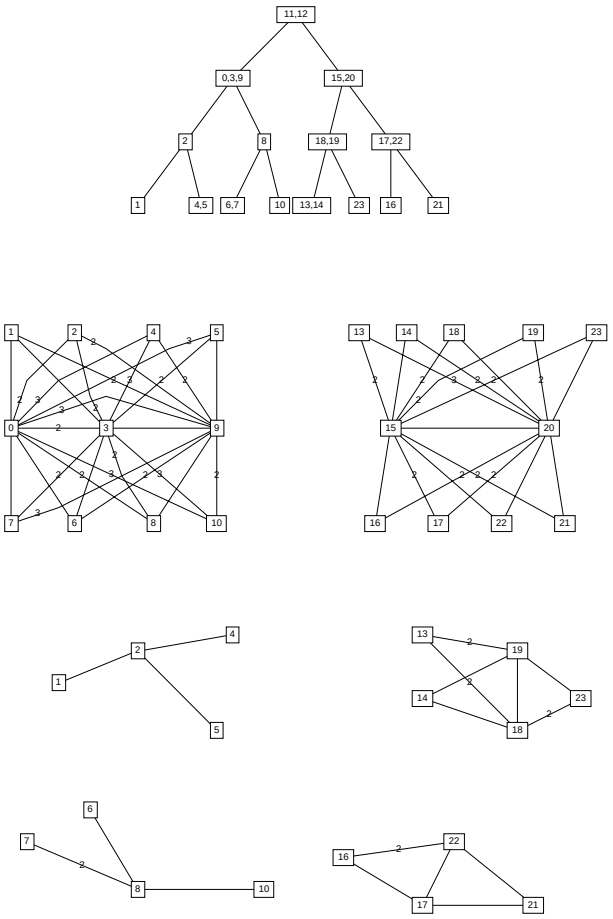

Abbildung 7.5: THG zum Graphen aus Abbildung 7.1.

Größe n (Anzahl der Knoten) nicht mehr in konstanter Zeit im Rechner dargestellt werden kann. Wir benutzen jedoch ein Registermaschinenmodell, das konstanten Zugriff auf beliebige Zahlen erlaubt.

Theorem 6 *Gegeben seien ein Graph G und ein α-β-$f(n)$-THG zu G, wobei die Funktion $f()$ die Gleichungen 7.6 und 7.7 erfüllt. SPSD kann in $O(f(n))$ gelöst werden.*

Beweis: Gegeben sei ein Weg P von einem Knoten aus Level k des α-β-$f(n)$-THG T zur Wurzel von T. Nach obiger Gleichung 7.8 ist zu zeigen, dass die Knotenmenge $\bigcup_{q \in V(P)} l^2(q)$ von der entsprechenden Größenordnung ist.

$$
\begin{aligned}
|\bigcup_{q\in V(P)} l^2(q)| &\leq \sum_{i=0}^{k} \beta f(\alpha^i n) && (7.9)\\
&\underbrace{\leq}_{7.6} \beta f(n) \sum_{i=0}^{\infty} (f(\alpha))^i && (7.10)\\
&\underbrace{\in}_{7.7} O(f(n)) && (7.11)
\end{aligned}
$$

■

Für den Fall eines planaren Graphen kann ein $\sqrt{n}$-THG aufgebaut werden, so dass SPSD in $O(\sqrt{n})$ gelöst wird. Falls es sich bei der Funktion $f()$ um eine konstante Funktion handelt, die Gleichungen 7.6 und 7.7 also nicht gelten, ist für SPSD mit einem Aufwand proportional zur Höhe des Separatorbaums, d.h. $O(\log n)$ zu rechnen. Für Funktionen $f(x) \leq \log x$ ist SPSD in $O(\log^2 n)$ Operationen zu berechnen, wenn ein α-β-$f(n)$-THG in der entsprechenden Datenstruktur zur Verfügung steht.

Theorem 7 *Ein α-β-$f(n)$-THG zu einem Graphen mit n Knoten hat die Größe $O(nf(n))$, wenn die Funktion $f()$ die Gleichungen 7.6 und 7.7 erfüllt.*

Beweis: Der THG T habe die Tiefe k und $Q_i \subseteq V(T)$ seien die Knoten aus Level $0 \leq i \leq k$. Wir zeigen, dass die Kantenmenge $E(T)$ von der entsprechenden Größenordnung ist.

$$
\begin{aligned}
|\bigcup_{q\in V(T)} [l^1(q), l^2(q)]| &= \sum_{q\in V(T)} |[l^1(q), l^2(q)]|\\
&\leq \sum_{q\in V(T)} |l^1(q)||l^2(q)|\\
&= \sum_{i=0}^{k} \sum_{q\in Q_i} |l^1(q)||l^2(q)|\\
&\underbrace{\leq}_{Gl.7.3} \sum_{i=0}^{k} l_{max,i} \sum_{q\in Q_i} |l^1(q)|\\
&\leq n \sum_{i=0}^{k} l_{max,i}\\
&\leq n \sum_{i=0}^{k} \beta f(\alpha^i n)\\
&\underbrace{\in}_{7.6,7.7} O(nf(n))
\end{aligned}
$$

Die Summation über die Knotenmengen $l^1(q)$ für alle Knoten eines Levels $q \in Q_i$ kann mit n der Anzahl aller Knoten abgeschätzt werden, weil die Knotenmengen disjunkt sind.

■

Für konstante Funktionen $f()$ hat ein α-β-$f(n)$-THG zu einem Graphen mit n Knoten höchstens eine Größe von $O(n \log n)$. Für Funktionen $f()$ mit $f(n) \leq \log n$ benötigt der α-β-$f(n)$-THG höchstens eine Größe von $O(n \log^2 n)$. Die Beweise verlaufen analog zum Beweis von Theorem 7.
Wir wenden uns nun der Frage nach dem Berechnungsaufwand eines THG zu. Für planare Graphen wenden wir einen linearen Algorithmus für SSSP an, während wir für den allgemeinen Fall ein spezielles Verfahren angeben, so dass die Berechnung fast linear in der Größe des THG bleibt.

Theorem 8 *Für einen planaren Graphen mit n Knoten kann ein α-β-$\sqrt{n}$-THG in $O(n^{1.5})$ Operationen aufgebaut werden.*

Beweis: Nach Theorem 7 hat der THG die Größe $O(n^{1.5})$. Q sei die Knotenmenge des zugehörigen Separatorbaums, und für jeden Knoten $q \in Q$ sei G_q ein markierter Graph mit den Knoten $l^1(q)$ und den Kanten $[l^1(q), l^2(q)]$. Die Bestimmung der Kantengewichte in G_q erfolgt durch $|l^2(q)|$-malige Lösung vom SSSP mittels des Algorithmus von Klein u.a. in dem von $l^1(q)$ in G induzierten planaren Graphen in linearer Zeit, d.h. in $O(|l^1(q)||l^2(q)|)$ Operationen [KRRS94]. Die Aussage folgt durch eine Summation ähnlich wie im Beweis zu Theorem 7.

■

Theorem 9 *Gegeben sei ein Graph mit α-β-$f(n)$-Separatorbaum, wobei die Funktion $f(n)$ die Gleichungen 7.6 und 7.7 erfüllt. Ein α-β-$f(n)$-THG lässt sich in $O((m + n \log n) f(n))$ Operationen berechnen.*

Beweis: Gegeben sei ein Graph mit n Knoten, m Kanten und α-β-$f(n)$-THG T der Tiefe k. Q_i seien die Knoten von T in Tiefe $0 \leq i \leq k$. Wir wenden den folgenden Algorithmus zur Berechnung aller Entfernungen im Level i an.

Algorithmus 13 *Verteilter Dijkstra*

Eingabe: *r disjunkte Graphen $G_j = (V_j, E_j, \delta_j)$ mit Startknoten $s_j \in V_j$ $(1 \leq j \leq r)$*

Ausgabe: *$\delta'(u)$ für alle Knoten $u \in V_j$ $(1 \leq j \leq r)$*

$(V, E, \delta) := \bigcup_{j=1}^{r} G_j$; Auf diesen Graphen beziehen sich die Funktionen $Next$ und $Update$.

$S := \{s_1, \ldots, s_r\}$;
for $u \in V$ **do** $\delta'(u) := \infty$ **endfor** ;
for $j := 1$ **to** r **do**
  for $u \in N(s_j)$ **do** $\delta'(u) := \delta(\{s_j, u\})$ **endfor** ;
endfor ;
while $AR(S) \neq \emptyset$ **do**
  $u := Next(\delta', S)$;
  $Update(u, \delta', S)$;
endwhile ;

Aufgrund der Disjunktivität der Graphen berechnet der *Verteilte Dijkstra* Algorithmus die Entfernungen $\delta'(u) = \delta_j(s_j, u)$ für alle Knoten $u \in V_j$ $(1 \leq j \leq r)$. Die Laufzeit ist $O(m + n \log n)$, wenn der Rand als Fibonacci-Heap organisiert wird (siehe Kapitel 3).

Zur Entfernungsberechnung in Level i führen wir $l_{max,i} \leq \beta f(\alpha^i n)$ mal den *Verteilten Dijkstra* Algorithmus durch, wobei wir bei jedem Durchlauf einen anderen Startknoten aus jeder der Knotenmengen $l^2(q)$ für $q \in Q_i$ wählen. Die induzierten Graphen $G[l^1(q)]$ bilden die disjunkten Eingabegraphen. Insgesamt ergibt sich ein Aufwand in Level i von $O((m + n \log n)\beta f(\alpha^i n))$ Operationen. Die Summation über die k Level ergibt analog zur Summation im Beweis zu Theorem 7 eine Laufzeit $O((m + n \log n) f(n))$.

■

Korollar 12 *Gegeben sei ein Graph mit α-β-$f(n)$-Separatorbaum. Für jede konstante Funktion $f()$ lässt sich ein α-β-$f(n)$-THG T in $O((m + n \log n) \log n)$ Operationen berechnen. Für jede monoton wachsende Funktion $f(x) \leq \log x$ wird in $O((m + n \log n) \log^2 n)$ Operationen ein α-β-$f(n)$-THG berechnet.*

Beweis: In Level $0 \leq i \leq H(T)$ werden sämtliche Entfernungen mittels des *Verteilten Dijkstra* Algorithmus in $O((m + n \log n) l_{max,i})$ Operationen berechnet (siehe Beweis Theorem 9). Die Summation über $O(\log_{\frac{1}{\alpha}} n)$ Level ergibt die obigen Aussagen.

■

7.3 Spezielle nichtplanare Graphen

Wir geben hier basierend auf den letzten Abschnitten die konkreten Aufwandsanalysen für k-planare Graphen, Graphen ohne K_h-Minoren und Graphen mit Geschlecht g an.

Definition 14 (k-planar) *Ein Graph $G = (V, E)$ heißt* **k-planar** *für $k \in \mathbb{N}_0$, wenn es eine Einbettung von G in die Ebene $\mathbb{R}^2$ gibt, so dass jede Kante höchstens von k Kanten gekreuzt wird.*

Die 0-planaren Graphen stimmen genau mit den planaren Graphen überein.
Sei G ein k-planarer Graph und G' ein planarer Teilgraph von G, der bezüglich der Einbettung von G, für die G k-planar ist, maximal in der Kantenzahl ist. T' sei ein α-β-$\sqrt{n}$-THG von G', der nach Theorem 8 existiert. Die Separatoren des Graphen G' können nach Miller so gewählt werden, dass sie einfache Kreise sind [Mil86] (siehe Satz 21 in Abschnitt 7.7). Im Graphen G wird jede Kante eines Separators durch maximal k Kanten gekreuzt. Für jede kreuzende Kante wird ein Endknoten zu dem entsprechenden Separator gefügt, so dass die Separatoreigenschaft auch für den Graphen G erfüllt wird. Es entsteht ein α-β-$\sqrt{n}$-THG der Größe $O(kn^{1.5})$, und für SPSD gilt: $\mathcal{Q} \in O(k\sqrt{n})$.
Alon u.a. untersuchten in [AST90] Graphen, die keinen K_h-Minor besitzen. Ein Graph G besitzt keinen H-**Minor**, wenn kein Teilgraph $G' = (V', E')$ von G existiert, so dass aus G' durch Kontraktionen der Graph H hervorgeht. Planare Graphen werden genau durch Graphen charakterisiert, die keinen $K_{3,3}$-Minor und keinen K_5-Minor besitzen. In $O(n\sqrt{hn})$ Operationen kann für Graphen, die keinen K_h-Minor besitzen, ein Separator der Größe $h\sqrt{hn}$ berechnet werden. Folglich werden THG's der Größe $\mathcal{S} \in O((hn)^{1.5})$ in der Zeit $\mathcal{T} \in O((m + n \log n)h\sqrt{hn})$ berechnet und für SPSD gilt: $\mathcal{Q} \in O(h\sqrt{hn})$.
Ein Graph mit **Geschlecht** $g \in \mathbb{N}_0$ lässt sich auf einer Kugel mit g Henkeln kreuzungsfrei einbetten. Für Familien von Graphen mit beschränktem Geschlecht $g \in \mathbb{N}_0$ haben Gilbert u.a. in [GHT84] gezeigt, dass sich Separatoren der Größe $\sqrt{gn}$ in Linearzeit $O(n + g)$ finden lassen. Somit lassen sich THG's der Größe $O(\sqrt{g}n^{1.5})$ in $O((m + n \log n)\sqrt{gn})$ Operationen aufbauen, und SPSD kann in $O(\sqrt{gn})$ Operationen gelöst werden.

7.4 Separatorweite und Baumweite

In diesem Abschnitt betrachten wir nur günstige Separatorbäume, weil für diese SPSD durch Betrachtung genau einer Separatormenge gelöst wird. Für eine effiziente Entfernungsberechnung benötigen wir kleine Separatoren (die obigen Mengen $l^2(q)$), und zur effizienten Speicherung ist eine niedrige Höhe des Separatorbaums von Vorteil.

Definition 15 *Gegeben sei ein Graph G mit n Knoten und günstigem Separatorbaum $T = (Q, E_Q, l)$. T heißt* **günstiger α-Separatorbaum** *von G, wenn für alle Knoten $q \in Q_i$ für $0 \leq i \leq H(T)$ gilt:*

$$|l^1(q)| \quad \leq \quad \alpha^i n$$

$l_{max} - 1$ heißt die **Weite des günstigen Separatorbaums** *T (Schreibweise $W(T) = l_{max} - 1$).*

Man beachte, dass wir die Weite nur für günstige Separatorbäume definiert haben. Die Höhe eines günstigen α-Separatorbaums ist folglich durch $H(T) \leq \log_{\frac{1}{\alpha}} n$ beschränkt. Im Weiteren betrachten wir stets den Logarithmus zur Basis $\frac{1}{\alpha}$.

Lemma 9 *Jeder günstige α-Separatorbaum kann in einen günstigen α-Separatorbaum mit höchstens n Knoten transformiert werden.*

Beweis: Sei $T = (Q, E_Q, l)$ ein günstiger α-Separatorbaum zum Graphen G. Die Knotenmenge $l^2(q)$ separiert die Knotenmengen $l^1(q'), l^1(q'')$ im Graphen G für alle Nachbarknoten $q', q'' \in N(q)$. Daraus folgt: $l^1(q') \cap l^1(q'') \subseteq l^2(q)$. Für den Fall $l^2(q') \subseteq l^2(q)$ entferne den Knoten q' von T, und füge die Knoten $N(q')$ als direkte Nachfolger an den Knoten q, so dass der Separatorbaum T' entsteht.

$$\begin{aligned} T' &= (Q - \{q'\}, E_Q - \{(q,q')\} - \{(q',x) \mid x \in N(q')\} \\ &\quad \cup \{(q,x) \mid x \in N(q')\}, l_{|Q-\{q'\}}) \end{aligned}$$

T' erfüllt offensichtlich für alle Knoten $q \in Q_i$ die Gleichung $|l^1(q)| \leq \alpha^i n$ und ist somit ein günstiger α-Separatorbaum. Wir entfernen solange Knoten des Separatorbaums, bis für alle benachbarten Knoten $q' \in N(q)$ gilt: $l^2(q') \not\subset l^2(q)$. Dann gibt es in jeder Separatormenge $l^2(q')$ mindestens einen Knoten, der nicht in der Separatormenge des Vaterknotens vorkommt, und somit gilt: $|Q| \leq |V(G)| = n$.

