Graphentheoretische Methoden der Optimierung

Prof. Dr. D. Schweigert
Kaiserslautern, SS 2000

Verfügbar unter:

http://kluedo.ub.uni-kl.de/Mathematik/Metadaten/script_7.html

Zusammenfassung von Markus Heidenreich

Inhaltsverzeichnis

1 Grundbegriffe

1.1 Graphen

Definition 1.1.1 Ein Graph ist ein Tripel $G = (V, E, \varphi)$, das aus einer nichtleeren Menge $V = V(G)$ von Ecken, einer davon disjunkten Menge $E = E(G)$ von Kanten und einer Inzidenzfunktion $\varphi = \varphi_G$ besteht. Die Inzidenzfunktion φ ordnet jeder Kante e ein ungeordnetes Paar a, b nicht unbedingt verschiedener Ecken zu: $\varphi(e) = ab$.

Definition 1.1.2 Seien a, b zwei Ecken eines Graphen G, die durch eine Kante e verbunden sind, $\varphi(e) = ab$. Die Kante e heißt *inzident* mit den Ecken a, b und umgekehrt heißen die Ecken a, b *inzident* mit der Kante e. Zwei Ecken a, b, die mit einer gemeinsamen Kante inzident sind, heißen *adjazent* (*benachbart*).

Definition 1.1.3 Eine Kante e mit $\varphi(e) = ab$ und $a = b$ heißt *Schlinge* (*Schleife*). Zwei Kanten e_1, e_2 mit $\varphi(e_1) = \varphi(e_2) = ab$ heißen *Mehrfachkante*.

Definition 1.1.4 G heißt *schlicht* (*einfach*), wenn G keine Schlingen und keine Mehrfachkanten hat. Ein Graph, der nicht schlicht ist, heißt auch *Multigraph*.

Definition 1.1.5 G heißt *endlich*, wenn die Eckenmenge V und die Kantenmenge E endlich sind. Dann wird die Anzahl der Ecken mit $\nu = \nu(G)$ und die Anzahl der Kanten mit $\varepsilon = \varepsilon(G)$ bezeichnet.

Definition 1.1.6 Ein schlichter Graph G, bei dem jedes Paar verschiedener Ecken durch eine Kante verbunden ist, heißt *vollständig*.

Definition 1.1.7 G heißt *bipartit*, wenn die Menge der Ecken $V(G)$ sich in zwei disjunkte Teilmengen X und Y so zerlegen läßt, daß jede Kante eine Ecke in X und die andere Ecke in Y hat. (X, Y) heißt *Bipartition* des Graphen G. Ein bipartiter Graph G heißt *vollständiger bipartiter* Graph, wenn jede Ecke von X mit jeder Ecke von Y verbunden ist. Ist $|X| = l$ und $|Y| = m$, dann wird der vollständige bipartite Graph mit $K_{l,m}$ bezeichnet.

1.2 Isomorphismus

Definition 1.2.1 Zwei Graphen G und H heißen *isomorph*, wenn es bijektive Abbildungen $\alpha : V(G) \to V(H)$ und $\beta : E(G) \to E(H)$ gibt, sodaß gilt:

$$\varphi_G(e) = uv \quad \Leftrightarrow \quad \varphi_H(\beta(e)) = \alpha(u)\alpha(v) \qquad \text{(Inzidenzverträglichkeit)}$$

Das Paar (α, β) von Abbildungen heißt *Isomorphismus* zwischen den Graphen G und H.

Definition 1.2.2 Ein Graphenisomorphismus (α, β) mit $\alpha : V(G) \to V(G)$ und $\beta : E(G) \to E(G)$ eines Graphen G heißt *Automorphismus* von G. Die Automorphismen bilden eine Gruppe.

1.3 Untergraphen

Definition 1.3.1 Ein Graph H ist ein *Untergraph* von G, wenn gilt:

a) $V(H) \subseteq V(G)$

b) $E(H) \subseteq E(G)$

c) $\varphi_H(e) = \varphi_G(e)$ für alle $e \in E(H)$

Definition 1.3.2 Ein Untergraph H heißt *(auf-)spannender* Untergraph von G, wenn $V(H) = V(G)$.

Definition 1.3.3 Seien H_1 und H_2 Untergraphen von G. H_1 und H_2 heißen *disjunkt*, wenn $H_1 \cap H_2 = \emptyset$. $H_1 \cup H_2$ ist der Untergraph von G mit der Eckenmenge $V(H_1) \cup V(H_2)$, der Kantenmenge $E(H_1) \cup E(H_2)$ und der Inzidenzfunktion $\varphi_{H_1 \cup H_2}(e) = \varphi_G(e)$ für alle $e \in E(H_1) \cup E(H_2)$. Sind H_1 und H_2 disjunkt, dann schreiben wir $H_1 + H_2$ für die disjunkte Vereinigung.

Definition 1.3.4 Sei $G = (V, E, \varphi)$ ein Graph und V' nicht leere Teilmenge von V. Der Untergraph von G, der aus allen