■

Definition 16 (Separatorweite) *Gegeben sei ein Graph $G = (V, E)$. Die minimale Weite aller günstigen α-Separatorbäume von G heißt die* **α-Separatorweite** *von G (Schreibweise $S_\alpha(G)$).*

Für jeden Graphen G mit günstigem α-Separatorbaum T gilt: $S_\alpha(G) \leq W(T)$. Wir klären nun, wie günstige Separatorbäume zur Entfernungsberechnung eingesetzt werden, und führen dazu Günstige-Trennende-Hierarchische-Graphen GT-HG ein. Gegeben seien ein Graph $G = (V, E, \delta)$, ein günstiger α-Separatorbaum $T = (Q, E_Q, l)$ mit weniger als n Knoten (Lemma 9) und eine Abbildung $g : V \to Q$ mit:

$$g(u) = q \implies u \in l^2(q) \qquad u \in V \tag{7.12}$$

Eine Abbildung g mit dieser Eigenschaft existiert, weil die Separatormengen von T alle Knoten von G überdecken. Von der Labelfunktion l verwalten wir nur die zweite Komponente explizit für jeden Knoten $q \in Q$ in Form einer Liste (die Separatormengen $l^2(q)$). Die ersten Komponenten sind implizit durch die Teilbäume gegeben. Sei $q \in Q$ die Wurzel des vollständigen Teilbaums T_q von T. Dann ist:

$$l^1(q) = \bigcup_{q' \in V(T_q)} l^2(q')$$

Für die Labelfunktion l ist also höchstens ein Speicherplatz von $nW(T)$ zu reservieren.

Definition 17 *Gegeben seien ein Graph G mit günstigem α-Separatorbaum T mit weniger als n Knoten. $H = (V(G), E', \delta')$ heißt* **Günstiger-Trennender-Hierarchischer-Graph GTHG***, wenn für alle Knoten $u, v \in V(G)$ gilt:*

$$\begin{array}{rcll} \{u,v\} \in E' & \Longleftrightarrow & \exists q \in Q : u \in l^2(q) \wedge v \in l^1(q) & \\ \delta'(\{u,v\}) & = & \delta(u,v) & \forall \{u,v\} \in E' \end{array}$$

Im GTHG werden also alle Knoten einer Separatormenge $l^2(q)$ mit allen Knoten des Teilbaums T_q durch Kanten miteinander verbunden. Die Kanten werden außerdem mit der kürzesten Entfernung zwischen den Knoten im gesamten Graphen gewichtet und nicht wie im THG mit der kürzesten Entfernung in einem induzierten Teilgraphen von G.

Theorem 10 *Gegeben seien ein Graph G mit günstigem α-Separatorbaum $T = (Q, E_Q, l)$ mit weniger als n Knoten und einem GTHG H. Es gilt:*

$$\mathcal{S} \in O(W(T) n \log_{\frac{1}{\alpha}} n) \tag{7.13}$$

$$\mathcal{Q} \in O(W(T)) \tag{7.14}$$

Beweis: Für den Speicherplatz ist zu zeigen, dass die Kantenmenge des GTHG H von der entsprechenden Größenordnung ist. Wir modifizieren die Beweisführung von Theorem 7 etwas, weil die Knotenmengen $l^1(q)$ für $q \in Q_i$ (für $0 \leq i \leq H(H)$) i.A. nicht disjunkt sind. Für jeden Knoten $q \in Q$ sei P'_q der Weg von q zur Wurzel von T und $P_q = Pr(P'_q, |P'_q| - 1)$.

Wenn die Kanten von H ausgehend von der Wurzel von T top-down eingefügt werden, dann sind am Knoten $q \in Q$ nicht mehr $|l^1(q)||l^2(q)|$ Kanten einzufügen, weil alle Kanten mit einem Endknoten in $\bigcup_{q' \in V(P_q)} l^2(q')$ bereits berücksichtigt sind. Wir definieren:

$$\begin{array}{rcl} l^1_{red}(q) & = & l^1(q) - \bigcup\limits_{q' \in V(P_q)} l^2(q') \\ l^2_{red}(q) & = & l^2(q) - \bigcup\limits_{q' \in V(P_q)} l^2(q') \end{array}$$

Nach der Definition von günstigen Separatorbäumen sind die Separatormengen $l^2(q)$ Separatoren im Graphen G, und somit gilt für alle Knoten $q, q' \in Q_i$ mit $q \neq q'$:

$$l^1_{red}(q) \cap l^1_{red}(q') = \emptyset \tag{7.15}$$

Dies ergibt für den Speicherplatz von H:

$$|E'| \leq \sum_{i=0}^{H(T)} \sum_{q \in Q_i} |l^1_{red}(q)||l^2_{red}(q)|$$

$$
\begin{aligned}
&\leq \sum_{i=0}^{H(T)} l_{max,i} \sum_{q \in Q_i} |l^1_{red}(q)| \\
&\underbrace{\leq}_{Gl.7.15} n \sum_{i=0}^{H(T)} W(T) \\
&\leq nW(T)(\log_{\frac{1}{\alpha}} n + 1)
\end{aligned}
$$

Die Laufzeit der Entfernungsberechnung für ein beliebiges Knotenpaar $u, v \in V$ folgt nach Lemma 8 in $W(T)$ Operationen durch Auswertung der folgenden Gleichung:

$$
\begin{aligned}
\delta(u,v) = \min\{&\delta'(\{u,x\}) + \delta'(\{x,v\}) \\
&\mid x \in l^2(nca(g(u), g(v)))\}
\end{aligned} \tag{7.16}
$$

■

Es stellen sich nun Fragen nach Größe und Berechnungsaufwand der Separatorweite von Graphen, die wir im Zusammenhang mit der Baumweite beantworten. Baumweite (engl. treewidth) ist ein Parameter, den Robertson und Seymour in ihren Arbeiten über Graphminoren eingeführt haben [RS86a]. Die Baumweite gibt - grob gesprochen - an, wie ähnlich ein Graph einem Baum ist. Eine neuere und relativ umfangreiche Untersuchung zu Arbeiten zum Thema Baumweite hat Hans L. Bodlaender in [Bod97] vorgenommen, in der 134 Literaturstellen referenziert werden.

Definition 18 (Baumweite) *Eine* **tree-decomposition** *von einem Graphen $G = (V, E)$ ist ein Paar $(\mathcal{X}, \mathcal{T})$ mit dem Baum $T = (I, F)$ und einer Familie $\mathcal{X} = \{X_i \subseteq V \mid i \in I\}$ von Teilmengen von V, für die es jeweils genau einen Knoten im Baum T gibt, d.h. es gibt eine bijektive Abbildung $h : I \longrightarrow \mathcal{X}, h(i) = X_i$. Weiterhin gilt:*

1. $\bigcup_{i \in I} X_i = V$,
2. *für jede Kante $\{u, v\} \in E$ gibt es mindestens ein $i \in I$ mit $u, v \in X_i$ und*
3. *für alle $i, j, k \in I$ gilt: falls j auf dem Weg von i nach k in T liegt, gilt $X_i \cap X_k \subseteq X_j$.*

$\max_{i \in I}(|X_i| - 1)$ *heißt die* **Weite der tree-decomposition** $(\{X_i \mid i \in I\}, (I, F))$. *Die minimale Weite über alle tree-decompositions des Graphen G ist die* **Baumweite** *(treewidth) von G (Schreibweise $B(G)$).*

Der wenig anschauliche Teil 3 der Definition kann auch durch folgenden äquivalenten Teil ersetzt werden: Jeder Teilgraph T_v von T (für $v \in V(G)$) ist zusammenhängend bzw. ein Teilbaum. Dabei besteht T_v aus allen Knoten $i \in V(T)$ mit $v \in h(i)$.

Wenn man anstatt eines Baumes nur einen Weg zulässt, spricht man von einer **path-decomposition**. Die **Pfadweite** (pathwidth) eines Graphen ist das Minimum über das Maximum der Weite von path-decompositions eines Graphen analog zur Baumweite (Schreibweise $P(G)$).
Die Baumweite eines Baums beträgt genau eins, wenn der Baum mehr als einen Knoten besitzt. Die Baumweite eines vollständigen Graphen K_{n+1} ist genau n, und die Baumweite eines Kreises C_n mit mindestens drei Knoten ist zwei.

Lemma 10 *[Bod93] Für jeden Graphen G gilt: $\omega(G) - 1 \leq B(G)$.*

Die Baumweite des in Abbildung 7.7 dargestellten Graphen ist drei, da der Graph den K_4 als Teilgraphen besitzt. In Abbildung 7.8 ist eine mögliche tree-decomposition zu G mit Weite drei angegeben.

Lemma 11 *Gegeben sei ein Graph $G = (V, E)$ mit tree-decomposition $(\mathcal{X}, \mathcal{T})$. Der Knoten $x \in V(T)$ teile den Baum T in die zusammenhängenden Teilbäume $T_1, \ldots, T_r$, und die Kante $(i, j) \in E(T)$ teilt T in die zusammenhängenden Teilbäume T'_i, T'_j. Es gilt dann:*

$$h(x) \quad \textit{separiert} \quad V(T_1), \ldots, V(T_r) \textit{ in } G \tag{7.17}$$

$$h(i) \cap h(j) \quad \textit{separiert} \quad V(T'_i), V(T'_j) \textit{ in } G \tag{7.18}$$

Beweis: (7.18) folgt aus (7.17) durch Einfügen eines Zwischenknotens x mit $h(x) = h(i) \cap h(j)$ in die tree-decomposition $(\mathcal{X}, \mathcal{T})$.

O.B.d.A. sei $w \in P^G(u, v)$ ein Weg zwischen $u \in V(T_1)$ und $v \in V(T_2)$ mit $u \neq v$ im Graphen G. Entweder u, oder v liegen in $h(x)$ oder es gibt zwei Kanten $\{u', v'\} \in V(T_1)$ und $\{v', v''\} \in V(T_2)$ auf dem Weg w. Dann muss nach Definition 18 $v' \in h(x)$ gelten.

■

Für $m \times k$-Gittergraphen G stimmen Pfad- und Baumweite genau überein. Die Pfadweite ist für einen $m \times k$-Gittergraphen genau k, wenn $k \leq m$ gilt. Wie die entsprechende path-decomposition aufgebaut wird, ist in Abbildung 7.6 am Beispiel eines 3×4-Gitters dargestellt.
Warum kann die Baumweite eines Gittergraphen nicht kleiner sein? Sei $(\mathcal{X}, T)$ eine tree-decomposition mit Weite $k - 1$ zum $m \times k$-Gittergraphen G. Nach Lemma 13 und Lemma 11 gibt es in T einen Knoten $u \in V(T)$, so dass $h(u)$ einen $\frac{1}{2}$-Separator in G bildet. Nach [GJ79] hat aber sogar jeder $\frac{2}{3}$-Separator in dem Gittergraphen mindestens die Größe k im Widerspruch zur Annahme. Somit gilt: $B(G) = P(G) = k$.
Die Bestimmung der Baumweite eines Graphen ist NP-vollständig [ACP87]. Für eine konstante Baumweite $k \in \mathbb{N}$, d.h. k ist nicht Teil der Eingabe, lässt sich aber in linearer Zeit feststellen, ob ein Graph höchstens die Baumweite k besitzt. Eine tree-decomposition der Weite k lässt sich in diesem Fall in linearer Zeit berechnen [Bod93].

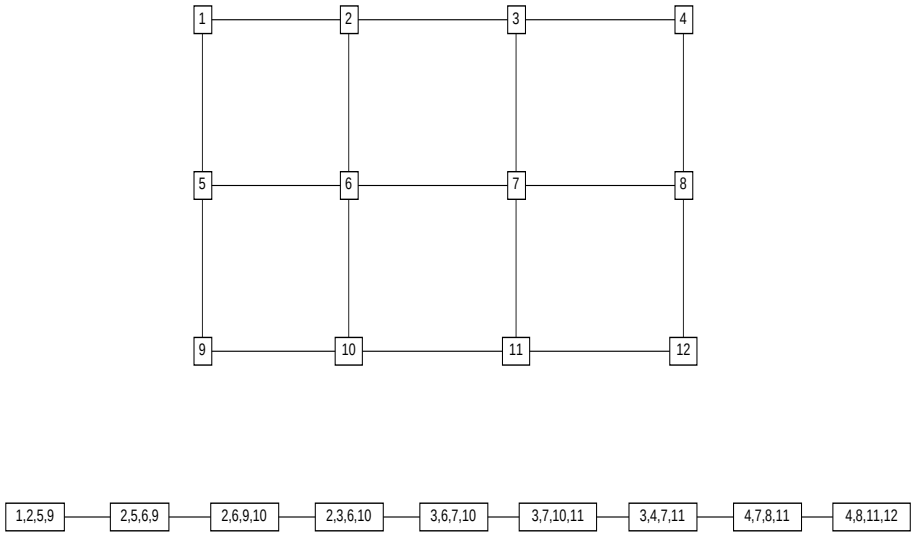

Abbildung 7.6: 3×4-Gittergraph mit path-decomposition mit Weite 3.

Die Untersuchungen zum Thema Baumweite sind aus verschiedenen Gründen von großem Interesse. Zum einen hat sich für viele NP-vollständige Probleme gezeigt, dass sie für Graphen mit konstanter Baumweite mit polynomiellem oder sogar mit linearem Aufwand lösbar sind. Als Beispiel seien hier nur die Probleme genannt, die sich mit monadischer Logik 2. Ordnung formulieren lassen [Cou90] [Cou89] [Cou92]. Dazu gehören beispielsweise: Partition Into Triangles, Vertex Cover und Independent Set [GJ79]. Für Graphen mit konstanter Baumweite lassen sich diese Probleme in linearer Zeit lösen [BPT91] [ALS91] [CM93a].
Zum anderen nimmt die Baumweite eine wichtige Rolle im Zusammenhang mit der von Robertson und Seymour entwickelten Minorentheorie ein. Wir geben hier verkürzt ein zentrales Ergebnis wieder. Sei $Forb(X)$ die Menge aller Graphen, die den Graphen X nicht als Minor enthalten. Die Baumweite der Graphen in $Forb(X)$ ist genau dann beschränkt, wenn der Graph X planar ist [RS86b].
Es gibt verschiedene Normalformen für tree-decompositions, die i.A. die Laufzeiten der Algorithmen nicht verbessern, aber das Design erleichtern (nice tree-decomposition [Bod97], smooth tree-decomposition [Bod93]).
Eine tree-decomposition $(\mathcal{X}, T)$ mit Weite $k \in \mathbb{N}$ heißt **smooth** genau dann, wenn für alle Knoten $v \in V(T)$ gilt: $|h(v)| = k + 1$ und für alle Kanten $\{u, v\} \in E(T)$ gilt: $|h(u) \cap h(v)| = k$. Die in Abbildung 7.6 dargestellte path-decomposition ist beispielsweise smooth.

Lemma 12 *[Bod93] Jede tree-decomposition zu einem Graphen $G = (V, E)$ kann in eine smooth tree-decomposition mit gleicher Weite transformiert werden. Jede smooth tree-decomposition mit Weite $k \in \mathbb{N}$ besitzt genau $|V(T)| = |V| - k$ Knoten.*

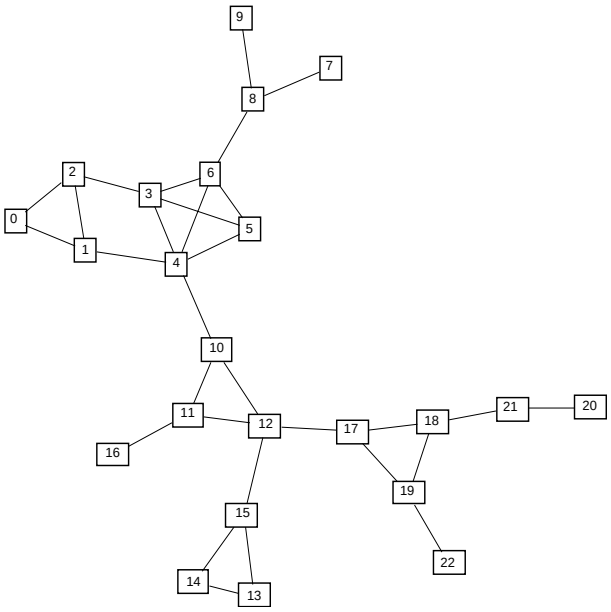

Abbildung 7.7: Beispielgraph G mit Baumweite $B(G) = 3$.

Eine wichtige Klasse von Graphen im Zusammenhang mit der Baumweite sind k-trees und partial k-trees, die erstmals 1969 von Beineke und Pippert definiert wurden [BP69].

Definition 19 *Ein* **k-tree** *wird rekursiv definiert. Eine Clique mit $k+1$ Knoten ist ein k-tree. Sei T_n ein k-tree mit n Knoten, dann wird ein k-tree mit $n+1$ Knoten durch T_n und einen zusätzlichen Knoten x aufgebaut, der mit einer k-Clique von T_n verbunden wird.*
Ein Teilgraph von einem k-tree heißt **partial k-tree**.

Man sieht leicht, dass ein k-tree die Baumweite k und ein partial k-tree höchstens die Baumweite k besitzen. Dazu ist beim Aufbau des k-trees eine entsprechende tree-decomposition aufzubauen. Ein k-tree ist k-zusammenhängend, und ein k-tree mit n Knoten besitzt genau $nk - \frac{k(k+1)}{2}$ Kanten.
Das folgende Lemma dient als Vorbereitung für das Theorem 11.

Lemma 13 *Gegeben sei ein Baum T mit n Knoten. In linearer Zeit kann ein Knoten $v \in V(T)$ gefunden werden, der T in zusammenhängende Teilbäume separiert, die höchstens die Größe $\frac{n}{2}$ besitzen.*

Beweis: Gewichte jeweils mit einem Tiefensuchendurchlauf Knoten-Kantenpaare mit der Anzahl der Knoten bis zu den Blättern. Bei einem zweiten Durchlauf wähle einen günstigen Knoten aus.

■

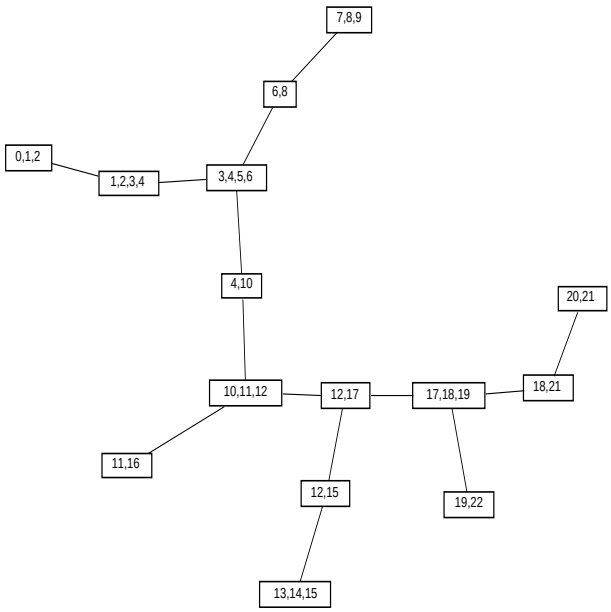

Abbildung 7.8: Tree-decomposition $(\mathcal{X}, T)$ mit Weite 3 zum Graphen aus Abbildung 7.7.

Theorem 11 *Für einen Graphen $G = (V, E)$ mit n Knoten gilt:*

$$\frac{B(G)}{\log_2 n} \quad \leq \quad S_{\frac{1}{2}}(G) \quad \leq \quad B(G)$$

Beweis: Zum Nachweis der zweiten Ungleichung transformieren wir eine smooth tree-decomposition $(\mathcal{X}, T)$ (siehe Lemma 12) von G mit minimaler Weite $k \in \mathbb{N}$ in einen binären günstigen $\frac{1}{2}$-Separatorbaum $T' = (Q, E_Q, l)$ mit Weite $W(T') = k$. h sei die bijektive Abbildung von $V(T)$ nach $\mathcal{X}$.

Algorithmus 14 *B2S*

Eingabe: *tree-decomposition $(\mathcal{X}, T)$ mit Weite k*

Ausgabe: *Günstiger $\frac{1}{2}$-Separatorbaum mit Weite k*

$T' := \emptyset$;
b2s(nil,T);
Funktion b2s (u: Knoten von T', t : Baum)
Füge einen neuen Knoten v und die Kante (u, v) in T' ein.
$l^1(v) := \bigcup_{x \in V(t)} h(x)$;
$r := |l^1(v)|$;
if $r \leq k$
then $l^2(v) := \bigcup_{x \in V(T)} h(x)$

else Sei i ein Knoten von t, der t in t_1, t_2 mit
$|V(t_1)|, |V(t_2)| \leq \frac{1}{2}r$ trennt.
$l^2(v) := h(i)$;
b2s(v, t_1); b2s(v, t_2);
endif ;
end b2s ;

Der Baum T' ist zu Beginn leer und wird durch den Funktionsaufruf b2s(nil,T) topdown von der Wurzel bis zu den Blättern aufgebaut. nil ist dabei der leere Knoten. Die lokale Variable r stimmt dabei immer genau mit der Anzahl der Knoten des Baums t überein. Der entstehende Baum T' ist offensichtlich binär.

Nach Lemma 11 bildet jeder Knoten in T eine Separatormenge, die aufgrund der Baumweite höchstens die Größe $k+1$ besitzt. Nach Lemma 13 existiert in jedem Baum ein Knoten, der ein $\frac{1}{2}$-Separator ist und in Linearzeit gefunden werden kann. Weil $(\mathcal{X}, T)$ smooth ist und nur $n-k$ Knoten enthält, besteht T' höchstens aus $n-k$ Knoten. Somit ist T' ein günstiger $\frac{1}{2}$-Separatorbaum zu G mit Weite $W(T) = k$. Für die Separatorweite von G folgt $S_{\frac{1}{2}}(G) \leq k$.

Sei nun ein günstiger $\frac{1}{2}$-Separatorbaum $T = (Q, E_Q, l)$ mit Weite $W(T) = k$ zu einem Graphen G gegeben. Wir konstruieren eine tree-decomposition $(\mathcal{X}, T' = (Q, E_Q))$ mit Weite

$$\begin{aligned} W(T') &\leq kH(T) \\ &\leq k(\log_2 n + 1). \end{aligned} \tag{7.19}$$

Die Knoten und Kanten des Baumes T werden für T' übernommen, und die Familie der Knotenmengen $\mathcal{X}$ und die zugehörige bijektive Abbildung h werden neu definiert. Für jeden Knoten $q \in V(T)$ sei P_q der eindeutige Weg von q zur Wurzel von T.

$$\mathcal{X} := \{ \bigcup_{r \in V(P_q)} h(r) \mid q \in V(T) \} \tag{7.20}$$

$$h(q) := \bigcup_{r \in V(P_q)} l^2(r) \tag{7.21}$$

Weil die betrachteten Wege höchstens die Länge $(\log_2 n + 1)$ besitzen, folgt die Gleichung 7.19.

Zum Nachweis, dass es sich um eine tree-decomposition handelt, überprüfen wir das Enthaltensein der Kanten und den Zusammenhang der Teilgraphen T_v. Nach Lemma 8 gibt es für jede Kante $\{x, y\} \in E(G)$ entweder einen Knoten $q \in Q$ mit $x, y \in l^2(q)$, dann ist nichts mehr zu zeigen, oder zwei verschiedene Knoten $q_1, q_2 \in Q$ mit $x \in l^2(q_1), y \in l^2(q_2)$ und

$$x \in h(nca(q_1, q_2)) \quad \text{oder} \quad y \in h(nca(q_1, q_2))$$

Aufgrund der Pfadkompression (Gleichung 7.21) folgt für den Baum T' und die Abbildung h:

$$\{x, y\} \subseteq h(q_1) \quad \text{oder} \quad \{x, y\} \subseteq h(q_2)$$

Der Teilgraph T_v von T für einen Knoten $v \in V(G)$ bestehend aus allen Knoten $q \in V(T)$ mit $v \in l^2(q)$ ist i.A. nicht zusammenhängend. Allerdings gilt für alle Knoten $p, q \in V(T_v)$:

$$nca(p, q) \quad \in \quad V(T_v) \tag{7.22}$$

Man beachte, dass sich die Abbildung $nca()$ in Gleichung 7.22 auf den Baum T bezieht. Aufgrund der Pfadkompression (Gleichung 7.21) sind die Teilgraphen T'_v von T' folglich für alle Knoten $v \in V(G)$ zusammenhängend.

■

Satz 18 *Es gibt Graphen G mit: $B(G) - S_\alpha(G) \in \Theta(\log_\alpha n)$, und es gibt Graphen, für die Baumweite und Separatorweite übereinstimmen.*

Beweis: Wir definieren rekursiv eine Graphfamilie von **S-Graphen** $G_{k,m}$, zu denen jeweils ein binärer günstiger $\frac{1}{2}$-Separatorbaum und ein ausgewählter Weg gehören. Eine k-Clique K_k ist ein S-Graph $G_{k,0}$ mit Separatorbaum $T = (\{r\}, \emptyset, l(r) = (V(K_k), V(K_k)))$ und ausgewähltem Pfad $[r]$ in T. Dabei besteht T nur aus dem Wurzelknoten r, und beide Komponenten der Abbildung l verweisen auf die gesamte Knotenmenge $V(K_k)$ des vollständigen Graphen. Seien nun $G^1_{k,i} = (V^1, E^1), G^2_{k,i} = (V^2, E^2)$ zwei S-Graphen mit verschiedenen Knoten- und Kantenmengen, mit Separatorbäumen T_1, T_2 und ausgewählten Pfaden P_1, P_2 jeweils von der Wurzel r_1, r_2 zu einem Blatt von T_1 bzw. T_2. Die Vereinigung $G^1_{k,i} \cup G^2_{k,i} \cup K_k$ mit zusätzlichen Kanten zwischen K_k und allen Knoten auf den Pfaden P_1 und P_2 bildet einen S-Graphen $G_{k,i+1}$, wobei die Knotenmenge $V(K_k)$ disjunkt zu denen der beiden S-Graphen ist. Der Separatorbaum zum S-Graphen $G_{k,i+1}$ besteht aus den beiden Separatorbäumen T_1, T_2 und einem zusätzlichen Knoten r, der die Wurzel von T bildet. Formal ergibt sich der S-Graph $G_{k,i+1}$ mit Separatorbaum T wie folgt:

$$\begin{aligned}
G_{k,i+1} &:= (V^1 \cup V^2 \cup V(K_k), E^1 \cup E^2 \cup E(K_k) \cup \\
&\quad \{\{a, b\} \mid \exists x \in V(P_1) \cup V(P_2) \text{ mit } a \in l^2(x) \text{ und } b \in V(K_k)\}) \\
T &:= (V(T_1) \cup V(T_2) \cup \{r\}, E(T_1) \cup E(T_2) \cup \{\{r, r_1\}, \{r, r_2\}\}, l) \\
l_{|V(T_i)} &:= l(T_i) \qquad \text{für } i = 1, 2 \\
l(r) &:= (V(G_{k,i+1}), V(K_k))
\end{aligned}$$

$l(T_i)$ bezeichnet hier die Labelfunktion l_i des Separatorbaums $T_i = (Q_i, E_{Q_i}, l_i)$. r bildet die Wurzel im Baum T und ist ein neuer Knoten. $P = [r] \circ P_1$ ist

der ausgewählte Pfad von dem S-Graphen $G_{k,i+1}$. Die Abbildung l wird von den beiden Separatorbäumen T_1 und T_2 übernommen und nur für r wie o.a. neu festgesetzt. In Abbildung 7.9 ist die Struktur eines Separatorbaums T von einem S-Graphen angegeben, und in Abbildung 7.10 sind die S-Graphen $G_{2,0}, G_{2,1}, G_{2,2}$ mit Separatorbäumen dargestellt.

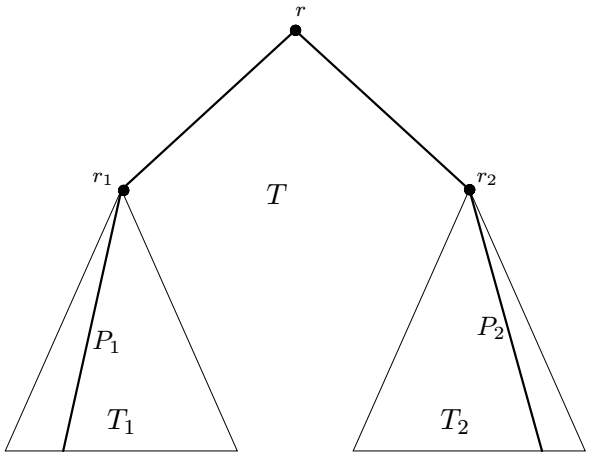

Abbildung 7.9: Separatorbaum T zu einem S-Graphen.

Es bleibt zu zeigen, dass T tatsächlich ein günstiger $\frac{1}{2}$-Separatorbaum für den Graphen $G_{k,i+1}$ ist. Die Separationseigenschaft folgt für Knoten $u \in E^1, v \in E^2$ sofort durch den Wurzelknoten von T, d.h. $l^2(r)$ separiert u, v in $G_{k,i+1}$. Wir definieren eine Abbildung $l^{-2} : V(G_{k,i+1}) \to V(T)$. Für alle Knoten $u \in V(G_{k,i+1})$ und alle Knoten $q \in V(T)$ sei:

$$l^{-2}(u) = q \iff u \in l^2(q) \tag{7.23}$$

Die Abbildung l^{-2} ist wohldefiniert, weil die Abbildung l^2 die Knoten von $G_{k,i+1}$ partitioniert.

Falls nun die beiden Knoten $u \in E^1, v \in E^2$ in einem Teilbaum, o.B.d.A. sei dies T_1 liegen, sei $u \notin l^2(nca(l^{-2}(u), l^{-2}(v)))$ und $v \notin l^2(nca(l^{-2}(u), l^{-2}(v)))$, andernfalls separiert $l^2(nca(l^{-2}(u), l^{-2}(v)))$ weiterhin die Knoten u, v in $G_{k,i+1}$. Somit liegen beide Knoten u, v nicht auf dem ausgewählten Pfad von der Wurzel zu einem Blatt in $G^1_{k,i}$, so dass sie in $G_{k,i+1}$ immer noch durch die Knoten $l^2(nca(l^{-2}(u), l^{-2}(v)))$ separiert werden.

Weil beim Übergang vom S-Graphen $G_{k,i}$ zum S-Graphen $G_{k,i+1}$ die Höhe des zugehörigen Separatorbaumes und somit die Tiefe jedes Knotens genau um eins steigt und die Anzahl der Knoten sich mindestens verdoppelt, ist T ein günstiger $\frac{1}{2}$-Separatorbaum.

Die Separatorweite des Graphen $G_{k,i+1}$ ergibt sich direkt durch die mitgeführten günstigen $\frac{1}{2}$-Separatorbäume zu $S_{\frac{1}{2}}(G_{k,i+1}) \leq W(T) = k - 1$.

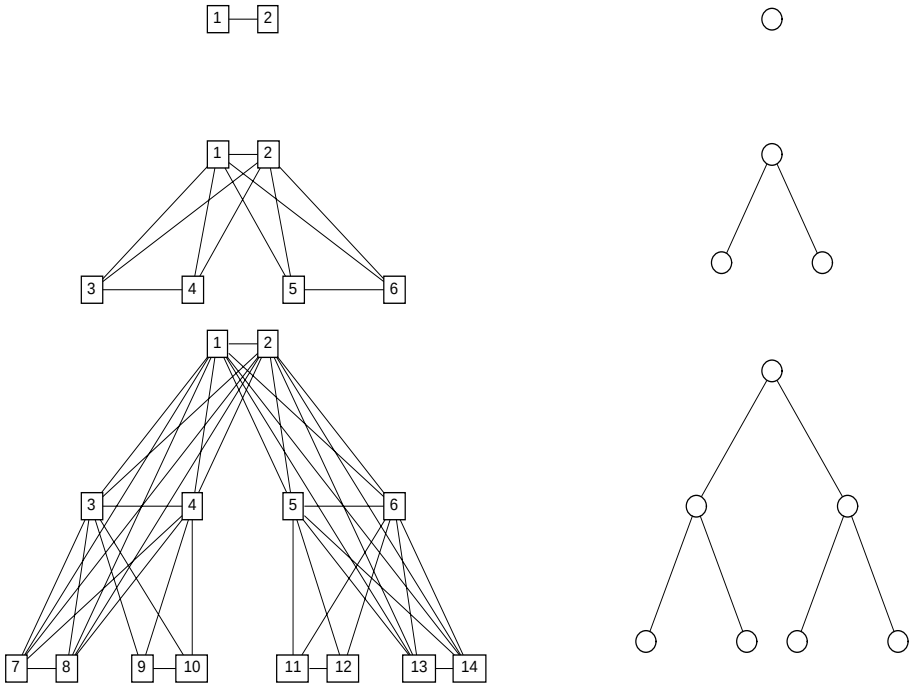

Abbildung 7.10: S-Graphen $G_{2,0}, G_{2,1}, G_{2,2}$ mit Separatorbäumen.

Für die Abschätzung der Baumweite betrachten wir den S-Graphen $G_{k,m}$ mit Separatorbaum T und ausgewähltem Weg $P = [v_0, \ldots, v_m]$ von einem Blatt v_m zur Wurzel v_0 in T ($n = |V(G_{k,m})| = (2^{m+1} - 1)k$). Die Knotenmenge $\bigcup_{i=0}^{m} l^2(v_i)$ ist eine Clique der Größe $k(m+1)$ in $G_{k,m}$. Nach Lemma 10 besitzt jede tree-decomposition von $G_{k,m}$ mindestens die Weite $\omega(G_{k,m}) \geq k(m+1) - 1$. Somit folgt für die Baumweite für jedes $k \geq 2$:

$$\begin{aligned} B(G_{k,m}) &\geq k(m+1) - 1 \\ &\geq m + 1 + \log_2 k \\ &= \log_2(2^{m+1}k) \\ &\geq \log_2(k(2^{m+1} - 1)) \\ &= \log_2 n \end{aligned}$$

Für den Nachweis, dass es Graphen G mit $S(G) \geq B(G) - 1$ gibt, betrachten wir k-trees. Die Baumweite von allen k-trees ist genau k, und k-trees sind k-zusammenhängend. Wir zeigen, dass es für jedes $0 \leq \alpha < 1$ einen k-tree G gibt, so dass $S_\alpha(G) \geq k$ ist. Wähle dazu die Anzahl von Knoten n so, dass gilt:

$$n - (k-1) \quad > \alpha n \tag{7.24}$$

Im Widerspruch zur Annahme sei T ein günstiger α-Separatorbaum für den k-tree G mit Weite $W(T) = k-1 = l_{max} - 2$. Angenommen, T sei ein Baum mit mindestens zwei verschiedenen Knoten $q_1, q_2 \in V(T)$ mit $nca(q_1, q_2) \neq q_i$ für $i = 1, 2$. Die Knotenmenge $l^2(nca(q_1, q_2))$ ist nach Voraussetzung ein α-Separator für die Knoten $l^1(q_1)$ und $l^1(q_2)$ in G. Aus dem k-Zusammenhang von G folgt:

$$\begin{aligned} S_\alpha(G) &\geq |l^2(nca(q_1, q_2))| - 1 \\ &\geq k - 1 \end{aligned}$$

Somit muss T die Struktur eines Pfades besitzen. Sei r die Wurzel von T, dann gilt:

$$\begin{aligned} |l^1(r)| &\geq n - |l^2(r)| \\ &\geq n - (k-1) \\ &> \alpha n \end{aligned}$$

Dies ergibt einen Widerspruch dazu, dass T ein α-Separatorbaum ist.

■

Theorem 12 *Zu Graphen mit konstanter Baumweite kann in einer $O(n \log n)$ Preprocessing-Phase auf $O(n \log n)$ Platz ein THG aufgebaut werden, mit dem die Entfernung in konstanter Zeit bestimmt werden kann.*

Beweis: Sei $G = (V, E, \delta)$ ein Graph mit Baumweite k. Bodlaender hat in [Bod93] ein Linearzeitverfahren zur Berechnung einer tree-decomposition $(\mathcal{X}, \mathcal{T})$ mit Weite k vorgestellt. Die konstanten Faktoren dieses Verfahrens sind allerdings von exponentieller Größenordnung in einem Polynom von der Größe k.

Wir benutzen den Algorithmus *B2S* aus dem Beweis zu Theorem 11 und konstruieren aus $(\mathcal{X}, \mathcal{T})$ in $O(n \log n)$ Operationen einen günstigen $\frac{1}{2}$-Separatorbaum mit Weite k. Da jede Separatormenge höchstens die Größe $k+1$ besitzt, benötigt der entsprechende THG weniger als $(k+1)n \log n \in O(n \log n)$ Speicherplatz. Die Entfernungsberechnung erfolgt nach Theorem 9 ebenfalls in dieser Größenordnung.

■

In Abbildung 7.11 ist ein günstiger $\frac{1}{2}$-Separatorbaum mit Weite 3 für den Graphen aus Abbildung 7.7 dargestellt.
Der Graph in Abbildung 7.1 besitzt die Baumweite drei, weil der Graph den Minor K_4 enthält und es eine tree-decomposition der Weite drei gibt (s. Abbildung 7.12). Der Graph besitzt die $\frac{1}{2}$-Separtorweite zwei. In Abbildung 7.13 ist ein günstiger $\frac{1}{2}$-Separtorbaum dargestellt.

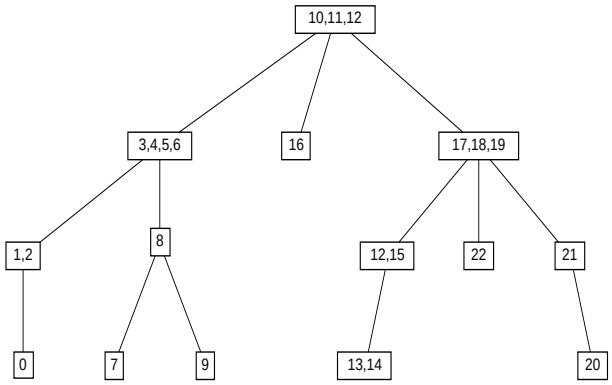

Abbildung 7.11: Separatorbaum zum Graphen aus Abbildung 7.7.

7.5 Chordale Graphen

Weil Trennende-Hierarchische-Graphen chordal sind (zumindest der flache THG), untersuchen wir in diesem Abschnitt chordale Graphen als Eingabegraphen und nutzen dabei Hyperbäume aus der Theorie der chordalen Graphen [Sim92] [Bra94]. Die Cliquengröße eines chordalen Graphen stellt sich als der entscheidende Parameter für die Laufzeit der Online-Entfernungsberechnung $\mathcal{Q}$ heraus. Gegeben sei ein THG mit Separatorbaum $T = (Q, E_Q, l)$ zum Graphen $G = (V, E, \delta)$. Der Graph $H = (V, E')$ mit

$$E' \quad = \quad \bigcup_{q \in Q} [l^1(q), l^2(q)]$$

heißt der **flache THG** zum Separatorbaum T.

Satz 19 *H ist chordal.*

Beweis: Angenommen, $w = [u_0, u_1, \ldots, u_{k-1}]$ sei ein Kreis der Länge $k \geq 4$ in H ohne Sehne. $q \in Q$ sei der nächste Knoten zur Wurzel von T mit $l^2(q) \cap V(w) \neq \emptyset$. Wegen $V(w) \subseteq l^1(q)$ gibt es einen Knoten von $V(w)$, der mit allen anderen Knoten von $V(w)$ durch eine Kante verbunden ist. Also besitzt w eine Sehne.

■

Die folgende Charakterisierung chordaler Graphen geben wir an, ohne sie hier zu beweisen.

Satz 20 *[Bra94] Ein Graph G ist genau dann chordal, wenn jeder minimale Separator eine Clique in G induziert.*

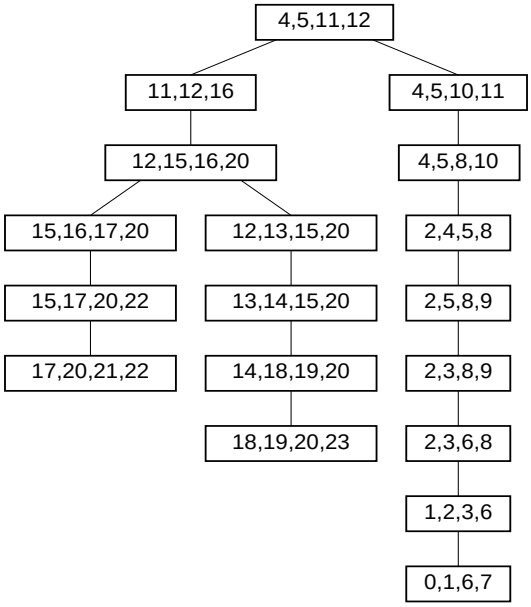

Abbildung 7.12: Tree-decomposition mit Weite drei zum Graphen aus Abbildung 7.1.

Nach Lemma 10 gilt i.A. $B(G) \geq \omega(G) - 1$. Angenommen, für einen chordalen Graphen G sei $B(G) > \omega(G) - 1$ und $(\mathcal{X}, T)$ sei eine tree-decomposition mit Weite $B(G)$. O.B.d.A. gibt es eine Knotenmenge $X \in \mathcal{X}$ mit $|X| - 1 = B(G)$, die ein minimaler Separator in G ist. Nach Satz 20 induziert X eine Clique in G, und es folgt:

$$\begin{aligned} \omega(G) &\geq |X| \\ &= B(G) + 1 \end{aligned}$$

Dies ergibt einen Widerspruch zur Annahme und für die Baumweite chordaler Graphen gilt:

$$B(G) = \omega(G) - 1 \tag{7.25}$$

Die Chordalitätseigenschaft kann für einen beliebigen Graphen mit n Knoten und m Kanten mit linearem Aufwand $O(n+m)$ nachgewiesen werden [RTL76].

Theorem 13 *Für chordale Graphen G mit Cliquengröße $\omega(G)$ gilt:*

$$\begin{aligned} \mathcal{S} &\in O(n\omega(G)\log n) \\ \mathcal{Q} &\in O(\omega(G)) \end{aligned}$$

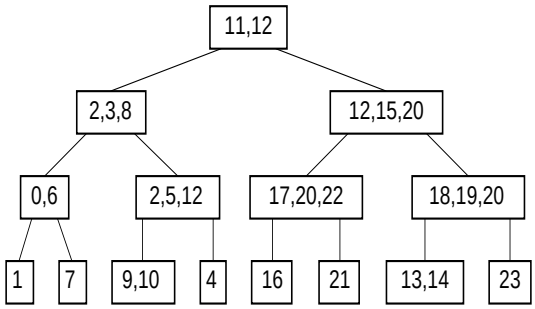

Abbildung 7.13: Günstiger $\frac{1}{2}$-Separatorbaum mit Weite zwei zum Graphen aus Abbildung 7.1.

Beweis: Wir führen hier die Begriffe Hypergraph und Hyperbaum ein, die im Zusammenhang mit chordalen Graphen eine wichtige Rolle spielen [Ber88] [Bra94]. Das Paar $H = (V, \mathcal{E})$ mit einer Knotenmenge V und einer Kantenmenge $\mathcal{E} \subseteq 2^V$ (Hyperkanten) heißt **Hypergraph**. Ungerichtete Graphen sind also ein Spezialfall von Hypergraphen, wobei alle Hyperkanten höchstens aus zwei Knoten bestehen. Zu einem Graphen G heißt der Hypergraph

$$\mathcal{C}(G) \quad = \quad (V(G), \{c \mid c \text{ ist maximale Clique in } G\})$$

der **Cliquengraph** von G. Ein Beispiel für einen Cliquengraphen ist

$$\mathcal{C}(G) \quad = \quad (\{0,1,2,3,4,5\}, \{\{0,1,3\}, \{0,2,3\}, \{1,3,4,5\}\}) \quad (7.26)$$

zu dem Graphen aus Abbildung 2.1. Zu einem Hypergraphen $H = (V, \mathcal{E})$ heißt ein Baum T mit Knotenmenge $\mathcal{E}$ **dualer Hyperbaum**, wenn für alle Knoten $v \in V$ des Hypergraphen H der induzierte Teilgraph $T[\{e \in \mathcal{E} \mid v \in e\}]$ in T zusammenhängend (also ein Teilbaum) ist. In Abbildung 7.14 ist der eindeutige duale Hyperbaum zum Cliquengraphen $\mathcal{C}(G)$ aus Gleichung 7.26 dargestellt. Duale Hyperbäume sind i. A. jedoch nicht eindeutig.

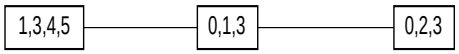

Abbildung 7.14: Dualer Hyperbaum zum Cliquengraphen $\mathcal{C}(G)$ (G aus Abbildung 2.1).

Das folgende Theorem stellt das zentrale Ergebnis aus der Theorie chordaler Graphen dar, das wir hier nicht beweisen.

Theorem 14 (Duchet/Flament) *G ist genau dann ein chordaler Graph, wenn es zu dem Cliquengraphen $\mathcal{C}(G)$ einen dualen Hyperbaum gibt.*

Die Erkennung von chordalen Graphen mit n Knoten und m Kanten und die Konstruktion eines zugehörigen dualen Hyperbaums kann in Linearzeit $O(n+m)$ erfolgen [TY84].

Sei nun also G ein chordaler Graph, $\mathcal{C}(G) = (V(G), C)$ ein Cliquengraph zu G und T ein dualer Hyperbaum zu $\mathcal{C}(G)$. Das Paar (C, T) bildet dann genau eine tree-decomposition, die mit Algorithmus *B2S* (siehe Theorem 11) in einen günstigen $\frac{1}{2}$-Separatorbaum mit Weite $\omega(G)$ transformiert wird. Daraus folgt die Behauptung.

■

Korollar 13 *Für dünne chordale Graphen gilt:*

$$\begin{aligned} \mathcal{S} &\in O(n^{1.5} \log n) \\ \mathcal{Q} &\in O(\sqrt{n}) \end{aligned}$$

Beweis: Für dünne chordale Graphen G gilt: $\omega(G) \in O(\sqrt{n})$. Nach Theorem 13 folgt die Aussage.

■

7.6 Triangulationen

Jeder Graph kann durch Hinzufügen geeigneter Kanten in einen chordalen Graphen transformiert werden. Zu einem gegebenen Graphen $G = (V, E)$ heißt der Graph $H = (V, E \cup E')$ **Triangulation**, wenn H chordal ist. Die Kanten E' werden auch als **Fill-in** bezeichnet.
Für einen Graphen G definieren wir:

$$\omega^*(G) = \min\{\omega(H) \mid H \text{ ist eine Triangulation von } G\}$$

Es stellen sich hier zwei Fragen nach der Berechnung von 'günstigen' Triangulationen, die i.A. beide NP-vollständig sind [ACP87] [Yan81]:

1. Minimum-Fill-in: Berechnung von minimalen Triangulationen in der Größe des Fill-in ($|E'|$).

2. Baumweite: Berechnung von Triangulationen mit minimaler maximaler Cliquengröße ($\omega^*(G)$).

Warum es sich bei Fragestellung 2 genau um die Bestimmung der Baumweite handelt, klärt das folgende Lemma.

Lemma 14 *Für alle Graphen $G = (V, E)$ gilt:*

$$B(G) \quad = \quad \omega^*(G) - 1$$

Beweis: Wir zeigen für einen Graphen $G = (V, E)$: $B(G) \geq \omega^*(G) - 1$ und $B(G) \leq \omega^*(G) - 1$.

$\geq$ Sei $(\mathcal{X}, T)$ eine tree-decomposition mit Weite $B(G)$. Erweitere die Kantenmenge E wie folgt, ohne die Baumweite zu vergrößern.

$$E' \quad = \quad \bigcup_{x \in V(T)} \{\{u, v\} \subset V \mid u \in h(x) \wedge v \in h(x)\}$$

Die Kantenmenge E' ist eine Obermenge von E. Der Graph $H = (V, E')$ ist offensichtlich chordal, und es gilt:

$$\begin{aligned} B(G) &= B(H) \\ &= \omega^*(H) - 1 \\ &\geq \omega^*(G) - 1 \end{aligned}$$

$\leq$ Sei H eine Triangulation von G mit $\omega(H) = \omega^*(G)$. Dann gilt:

$$\begin{aligned} B(G) \quad &\leq \quad B(H) \\ &\underbrace{=}_{Gleichung 7.25} \quad \omega(H) - 1 \\ &= \quad \omega^*(G) - 1 \end{aligned}$$

■

Wir werden wie folgt einen Separatorbaum T zu einem Graphen G bestimmen.

1. Berechnung einer Triangulation H zu G mit kleiner maximaler Cliquengröße
2. Berechnung eines dualen Hyperbaums T' zum Cliquengraphen $\mathcal{C}(H)$
3. Transformation von T' in einen Separatorbaum T mit Algorithmus *B2S*

Zur Berechnung einer Triangulation bzw. eines Fill-in eines Graphen haben Tarjan und Yannakakis 1984 in [TY84] Algorithmen vorgestellt, die eine Abwandlung des *Chordalitätstests RTL* von Rose, Tarjan und Lueker sind [RTL76].
Zur Einführung dieser Algorithmen benötigen wir noch den Begriff der Ordnung.

Definition 20 ([Bra94]) *Es sei $G = (V, E)$ ein Graph. Eine Knotenreihenfolge $\sigma = (v_1, \ldots, v_n)$ von V heißt* **Ordnung** *von V mit $\sigma(v_i) = i$.*

$F(\sigma) \quad = \quad \{\{u, v\} \notin E \mid \exists$ *ein Weg* $w \in P^G(u, v)$*, der nur innere Knoten x mit $\sigma(x) \leq \sigma(u) \wedge \sigma(x) \leq \sigma(v)$ enthält*$\}$

heißt **Fill-in von σ von** G*. Eine Ordnung σ heißt* **perfekt** *genau dann, wenn $F(\sigma) = \emptyset$ gilt.*

Lemma 15 ([Gol92]) *Für einen Graphen existiert genau dann eine perfekte Ordnung, wenn der Graph chordal ist.*

Tarjan und Yannakakis berechnen eine sogenannte 'Maximum Cardinality - Search'-Ordnung (MCS), bei der die Knoten in absteigender Reihenfolge von n bis 1 nummeriert werden [TY84]. Sie zeigen, dass eine MCS-Ordnung genau dann einen leeren Fill-in liefert, wenn der gegebene Graph chordal ist, und nutzen diese Eigenschaft zur Erkennung chordaler Graphen. Außerdem geben sie Linearzeit-Algorithmen zur Berechnung einer MCS-Ordnung σ und des Fill-in $F(\sigma)$ an, die wir hier verkürzt wiedergeben.
MCS: Wähle einen beliebigen Knoten $u \in V$, und nummeriere ihn mit $\sigma(u) = n$. Angenommen, die Knoten $X \subseteq V$ seien bereits nummeriert, dann bekommt der Knoten $u \in V - X$ mit den meisten Nachbarn in X (nichtdeterministische Auswahl bei mehreren Möglichkeiten)

$$|N(u) \cap X| \quad = \quad \max\{|N(v) \cap X| \mid v \in V - X\}$$

die nächste Nummer $\sigma(u) = |V - X|$.
Fill-in: Die Ordnung σ wird in aufsteigender Reihenfolge durchlaufen. Durch Einfügen geeigneter Kanten wird dafür gesorgt, dass nach Bearbeitung des Knotens $\sigma^{-1}(i)$ der durch die ersten i Knoten der Ordnung σ induzierte Teilgraph mit berechnetem Fill-in chordal ist.
Beide Algorithmen arbeiten in Linearzeit, d.h. MCS in $O(n + m)$ Operationen und die Berechnung des Fill-in F in $O(n+m')$ Operationen, wobei $m' = |E \cup F|$ ist. Für MCS verwendet man ein Feld von Mengen $set(i)$ für $0 \leq i \leq n-1$, wobei die Menge $set(i)$ genau die Knoten enthält, die zu i bereits nummerierten Knoten benachbart sind. Anfangs sind alle Knoten in $set(0)$ enthalten. In jedem Schritt wird das maximale i mit $set(i) \neq \emptyset$ gewählt, und ein beliebiger Knoten aus $set(i)$ erhält die nächste Nummer. Die noch nicht nummerierten Nachbarn des gewählten Knotens steigen in eine um eins größere set-Menge auf. Das neue maximale i mit $set(i) \neq \emptyset$ könnte sich um eins erhöht haben, d.h. wenn im letzten Schritt j gewählt wurde, erfolgt nun die Suche nach einer nichtleeren set-Menge absteigend beginnend mit $j + 1$. Die Knoten des Graphen und die set-Mengen werden doppelt verkettet. Die Berechnung des Fill-in in Linearzeit erfolgt durch eine ähnliche Datenstruktur.
Die Algorithmen erzeugen i.A. keinen minimalen Fill-in (aufgrund der NP-Vollständigkeit ist das auch nicht zu erwarten) und unter Umständen sogar zu einem dünnen Graphen einen dichten Fill-in, wie das folgende Beispiel demonstriert.
Wir definieren eine 3-reguläre Familie von Graphen $G_n = (V_n, E_n)$ für $n \geq 4$, und n sei gerade.

$$\begin{aligned} V_n &= \{1, 2, \ldots, n\} \\ E_n &= \{\{1,2\}, \{1,n\}, \{1, \frac{n}{2}+1\}\} \cup \{\{i, i+1\} \mid 2 \leq i < n\} \\ &\quad \cup \{\{\frac{n}{2}+1-i, \frac{n}{2}+1+i\} \mid 1 \leq i < \frac{n}{2}\} \\ |E_n| &= n + \frac{n}{2} \end{aligned}$$

Die Knotennummerierung ist eine mögliche Nummerierung σ, die durch MCS berechnet werden kann. Der Fill-in-Algorithmus berechnet:

$$\begin{aligned} F(\sigma) &= \{\{\frac{n}{2}+1, i\} \mid 2 \leq i < \frac{n}{2}\} \cup \\ &\quad \{\{\frac{n}{2}+2+j, \frac{n}{2}+1+i\} \mid 1 \leq j < \frac{n}{2}-1, -j \leq i < j\} \\ |F(\sigma)| &= \frac{n^2}{4} - n \end{aligned}$$

In Abbildung 7.15 ist der Graph G_{10} ohne und mit dem berechneten Fill-in (15 Kanten) dargestellt.

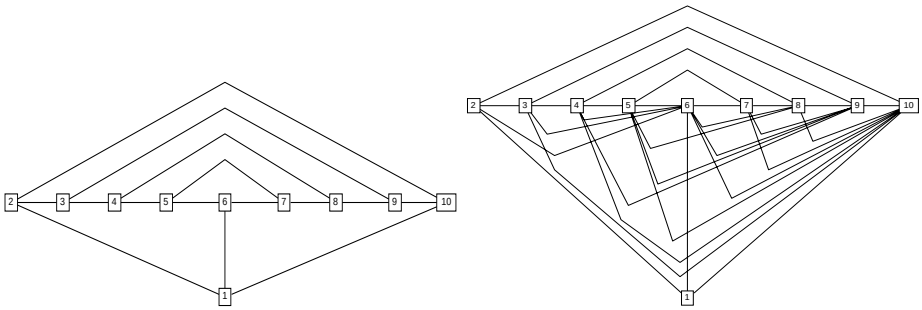

Abbildung 7.15: Der Graph G_{10} ohne und mit Fill-in.

Wenn man die Algorithmen MCS und den Fill-in Algorithmus immer abwechselnd aufruft, d.h. nach jedem Einfügen einer Kante im Fill-in wird mit MCS erneut eine perfekte Ordnung berechnet, so erhält man für den Graphen G_n nur einen Fill-in der Größe $n + \frac{n}{2} - 6$. In Abbildung 7.16 ist der Graph G_{10} mit dem so entstandenen Fill-in (9 Kanten) dargestellt.

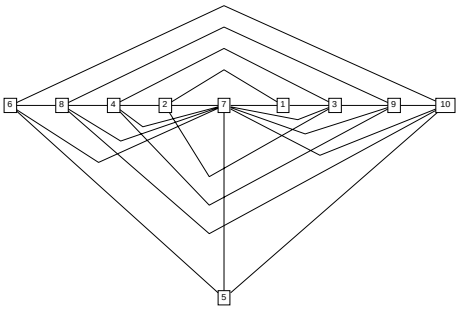

Abbildung 7.16: Der Graph G_{10} mit günstigerem Fill-in.

Es stellen sich hier für künftige Untersuchungen insbesondere für dünne Graphen die Fragen, ob der Minimum-Fill-in höchstens linear ist und ob die Baum-

weite von der Größenordnung $O(\sqrt{n})$ ist. Sofern man den ersten Teil positiv beantworten kann, folgt daraus, dass auch die Baumweite höchstens von der Größenordnung $O(\sqrt{n})$ ist. Dies ergibt sich daraus, dass nach Lemma 7.25 für jeden chordalen Graphen G gilt:

$$|E(G)| \quad \geq \quad (B(G))^2$$

7.7 Dünne-Trennende-Hierarchische-Graphen

In diesem Abschnitt betrachten wir nur planare Graphen und verwenden günstige Separatorbäume, um den Platz für die hierarchischen Graphen auf $O(n \log n)$ zu verringern. Der Aufwand zur Berechnung von SPSD erhöht sich dadurch um den Faktor $O(\log n)$.
Die verwendeten Separatoren bestehen entweder nur aus einem Knoten oder sind einfache Kreise in dem planaren Graphen. Die Separatoren bezeichnen wir daher als Kreisseparatoren.

Satz 21 ([Mil86]) *In einem planaren Graphen G gibt es entweder einen $\frac{2}{3}$-Separatorknoten oder einen einfachen Kreis, der ein $\frac{2}{3}$-$\sqrt{8n}$-Separator in G ist. Der Separator kann in linearer Zeit berechnet werden.*

In einem Baum gibt es beispielsweise keinen einfachen Kreis, der einen Separator bildet, jedoch gibt es mindestens einen $\frac{2}{3}$-Separatorknoten. Miller zeigt sogar, dass es für alle 2-zusammenhängenden planaren Graphen G einfache Kreise gibt, die $\frac{2}{3}$-$\sqrt{8n}$-Separatoren in G sind. Dazu ist es unter Umständen notwendig, die planare Einbettung des Graphen zu ändern.
Gegeben sei der Kreisseparator $S = \{x_1, \ldots, x_k\}$, den wir auch als geordnetes Tupel $S = (x_1, \ldots, x_k)$ schreiben, wobei die Ordnung durch die entsprechende planare Einbettung des zugrunde liegenden Graphen gegeben ist. Die Funktion $Clock : S \times S \to 2^S$ gibt die zwischen zwei Knoten liegenden Knoten an.

$$Clock(x_i, x_j) \quad = \quad \begin{cases} \{x_l \in S \mid i \leq l \leq j\} & \text{falls} \quad i \leq j \\ \{x_l \in S \mid i \leq l \leq k \vee 1 \leq l \leq j\} & \text{sonst} \end{cases}$$

Einen Separatorbaum, dessen Separatoren alle Kreise sind, bezeichnen wir als **Kreis-Separatorbaum**. Im planaren Fall folgt offensichtlich, wenn T ein Kreis-Separatorbaum ist, so ist T auch ein günstiger Separatorbaum.
Die Berechnung von Kreis-Separatorbäumen für planare Graphen kann ebenso wie für herkömmliche Separatorbäume in $O(n \log n)$ Operationen durchgeführt werden. Dazu berechnet man für jeden Knoten des Separatorbaums in linearer Zeit eine Separatormenge, die einen Kreis in dem Graphen bildet (s. auch simple-cycle-separator in [Lin90]). Djidjev und Venkatesan haben in [DV97] gezeigt, wie dieser Separator auch für die Parameter $\alpha = \frac{2}{3}$ und $\beta = 1.97$ in linearer Zeit berechnet werden kann.

In Günstigen-Trennenden-Hierarchischen-Graphen hatten wir für jeden Knoten $q \in V(T)$ eines Separatorbaums T die Kanten $[l^1(q), l^2(q)]$ eingefügt. Nun werden wir nur noch die Kanten zwischen den Knoten $l^2(q)$ und allen Knoten der Separatormengen auf einem absteigenden Pfad von q zu einem Blatt berücksichtigen, so dass der Speicheraufwand $O(n \log n)$ gewährleistet wird. Die Auswahl dieses Pfades und die Definition eines Dünnen-Trennenden-Hierarchischen-Graphen gehen auf Ansätze aus einer Arbeit von Lingas von 1990 zurück [Lin90].

Definition 21 (trennender Pfad) *Gegeben sei ein Graph mit Kreis-Separatorbaum $T = (Q, E_Q, l)$ mit Wurzel $q_0 \in Q$. Ein absteigender Weg $P = [q_0, q_1, \ldots, q_k]$ in dem günstigen Separatorbaum T heißt* **trennender Pfad**, *wenn für $1 \leq j < k$ gilt: $l^2(q_j)$ trennt $l^2(q_0)$ und $l^2(q_{j+1})$ nicht (s. Abbildung 7.17).*

Für einen Kreis-Separatorbaum mit Wurzel $q_0 \in Q$ gibt es zwei maximale trennende Pfade von q_0 über den rechten Sohn und von q_0 über den linken Sohn bis zu den Blättern. Weil jeder Teilbaum eines Kreis-Separatorbaums wiederum ein Kreis-Separatorbaum ist, gibt es für jeden Knoten $q \in Q$ zwei maximale trennende Pfade bis zu einem Blatt. Aus Gründen der einfacheren Schreibweise bezeichnen wir die Knotenmenge $l^2(q_i)$ mit S_i, wenn aus dem Zusammenhang der Knoten q_i hervorgeht.

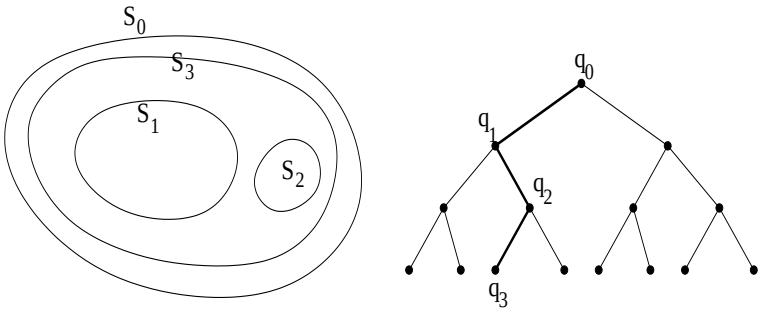

Abbildung 7.17: Ein trennender Pfad.

Das folgende Lemma wurde in einer ähnlichen Form 1990 von Lingas gezeigt und führt uns direkt zu dem Dünnen-Trennenden-Hierarchischen-Graphen.

Lemma 16 *[Lin90] Für einen Graphen G mit einem Kreis-Separatorbaum $T = (Q, E_Q, l)$ und einem trennenden Pfad $P = [q_0, q_1, \ldots, q_k]$ von der Wurzel bis zu einem Blatt gilt, dass $\bigcup_{j=1}^{k} S_j$ ein Separator von S_0 und $l^1(q_1)$ in G ist.*

Definition 22 (Dünner-Trennender-Hierarchischer-Graph DTHG) *Gegeben sei ein Graph $G = (V, E, \delta)$ mit einem günstigen Separatorbaum $T = (Q, E_Q, l)$ mit Wurzel $q_0 \in Q$. Die Söhne q_l, q_r von q_0 sind die Wurzeln des linken und rechten Teilbaums T_l, T_r von T. Der* **Dünne-Trennende-Hierarchische-Graph** *G^T wird rekursiv definiert: $G^T = G^{T_l} \cup G^{T_r} \cup LR$. Dabei*

sei G^{T_d} der Dünne-Trennende-Hierarchische-Graph, der durch den Graphen $G[l^1(q_d)]$ mit günstigem Separatorbaum T_d entsteht (für $d \in \{l, r\}$). Die eindeutigen trennenden Pfade von der Wurzel q_0 zu den Blättern seien: $P = [q_0, q_1, \ldots, q_k]$ und $Q = [q_0, q'_1, \ldots, q'_m]$ mit Separatormengen $S_0, S_1, \ldots, S_k$ und $S_0, S'_1 \ldots, S'_m$. Daraus ergibt sich folgender Graph:

$$LR \quad = \quad (S_0 \cup \bigcup_{i=1}^{k} S_i \cup \bigcup_{i=1}^{m} S'_i, E_{LR}, \delta_{LR})$$

Ein Knotenpaar $u, v \in S_0 \cup \bigcup_{i=1}^{k} S_i \cup \bigcup_{i=1}^{m} S'_i$ gehört genau dann in die Kantenmenge E_{LR}, wenn mindestens einer der beiden Knoten u, v in S_0 liegt. Für alle Kanten $\{u, v\} \in E_{LR}$ gilt weiterhin:

$$\delta_{LR}(\{u, v\}) \quad = \quad \delta(u, v)$$

Falls der zugrunde liegende Graph G planar ist, kann der Graph G^T auf $O(n \log n)$ Platz aufgebaut werden. Dazu ist der Kreis-Separatorbaum T in $O(n \log n)$ Operationen auf $O(n)$ Platz zu konstruieren. Nach Definition 22 gilt für die Knotenmenge V_{LR} des Graphen LR: $|V_{LR}| \in O(\sqrt{n})$ und somit für die Kantenmenge E_{LR}: $|E_{LR}| \in O(n)$. Die Größe eines Dünnen-Trennenden-Hierarchischen-Graphen mit n Knoten sei $\Gamma(n)$. Mit der Rekursionsgleichung:

$$\Gamma(n) \quad = \quad \Gamma(\alpha n) + \Gamma((1 - \alpha)n) + O(n) \tag{7.27}$$

folgt für $0 < \alpha < 1$ mit der Anfangsbedingung $\Gamma(1) = 1$ [GK81]:

$$\Gamma(n) \quad \in \quad O(n \log n)$$

Die Berechnung des Graphen G^T ohne Berücksichtigung der Kantengewichte kann in $O(n \log n)$ Operationen erfolgen. Allerdings bezieht sich die Kantengewichtsfunktion im DTHG auf den gegebenen Graphen G und nicht, wie es beim THG der Fall ist, auf einen in G induzierten Graphen. Für planare Graphen werden alle Kantengewichte eines DTHG durch Anwendung des Verfahrens von Klein u.a. [KRRS94] in $O(n^2)$ berechnet.
Es ist noch zu zeigen, dass die Graphen G und G^T längengleich sind ($G \equiv G^T$). Zum Nachweis führen wir eine Induktion über die Höhe des Separatorbaums durch. Für einen Separatorbaum T mit Höhe Null besteht die Separatormenge der Wurzel aus allen Knoten. G^T ist dann ein vollständiger Graph, der durch die Berechnung aller Entfernungen von G entsteht. Somit sind G und G^T längengleich.
Angenommen, der Separatorbaum T habe die Wurzel q_0 mit den Söhnen q_1, q_2 mit $l(q_0) = (V, S_0), l(q_1) = (V_1, S_1), l(q_2) = (V_2, S_2)$. Nach Induktionsannahme gilt:

$$\begin{aligned} G^{T_1} &\equiv G[V_1] \\ G^{T_2} &\equiv G[V_2] \end{aligned}$$

Es bleibt zu zeigen, dass für alle Knoten $u \in V_1$ und $v \in V_2$ gilt:

$$\delta_T(u,v) \quad = \quad \delta(u,v)$$

Nach Lemma 8 führt jeder Weg $w' \in P(u,v)$ über S_0. Sei $r \in S_0$ ein Knoten auf einem kürzesten Weg $w \in SP(u,v)$ und $P_1 = [q_0, q_1, \ldots], P_2 = [q_0, q_2, \ldots]$ maximal lange trennende Pfade bis zu den Blättern. Dann gibt es nach Lemma 16 Knoten $x \in \bigcup_{q \in V(Suf(P_1,|P_1|-1))} l^2(q)$ und $y \in \bigcup_{q \in V(Suf(P_2,|P_2|-1))} l^2(q)$ in den Separatormengen von P_1 und P_2 ohne jeweils den Knoten q_0, die auf dem Pfad w liegen, so dass gilt:

$$\begin{aligned} \{x,r\},\{y,r\} &\in E_T \\ \delta_T(\{x,r\}) &= \delta(x,r) \\ \delta_T(\{r,y\}) &= \delta(r,y) \end{aligned}$$

Da der Präfix bis zum Knoten r und der Suffix ab dem Knoten r des Weges $w = [u, \ldots, x, \ldots, r, \ldots, y, \ldots, v]$ vollständig in V_1 bzw. V_2 liegen, ergibt sich nach Induktionsannahme die Länge von w durch

$$\delta(w) \quad = \quad \delta_{T_1}(u,x) + \delta_T(\{x,r\}) + \delta_T(\{r,y\}) + \delta_{T_2}(y,v)$$

Damit ist auch die Längengleichheit von G und G^T gezeigt.
Wir wenden uns nun der Online-Entfernungsberechnung mit einem DTHG zu. Bei der Online-Entfernungsberechnung mit THG und günstigen Separatorbäumen haben wir nur eine Separatormenge betrachtet. Nun werden wir abhängig vom Verlauf der trennenden Pfade bestimmte Separatoren $S_1, S_2, \ldots, S_k$ auswählen, die auf dem eindeutigen Weg zwischen zwei Separatoren des günstigen Kreisseparatorbaumes liegen, die Start- und Zielknoten u, v enthalten. Wir berechnen dann sukzessive für $i = 1, \ldots, k$ die Entfernungen zwischen u und S_i und benutzen dabei die Kenntnis der Entfernungen zwischen u und S_{i-1}. Weil wir nicht die kürzesten Entfernungen berechnen, definieren wir eine spezielle Entfernungsfunktion, die die kürzeste Entfernung von einem Knoten eines Separators zu einem anderen Knoten angibt, wobei nur Wege betrachtet werden, die nicht den Separator kreuzen.

Definition 23 *Gegeben seien ein Graph $G = (V, E, \delta)$, eine Knotenmenge $C \subseteq V$ und die Knoten $u, v \in V$.*

$$\delta(u,v,C) \quad = \quad \min\{\delta(w) \mid w \in P^G(u,v), V(w) \cap C - \{v\} = \emptyset\}$$

heißt **kürzeste C-freie-Entfernung**, *und ein Weg $w \in P^G(u,v)$ mit $V(w) \cap C - \{v\} = \emptyset$ und $\delta(w) = \delta(u,v,C)$ heißt* **kürzester C-freier-Weg** *von u nach v. Eine Funktion δ' heißt eine* **untere C-Schranke**, *wenn für alle Knoten $u, v \in V$ gilt:*

$$\delta(u,v) \quad \leq \quad \delta'(u,v,C) \quad \leq \quad \delta(u,v,C)$$

Für jedes Knotenpaar $u, v \in V$ und jede Knotenmenge $C \in V$ gilt:

$$\begin{aligned} \delta(u,v) &\leq \delta(u,v,C) \\ \delta(u,v) &= \delta(u,v,\{v\}) \end{aligned}$$

Die Eigenschaft, kürzester C-freier-Weg zu sein, überträgt sich wie die Eigenschaft, kürzester Weg zu sein, auch auf die Teilwege.

Lemma 17 *Gegeben seien ein Graph $G = (V, E, \delta)$, eine Knotenmenge $C \subseteq V$ und ein kürzester C-freier-Weg w. Für jeden Teilweg w' von w gilt: w' ist kürzester C-freier-Weg.*

Lemma 18 *Gegeben seien ein planarer Graph G, knotendisjunkte Kreisseparatoren $X, Y \subseteq V(G)$ und ein Knoten $u \in V(G)$. Y separiere den Knoten u und den Kreisseparator X in G. Bekannt seien die Entfernungen $\delta(x,y)$ für alle $x \in X$ und alle $y \in Y$ und eine untere Y-Schranke $\delta^u(u,y,Y)$ für alle Knoten $y \in Y$. Eine untere X-Schranke $\delta^u(u,x,X)$ lässt sich für alle Knoten $x \in X$ in $O(|X| + |Y| \log |X|)$ Operationen berechnen.*

Beweis: Wir führen den Beweis konstruktiv. Seien $X = (x_1, \ldots, x_k)$ und $Y = (y_1, \ldots, y_l)$ knotendisjunkte geordnete Kreisseparatoren, und Y trennt u und X. Berechne in $O(|Y|)$ Operationen eine untere X-Schranke für die Knotenpaare u, x_1 und u, x_k.

$$\begin{aligned} \tilde{\delta}(u,x_1,X) &= \min\{\delta^u(u,y,Y) + \delta(y,x_1) \mid y \in Y\} \\ \tilde{\delta}(u,x_k,X) &= \min\{\delta^u(u,y,Y) + \delta(y,x_k) \mid y \in Y\} \end{aligned}$$

O.B.d.A. seien $\tilde{\delta}(u,x_1,X) = \delta(u,y_1,Y) + \delta(y_1,x_1)$ und $\tilde{\delta}(u,x_k,X) = \delta(u,y_l,Y) + \delta(y_l,x_k)$.

Die Funktion Mitte(x,x',X) bestimmt zu den Knoten $x, x' \in X$ des Kreisseparators X einen mittleren Knoten im Uhrzeigersinn, d.h. es gilt:

$$||Clock(x, Mitte(x,x',X))| - |Clock(Mitte(x,x',X),x')|| \leq 1$$

Algorithmus 15 *KW-Kreisseparator*

Eingabe: *x, x': Knoten vom Kreisseparator X; y, y': Knoten vom Kreisseparator Y*

Ausgabe: *$\tilde{\delta}(u,x,X)$ für jeden Knoten $x \in X - \{x_1, x_k\}$*

$x'' :=$ Mitte(x,x',X);
if $x'' \neq x$ **and** $x'' \neq x'$ **then**
  bestimme $y'' \in Clock(y,y')$ mit
  $\delta(x'',y'') + \delta^u(u,y'',Y) = \min\{\delta(x'',v) + \delta^u(u,v,Y) \mid v \in Clock(y,y')\}$

$\tilde{\delta}(u, x'', X) := \delta(x'', y'') + \delta^u(u, y'', Y)$;
KW-Kreisseparator(x, x'', y, y'');
KW-Kreisseparator(x'', x', y'', y');
endif ;

Zum Nachweis, dass $\tilde{\delta}$ eine untere X-Schranke ist, führen wir eine Induktion über die Anzahl der rekursiven Aufrufe von *KW-Kreisseparator* durch. Bis zu dem rekursiven Aufruf von *KW-Kreisseparator*$(x_i, x_j, y_{i'}, y_{j'})$ $(1 \leq i, j \leq k, 1 \leq i', j' \leq l)$ sei $\tilde{\delta}$ eine untere X-Schranke, und es sei $x =$Mitte(x_i, x_j). Es ist zu zeigen, dass gilt:

$$\min\{\delta^u(u, v, Y) + \delta(v, x) \mid v \in Clock(y_{i'}, y_{j'})\} \quad \leq \quad \delta(u, x, X) \quad (7.28)$$

Sei $w_1 \in P(u, x_i)$ ein Weg, auf dem der Knoten $y_{i'}$ liegt und dessen Länge bereits berechnet wurde:

$$\tilde{\delta}(u, x_i, X) \quad = \quad \delta(w_1)$$

Analog dazu sei $w_2 \in P(u, x_j)$ der Weg mit Knoten $y_{j'}$ und $\tilde{\delta}(u, x_j, X) = \delta(w_2)$ (siehe Abbildung 7.18).

Angenommen, es gibt einen kürzesten X-freien-Weg $w \in P(u, x)$, der über keinen Knoten von $Clock(y_{i'}, y_{j'})$ führt. O.B.d.A. kreuzen sich die Wege w und w_1 im Knoten t, weil der Graph G planar ist. Nach Induktionsannahme und Lemma 17 ist $Pr(w_1, t)$ ein kürzester X-freier-Weg, und es folgt:

$$\begin{aligned}
\delta(w) &= \delta(Pr(w, t)) + \delta(Suf(w, t)) \\
&\geq \delta(Pr(w_1, t)) + \delta(Suf(w, t)) \\
&\geq \delta^u(u, y_{i'}, Y) + \delta(y_{i'}, x) \\
&\geq \min\{\delta^u(u, v, Y) + \delta(v, x) \mid v \in Clock(y_{i'}, y_{j'})\}
\end{aligned}$$

Daraus folgt Gleichung 7.28.

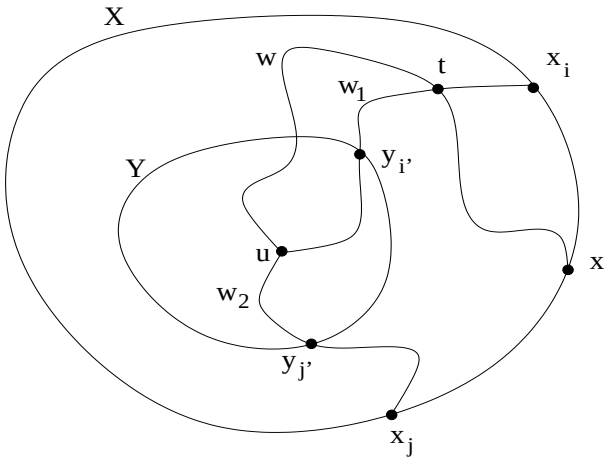

Abbildung 7.18: Anschauung zu Lemma 18.

Weil die maximale Rekursionstiefe $\lceil \log |X| \rceil$ beträgt und in jeder Rekursionstiefe höchstens $O(|Y|)$ Knoten zu betrachten sind, folgt der Aufwand $O(|X| + |Y| \log |X|)$.

■

Theorem 15 *Gegeben seien ein planarer Graph G, ein $f(n)$-Kreis-Separatorbaum T und ein DTHG G^T, wobei die Funktion $f()$ die Gleichungen 7.6 und 7.7 erfüllt. Dann lässt sich SPSD in $O(f(n) \log^2 n)$ Operationen berechnen.*

Beweis: Gesucht sei $\delta(u, v)$ für die Knoten $u, v \in V(G)$. O.B.d.A. liegen die Knoten u, v in den Separatormengen S_u und S_v mit $S_u \neq S_v$. $P = [q_1, \ldots, q_k]$ sei der eindeutige Weg in dem günstigen Separatorbaum T mit $l^2(q_1) = S_u, l^2(q_k) = S_v$. Wir bestimmen eine untere S_i-Schranke $\delta^u(u, x, S_i)$ für alle Knoten $x \in S_i$ und ein $1 < i \leq k$, dann eine untere S_j-Schranke für ein $i < j \leq k$ usw. (s. Abbildung 7.19).

Die Lösung ergibt sich dann durch die Berechnung einer unteren $\{v\}$-Schranke $\delta^u(u, v, \{v\})$ aus der unteren S_k-Schranke $\delta^u(u, x, S_k)$:

$$\delta(u, v) \quad = \quad \min\{\delta^u(u, x, S_k) + \delta(x, v) \mid x \in S_k\}$$

Der Einfachheit halber nehmen wir im Weiteren an, dass der Weg P ein absteigender Weg in dem Separatorbaum T sei. Ansonsten ist der aufsteigende und der absteigende Teilweg von P gesondert zu behandeln bezüglich der trennenden Pfade, die dann zu spiegeln sind. Sei $\delta^u(u, x, S_i)$ für ein $1 < i \leq k$ und alle $x \in S_i$ bestimmt und S_i separiert den Knoten u und die

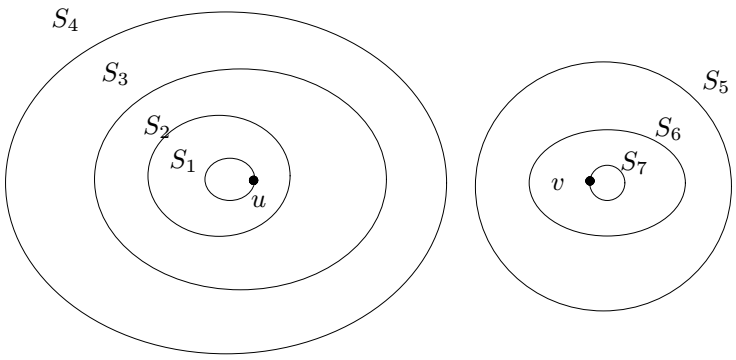

Abbildung 7.19: Beispielseparatoren $S_1 = S_u, S_2, \ldots, S_6, S_7 = S_v$ zu einem Weg $P = [q_1, \ldots, q_7]$.

Knoten $l^1(q_{i+1})$. Das längste Präfix von $q_i, q_{i+1}, \ldots, q_k$, das ein trennender Pfad ist, sei $q_i, q_{i+1}, \ldots, q_j$ für $j > i$. Alle Wege von u nach S_j führen über S_i, weil S_i nach Voraussetzung u und $S_j \subseteq l^1(_{i+1})$ separiert. Weiterhin gilt nach Lemma 16, dass S_j den Knoten u und die Knoten $l^1(q_{j+1})$ separiert, weil sonst der trennende Pfad länger wäre. Die Entfernungen $\delta^u(u, x, S_i)$ für $x \in S_i$ sind nach Voraussetzung bereits bestimmt, und die Entfernungen $\delta(x, y)$ für $x \in S_i$ und $y \in S_j$ liegen im DTHG nach Konstruktion vor. Somit berechnen wir eine untere S_j-Schranke $\delta^u(u, y, S_j)$ für alle Knoten $y \in S_j$ nach Lemma 18 in $O(|S_j| + |S_i| \log |S_j|)$ Operationen. Wegen $k \in O(\log n)$ ergibt eine Summation über die Separatormengen des Weges P die Aussage:

$$\begin{aligned} \sum_{i=2}^{k} O(|S_i| + |S_{i-1}| \log |S_i|) &\in O(f(n) \log n \log f(n)) \\ &\in O(f(n) \log^2 n) \end{aligned}$$

■

Korollar 14 *Für planare Graphen können Dünne-Trennende-Hierarchische-Graphen fast linearer Größe $O(n \log n)$ in $O(n^{1.5})$ Operationen berechnet werden, so dass SPSD in $O(\sqrt{n} \log^2 n)$ gelöst werden kann.*

Es stellt sich die Frage, ob nicht der folgende einfachere Ansatz bereits zu diesem Erfolg geführt hätte. In einem günstigen Separatorbaum T seien $q, q' \in V(T)$ direkt aufeinander folgende Knoten $q' \in N(q)$. Führe für jedes Knotenpaar $u \in l^2(q), v \in l^2(q')$ eine Kante ein, die mit der kürzesten Entfernung beschriftet wird. Verahre so mit allen Nachbarknoten von T. Wende zur Online-Entfernungsberechnung nun das im Beweis zu Theorem 15 entwickelte Verfahren an. Dies führt i. A. nicht zur Berechnung der kürzesten Entfernung wie

man beispielsweise anhand der Knoten $u \in S_3 = l^2(q_3)$ und $v \in S_0 = l^2(q_0)$ in Abbildung 7.17 feststellen kann. Würde man hier sukzessive die Entfernungen zwischen S_2 und S_3, dann zwischen S_1 und S_3 und dann zwischen S_0 und S_3 bestimmen, ist unklar, wie die bereits berechneten Entfernungen in weitere Berechnungen einfließen sollen. Schließlich verlaufen Wege zwischen S_0 und S_3 eben nicht unbedingt über die Knoten von S_1.

7.8 Hierarchische-Graphen zur Approximation

In diesem Abschnitt betrachten wir hierarchische Graphen fast linearer Größe, mit denen SPSD in fast konstanter Zeit approximiert wird. Wir nehmen wiederum einen Separatorbaum als gegeben an und markieren dessen Knoten analog zum THG mit Graphen, allerdings nur mit dünnen Graphen.

Definition 24 (Einfacher-Trennender-Hierarchischer-Graph ETHG) *Gegeben seien ein Graph $G = (V, E, \delta)$ und ein Separatorbaum $T = (Q, E_Q, l)$. Die Kantengewichtsfunktion der induzierten Graphen $G[l^1(q)]$ in G bezeichnen wir als δ'_q. Jeder Knoten $q \in Q$ sei mit einem Graphen $G_q = (l^1(q), E_q, \delta_q)$ markiert, wobei für E_q und die Kantengewichte δ_q gilt:*

$$\begin{aligned} E_q &= E_q^1 \cup E_q^2 \\ E_q^1 &= \{\{u,v\} \subseteq l^1(q) \mid u \notin l^2(q), v \in l^2(q), \delta'_q(u,v) = \delta'_q(u, l^2(q))\} \\ E_q^2 &= l^2(q) \times l^2(q) \\ \delta_q(\{u,v\}) &= \delta_q(u,v) \end{aligned}$$

δ_q sei die Kantengewichtsfunktion von dem induzierten Graphen $G[l^1(q)]$ in G.

Für die Knoten u, v, die durch einen Separator getrennt werden, werden in einem ETHG nur die kürzesten Entfernungen von u und v zu dem 'nächsten' Knoten des Separators vermerkt (E_q^1). Wir gehen im Weiteren davon aus, dass es nur einen 'nächsten' Knoten gibt, so dass E_q^1 also genau aus $|l^1(q)||l^2(q)|$ Knoten besteht. Jede Separatormenge selbst ist vollständig durch Kanten verbunden (E_q^2).

Satz 22 *Gegeben sei ein α-β-$f(n)$-Separatorbaum für $f(n) \in O(\sqrt{n})$. Ein ETHG der Größe $O(n \log n)$ wird in $O((m + n \log n) f(n))$ Operationen aufgebaut.*

Beweis: Die Größe des ETHG ist durch die Anzahl der Kanten bestimmt. Für einen Graphen G_q sind höchstens $O(|l^1(q)| + |l^2(q)|^2) = O(|l^1(q)|)$ Kanten zu berücksichtigen, so dass mit Rekursionsgleichung 7.27 eine maximale Größe von $O(n \log n)$ folgt.

Seien wiederum $Q_i \subseteq V(T)$ die Knoten in Level i und $l_{max,i} = \max\{|l^2(q)| \mid q \in Q_i\}$. Zur Entfernungsberechnung verwenden wir analog zum Beweis von Theorem 9 den *Verteilten Dijkstra* Algorithmus. Zur Berechnung der Kantengewichte von E_q^1 genügt es, den *Verteilten Dijkstra* in jedem Level

nur einmal durchzuführen, d.h. es genügen $O((m + n \log n) \log n)$ Operationen. Um die Kanten E_q^2 entsprechend zu gewichten, ist der *Verteilte Dijkstra* $l_{max,i}$ mal im Level i durchzuführen, d.h. es ergibt sich die angegebene Laufzeit.

■

SPSD verläuft im ETHG analog zum THG, allerdings wird für jede Separatormenge nur eine Kante betrachtet. Gegeben seien zwei Knoten $u, v \in V$ und ein Knoten $q \in Q$ mit maximaler Tiefe mit $u, v \in l^1(q)$. $P = [q_0, \ldots, q_k = q]$ sei ein Weg in dem Separatorbaum T von der Wurzel q_0 nach q. Wir bezeichnen den folgenden Wert als **Entfernungsapproximation** für die Knoten u, v:

$$\tilde{\delta}(u,v) \quad = \quad \min\{\delta_{q_i}(u,x) + \delta_{q_i}(x,y) + \delta_{q_i}(y,v) \mid \{x,y\} \in \bigcup_{i=0}^{k} l^2(q_i)\} \quad (7.29)$$

Den **Approximationsfehler** definieren für die Knoten u, v als:

$$\xi(u,v) \quad = \quad \tilde{\delta}(u,v) - \delta(u,v) \quad (7.30)$$

Für jeden Knoten aus $l^1(q)$ gibt es nur einen 'nächsten' Knoten in $l^2(q)$, so dass der Übergang, d.h. die Kante $\{x, y\}$ in Gleichung 7.29, durch den Separator eindeutig bestimmt ist. Somit folgt:

$$\mathcal{Q} \quad \in \quad O(\log n)$$

Theorem 16 *Gegeben sei ein Graph $G = (V, E, \delta)$ mit ETHG T. Dann gilt für die Entfernungsapproximation und den Approximationsfehler für alle Knoten $u, v \in V$:*

$$\begin{aligned} \tilde{\delta}(u,v) &\leq 3\delta(u,v) \\ \xi(u,v) &\leq \max\{\delta_q(x,y) \mid x,y \in l^2(q), q \in V(T)\} \end{aligned}$$

Beweis: Gegeben seien die Knoten $u, v \in V$, ein Weg $w \in SP(u,v)$ und ein Knoten $q \in V(T)$ mit $V(w) \subseteq l^1(q)$, d.h. der induzierte Graph $G[l^1(q)]$ enthält einen kürzesten Weg zwischen u und v, und $l^2(q)$ separiert u, v.

$x, y \in l^2(q)$ seien so gewählt, dass gilt:

$$\begin{aligned} \delta_q(u,x) &= \min\{\delta_q(u,z) \mid z \in l^2(q)\} \\ \delta_q(v,y) &= \min\{\delta_q(v,z) \mid z \in l^2(q)\} \end{aligned}$$

Es gilt dann offensichtlich für jeden Knoten $t \in l^2(q)$

$$\delta_q(u,t) + \delta_q(t,v) \quad \leq \quad \delta(u,v) \quad (7.31)$$

Somit folgt:

$$\begin{aligned}
\tilde{\delta}(u,v) &\leq \delta_q(u,x) + \delta_q(x,y) + \delta_q(y,v) \\
&\underbrace{\leq}_{7.31} \delta(u,v) + \delta_q(x,y) \qquad (7.32) \\
&\leq 2\delta(u,v) + \delta_q(u,x) + \delta_q(y,v) \\
&\underbrace{\leq}_{7.31} 3\delta(u,v)
\end{aligned}$$

Die Abschätzung für die Fehlerapproximation folgt aus Gleichung 7.32.

■

In Abbildung 7.20 ist der Fall skizziert, bei dem der maximale Fehler eintritt.

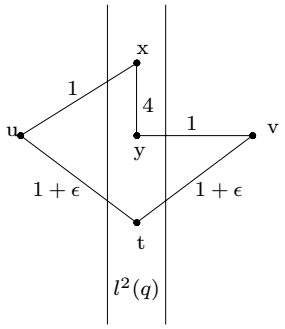

Abbildung 7.20: Maximaler Fehler bei Verwendung von ETHG.

Eine Verbesserung der Approximation kann man im Mittel dadurch erreichen, dass nicht nur der 'nächste' Knoten in der Separatormenge betrachtet wird, sondern eine Menge von 'nächsten' Knoten. Dies führt auf der anderen Seite allerdings auch zu einer höheren Laufzeit von ASPSD, weil sämtliche nächsten Knoten berücksichtigt werden müssen.

7.9 Zusammenfassung

In diesem Kapitel haben wir mit einem rekursiven Separationsansatz Separatorbäume definiert, die die Grundlage diverser Hierarchischer Graphen bilden. Für die Online-Laufzeit $\mathcal{Q}$ ist dabei die Größe der Separatormengen der entscheidende Parameter, den wir in Abschnitt 7.4 als die Separatorweite bezeichnet haben. Weil die Baumweite eine obere Schranke für die Separatorweite ist, kann für Graphen mit beschränkter Separatorweite auf fast linearem Platz und nach Theorem 9 in einer Preprocessing-Phase der Laufzeit $O(m + n \log n)$

ein Trennender-Hierarchischer-Graph aufgebaut werden, so dass die Online-Entfernungsberechnung in konstanter Laufzeit erfolgt. Weiterhin haben wir konstruktiv gezeigt, dass Separator- und Baumweite übereinstimmen und bis zu einem logarithmischen Wert voneinander abweichen können.
In Abschnitt 7.5 haben wir festgestellt, dass flache THG'en chordal sind, und chordale Graphen als Eingabegraphen untersucht. Für einen chordalen Graphen G kann ein THG konstruiert werden, so dass die Laufzeit der Online-Entfernungsberechnung linear in der Cliquengröße $\omega(G)$ ist.
Für planare Graphen haben wir Dünne-Trennende-Hierarchische-Graphen und eine stufenweise Entfernungsberechnung eingeführt, so dass $\mathcal{SQ} \in \tilde{O}(n^{1.5})$ gilt. Aufgrund der Tatsache, dass die Separatorweite bereits für Gittergraphen von der Größenordnung $O(\sqrt{n})$ ist, ist dies eine untere Schranke (bis auf logarithmische Faktoren) und damit optimal. In Abschnitt 7.8 haben wir Separatorbäume nur um wenige Kanten zu Einfachen-Trennenden-Hierarchischen-Graphen erweitert, so dass auf fast linearem Platz eine Approximation für die Entfernung in konstanter Zeit berechenbar ist. Die Ergebnisse bezüglich der Laufzeiten und des Speicherplatzes sind in der folgenden Tabelle zussammengefasst.

	$\mathcal{T}$	$\mathcal{S}$	$\mathcal{Q}$
THG	$O((m + n \log n) f(n))$	$O(nf(n))$	$O(f(n))$
THG und $S_\alpha(G) \leq k$	$O(m + n \log n)$	$O(n \log n)$	$O(1)$
DTHG und G planar	$O(n^{1.5})$	$O(n \log n)$	$O(\sqrt{n} \log^2 n)$
ETHG und $f(n) = \sqrt{n}$	$O(n^{1.5} \log n)$	$O(n \log n)$	$O(\log n)$
G chordal	$O(n\omega(G) \log n)$	$O(n\omega(G) \log n)$	$O(\omega(G))$

Kapitel 8

Zusammenfassung und Ausblick

Für Bäume, kreisfreie Graphen und Graphen, für die innerhalb jeder 2-Zusammenhangskomponente die Online-Entfernungsberechnung in konstanter Laufzeit möglich ist, haben wir in Kapitel 5 optimale Werte für $\mathcal{S}, \mathcal{T}$ und $\mathcal{Q}$ gezeigt, d.h. eine lineare Preprocessing-Phase genügt, um SPSD in konstanter Laufzeit zu berechnen.

In den Kapiteln 6 und 7 haben wir zwei konträre Ansätze zur Konstruktion Hierarchischer Graphen untersucht. Geeignete Knotenmenge wie auch die Separatormengen sollten möglichst klein sein, jedoch separieren Geeignete Knotenmengen den Graphen i. A. nicht, und Separatormengen sind i. A. keine Geeigneten Knotenmengen. Die Berechnung minimaler Geeigneter Knotenmengen und die Berechnung minimaler Umgebungsabstände haben wir als NP-vollständig nachgewiesen und eine logarithmische bzw. konstante Approximation (für den Umgebungsabstand) angegeben. Außerdem haben wir eine Verallgemeinerung Geeigneter Graphen zu teilgeeigneten Graphen definiert und das Eignungskonzept auf Levelgraphen übertragen.

In Kapitel 7 haben wir ein rekursives Separationskonzept - Separatorbäume - eingeführt und darauf verschiedene Hierarchische Graphen definiert. Sofern nur ein α-β-$f(n)$-Separatorbaum zur Verfügung steht, haben wir $\mathcal{S} \in O(n(f(n) + \log n))$, $\mathcal{T} \in O((m + n \log n) f(n))$ und $\mathcal{Q} \in O(f(n))$ nachgewiesen. Aus den Separatorbäumen haben wir die Separatorweite als einen neuen charakteristischen Parameter eines Graphen abgeleitet und gezeigt, dass sie i. A. von der Baumweite verschieden ist und höchstens um einen logarithmischen Faktor von der Baumweite abweicht. Hier ist die These naheliegend, dass einige i.A. NP-vollständige Probleme für Graphen mit konstanter Separatorweite ebenso wie für Graphen mit konstanter Baumweite in P liegen. Dies ist noch zu prüfen. Für planare Graphen konnten wir mittels sogenannter Dünner-Trennender-Hierarchischer-Graphen zeigen, dass fast ein linearer Speicherplatz genügt, um $\mathcal{Q} \in \tilde{O}(\sqrt{n})$ zu gewährleisten. Das Konzept der Separation eignet sich auch für Approxi-

mationen. Für beliebige Graphen kann auf fast linearem Platz ein Einfacher-Trennender-Hierarchischer-Graph berechnet werden, so dass online die Entfernung in fast konstanter Laufzeit approximiert wird.
Für beide untersuchten Ansätze zur Konstruktion Hierarchischer Graphen haben sich weitere Fragestellungen ergeben. Für Geeignete Graphen sollte insbesondere das Kriterium für die Umgebungsgröße variiert werden. Um dabei auch konkrete Werte für $\mathcal{S}, \mathcal{T}$ und $\mathcal{Q}$ in der O-Notation zu erhalten, muss das Eignungskonzept auf spezielle Graphklassen, ähnlich den k-Levelgittergraphen in Abschnitt 6.6.1, angewendet werden.
Der Ansatz der rekursiven Separation lässt sich eventuell dadurch verbessern, dass man Separationen nur bezüglich der kürzesten Wege betrachtet, d.h. man ändert in der Definition 10 die Gleichung 7.1 zu:

$$\forall_{u \in X_1} \forall_{v \in X_2} \forall_{w \in SP(u,v)} \qquad V(w) \cap Y \neq \emptyset$$

Hier benötigt man dann ein neues Separationsverfahren ähnlich dem von Lipton und Tarjan, das wir in dieser Arbeit benutzt haben [LT79].
Interessant ist außerdem, ob sich das Konzept der DTHG auch auf andere nichtplanare Graphen adaptieren lässt.
Kombinationen der beiden Hierarchiekonzepte und die Angemessenheit, um dynamische Aspekte von Graphen abzubilden, sollten ebenfalls künftig untersucht werden. Insbesondere im Zusammenhang mit Verkehrsinformationssystemen besteht hier ein großer Bedarf. Aus theoretischer Sicht sind die unteren Schranken für die Parameter $\mathcal{S}, \mathcal{T}$ und $\mathcal{Q}$ i. A. und abhängig von den gegebenen Graphen zu untersuchen. Einen Hinweis liefert hier das Resultat $\mathcal{SQ} \in \tilde{O}(n^{1.5})$ für planare Graphen, das modulo logarithmischer Faktoren optimal ist.

Index

Literaturverzeichnis

[ABCP93] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear cost sequential and distributed constructions of sparse neighborhood covers. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 638–647, Palo Alto, CA, November 1993.

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM J. Comput.*, 28(1):263–277, 1998.

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *Journal Alg. Disc. Meth.*, 8:277 – 284, 1987.

[ADDJ90] Ingo Althöfer, Gautam Das, David P. Dobkin, and Deborah Joseph. Generating sparse spanners for weighted graphs. In John R. Gilbert and Rolf G. Karlsson, editors, *SWAT 90, 2nd Scandinavian Workshop on Algorithm Theory*, volume 447 of *Lecture Notes in Computer Science*, pages 26–37, Bergen, Norway, 11–14 1990. Springer.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 503–513, St. Louis, MS, October 1990.

[AST90] Noga Alon, Paul Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3:801–808, 1990.

[Ber88] Claude Berge. *Hypergraphs*, volume 3 of *Selected Topics in Graph Theory, ed. Lowell W. Beineke and Robin J. Wilson.* Academic Press, 1988.

[Bod93] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. 25th Symp. Theory of Computing*, pages 226–234. ACM, 1993.

[Bod97] Hans L. Bodlaender. Treewidth: Algorithmic Techniques and Results. In *Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 19 – 36, 1997.

[Bol78] Béla Bollobás. *Extremal Graph Theory*. L.M.S. Monographs. Academic Press. Inc. London, 1978.

[BP69] Lowell W. Beineke and R. E. Pippert. The number of labeled k-dimensional trees. *Journal Combinatorial Theory*, (6):200–205, 1969.

[BPT91] Richard B. Borie, R. Gary Parker, and Craig A. Tovey. Deterministic decomposition of recursiver graph classes. *Journal Disc. Meth.*, 4:481–501, 1991.

[BR97] Friedhelm Buchholz and Bernhard Riedhofer. Hierarchische Graphen zur kürzesten Wegesuche in planaren Graphen. Fakultätsbericht Nr. 1997/13, Universität Stuttgart, 1997.

[Bra94] Andreas Brandstädt. *Graphen und Algorithmen*. Leitfäden und Monographien der Informatik. Teubner, Stuttgart, 1994.

[BS81] Ilja N. Bronstein and Konstantin A. Semendjajew. *Taschenbuch der Mathematik*. H. Deutsch, Thun; Frankfurt, 20. Aufl. edition, 1981.

[Car93] Adrijana Car. Hierarchisches Strassennetz - Konzept für effiziente Wegesuche. In *Proc. Grazer Geoinformatiktage*, pages 31–38, Graz, Austria, 1993.

[Car96] Adrijana Car. *Hierarchical Spatial Reasoning: Theoretical Consideration and its Application to Modeling Wayfinding*. PhD thesis, Department of Geoinformation TU Wien, 1996.

[CDNS92] Barun Chandra, Guatam Das, Giri Narasimhan, and Jose Soares. New sparseness results on graph spanners. In *Proc. 8th ACM Symposium on Computational Geometry*, 1992.

[CF93] Adrijana Car and Andrew U. Frank. Hierarchical street networks as a conceptual model for efficient way finding. In *Proc. EGIS'93*, pages 134–139, Genova, Italy, 1993.

[CF94] Adrijana Car and Andrew U. Frank. Modelling a hierarchy of space applied to large road networks. In *Proceedings of IGIS'94, International Workshop on Advanced Research in Geographic Information Systems*, pages 15–24, Ascona, Switzerland, 1994.

[Chu91] Fan R. K. Chung. Improved separators for planar graphs. In *ICTAG: 6th Quadrennial International Conference on the Theory and Applications of Graphs*, 1991.

[Chv79] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233 – 235, 1979.

[CLR96] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. Number XVII in The MIT electrical engineering and computer science series. MIT Press, Cambridge, Mass. [u.a.], 17. print. edition, 1996.

[CM93a] Bruno Courcelle and Mohamed Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theoretical Computer Science*, 109(1–2):49–82, March 1993.

[CM93b] John Richard Current and M. Marsh. Multiobjective design of transportation networks: Taxonomy and annotation. *European Journal of Operational Research*, 65(1):146, 1993.

[Coh93] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t (extended abstract). In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1993.

[Cou89] Bruno Courcelle. The Monadic Second-Order Logic of Graphs, II: Infinite graphs of bounded width. *MST: Mathematical Systems Theory*, 21:187–221, 1989.

[Cou90] Bruno Courcelle. The Monadic Second-Order Logic of Graphs, I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[Cou92] Bruno Courcelle. The Monadic Second-Order Logic of Graphs, III: tree–width, forbidden minors and complexity issues. *Informatique Théorique*, 26:257–286, 1992.

[CZ97] Edith Cohen and Uri Zwick. All Pairs Small Stretch Paths. In *Proc. of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 93–102, January 1997.

[DHZ97] Dorit Dor, Shay Halperin, and Uri Zwick. All pairs almost shortest paths. In *Electronic Colloquium on Computational Complexity, technical reports*. 1997.

[Die96] Reinhard Diestel. *Graphentheorie*. Springer, Berlin [u.a.], 1996.

[Dij59] Edsger W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269 – 271, 1959.

[Dji96] Hristo N. Djidjev. Efficient Algorithms for Shortest Path Queries in Planar Digraphs. In *Proceedings 22nd Workshop on Graph-Theoretic Concepts in Computer Science (WG '96)*, Lecture Notes in Computer Science, pages 151–165. Springer Verlag, 1996.

[DP84] Narsingh Deo and Chi-Yin Pang. Shortest path algorithms: taxonomy and annotation. *Networks*, 14:275 – 323, 1984.

[DP85] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A^*. *Journal of the Association for Computing Machinery*, 15(3):505 – 536, 1985.

[DV97] Hristo N. Djidjev and Shankar M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231 – 243, 1997.

[FG88] Thomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Ann. ACM Symp. Theory Comput.*, pages 434–444, 1988.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on computing*, 16(6):1004 – 1022, 1987.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[FW90] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *Proc. 31st Symp. Foundations of Computer Science*, pages 719–725. IEEE, 1990.

[Gel77] D. Gelperin. On the optimality of A*. *Artificial Intelligence*, 8(1):69–76, 1977.

[GHT84] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs with bounded genus. *Journal of Algorithms*, 5:391–407, 1984.

[GJ79] Michael R. Garey and David S. Johnson. *Computer and intractability : a guide to the theory of NP-completeness*. Computer science : mathematics. Freeman, San Francisco, 1979.

[GK81] Daniel H. Greene and Donald E. Knuth. *Mathematics for the analysis of algorithms*. Progress in computer science ; 1. Birkhäuser, Boston [u.a.], 1981. 107 S.

[Gol92] Martin C. Golumbic. *Algorithmic graph theory and perfect graphs*. Computer science and applied mathematics. Academic, Boston [u.a.], 1992.

[Hal95] Magnus M. Halldórsson. Approximating discrete collections via local improvements. In *Proc. 6th Annual: Symposium on Discrete Algorithms*, 1995.

[Har69] Frank Harary. *Graph Theory.* Addison-Wesley, Reading, Mass., 1969.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of System Science and Cybernetics*, SSC-4(2):100–107, 1968.

[HT84] Dov Harel and Robert Endre Tarjan. Fast Algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[JP96] Sungwon Jung and Sakti Pramanik. HiTi graph model of topographical road maps in navigation systems. In *Proc. IEEE 12th Int. Conf. Data Engineering*, pages 76–84. New Orleans, LA, 1996.

[Jun95] Sungwon Jung. *Recursive query processing in large databases.* PhD thesis, Dept. of Computer Science, Michigan State Univ., 1995.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[Klo94] Ton Kloks. *Treewidth: Computations and Approximations.* LNCS 842. Springer, Berlin, Heidelberg, 1994.

[KLPS90] Bernhard Korte, László Lovász, Hans Jürgen Prömel, and Alexander Sschrijver. *Paths, Flows and VLSI-Layout.* Algorithms and combinatorics, 9. Springer Verlag, Berlin, Heidelberg, New York, 1990.

[Knu97] Donald E. Knuth. *The art of computer programming*, volume 1. Addison-Wesley, Reading, Mass. [u.a.], 1997.

[KP92] Guy Kortsarz and David Peleg. Generating sparse 2-spanners. In *Proc. of the 3rd Scandinavian Workshop on Algorithm Theory*, volume 621 of *LNCS*, pages 73–82, Helsinki, Finland, July 1992.

[KP98] Guy Kortsarz and David Peleg. Generating Low-Dedgree 2-spanners. *SIAM Journal Computing*, 27(5):1438–1456, 1998.

[KRRS94] Philip N. Klein, Satish Rao, Monika H. Rauch, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. In *Proc. 26th Symp. Theory of Computing*, pages 27–37. Assoc. for Computing Machinery, 1994.

[Lin90] Andrzej Lingas. Efficient parallel algorithms for path problems in planar directed graphs. In *ISAAC: 1st International Symposium on Algorithms and Computation (formerly SIGAL International Symposium on Algorithms), Organized by Special Interest Group on Algorithms (SIGAL) of the Information Processing Society of Japan (IPSJ) and the Technical Group on Theoretical Foundation of Computing of the Institute of Electronics, Information and Communication Engineers (IEICE))*, 1990.

[Lov75] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, (13):383 – 390, 1975.

[LR95] Arthur L. Liestman and Dana S. Richards. Degree-constrained pyramid spanners. *Journal Parallel Distributed Computing*, 25:1–6, 1995.

[LRC+92] Guy Lapalme, Jean-Marc Rousseau, Suzanne Chapleau, Michel Cormier, Pierre Cossette, and Serge Roy. GeoRoute. *Communications of the ACM*, 35(1):80–88, 1992.

[LS93a] Arthur L. Liestman and Thomas C. Shermer. Additive graph spanners. *NETWORKS: An International Journal*, 23:343 – 364, 1993.

[LS93b] Arthur L. Liestman and Thomas C. Shermer. Grid spanners. *NETWORKS: An International Journal*, 23:123 – 133, 1993.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal Appl. Math.*, 36:177 – 189, 1979.

[LT80] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar seperator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, (41):960 – 982, 1994.

[Mil86] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, June 1986.

[Pea84] Judea Pearl. *HEURISTICS: Intelligent Search Strategies for Computer Problem Solving*. Reading Mass. Addison Wesley, 1984.

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

[Rie97] Bernhard Riedhofer. Hierarchische Straßengraphen. Diplomarbeit 1452, Universität Stuttgart, Institut für Informatik, 1997.

[RS86a] Neil Robertson and Paul D. Seymour. Graph minors II: algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[RS86b] Neil Robertson and Paul D. Seymour. Graph minors V: excluding a planar graph. *J. Combinatorial Theory, Series B*, 41:92–114, 1986.

[RTL76] Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elemination on graphs. *Journal on computing*, (5):266 – 283, 1976.

[Sim92] Klaus Simon. *Effiziente Algorithmen für perfekte Graphen.* Leitfäden und Monographien der Informatik. Teubner, Stuttgart, 1992.

[SKC93] Shashi Shekhar, Ashim Kohli, and Mark Coyle. Path computation algorithms for Advanced Traveler Information System (ATIS). In *Proc. IEEE 9th Int. Conf. Data Engineering*, pages 31–39, 1993.

[Sla95] Petr Slavík. Improved performance of the greedy algorithm for the minimum set cover and minimum partial cover problems. In *Electronic Colloquium on Computational Complexity, technical reports.* 1995. http://www.eccc.uni-trier.de/eccc/.

[Soa94] Jose Soares. Approximating euclidean distances by small degree graphs. *GEOMETRY: Discrete & Computational Geometry*, 11, 1994.

[Sti97] Günther Stiege. An algorithm for Finding the Connectivity Structure of Undirected Graphs. Hildesheimer Informatik Bericht 13/97, Universität Hildesheim, Mai 1997.

[Sti98] Günther Stiege. Edge Partitions in Undirected Graphs. Informatik Bericht 5/98, Universität Oldenburg, Sept. 1998.

[TY84] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclity of hypergraphs, and selectively reduce acyclic hypergraphs. *Journal on Computing*, 13(3):566 – 579, 1984.

[WB89] Klaus Wagner and Rainer Bodendiek. *Graphentheorie I - Anwendung auf Topologie, Gruppentheorie und Verbandstheorie*, volume 1. B.-I.-Wissenschaftsverlag, Mannheim [u.a.], 1989.

[Weg93] Ingo Wegener. *Theoretische Informatik : Eine algorithmische Einführung.* Leitfäden und Monographien der Informatik. Teubner, Stuttgart, 1993.

[WV74] Hans Joachim Walther and Heinz Jürgen Voß. *Über Kreise in Graphen.* Dt. Verl. d. Wiss., Berlin, 1974.

[Yan81] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *Journal Alg. Disc. Meth.*, 2:77 – 79, 1981.

Lebenslauf

Geburtstag:	22. September 1964
Geburtsort:	Rotenburg / Fulda
Familienstand:	verheiratet, 2 Kinder

1971 - 1975	Grundschule Fritzlar
1975 - 1984	Gymnasium Fritzlar (Abitur)
1984 - 1987	Verkehrsunfall und Rehabilitation
1987 - 1990	Adolf-Lazi-Schule (AV-Mediendesign), Esslingen
1990	Abschluss als AV-Mediendesigner
1989 - 1995	Studium der Informatik mit Nebenfach Mathematik an der Universität Stuttgart
1995	Diplom in Informatik
1995 - 2000	wissenschaftlicher Mitarbeiter am Lehrstuhl 'Formale Konzepte' des Instituts für Informatik der Universität Stuttgart
seit 1.6.2000	Softwareentwickler bei debitel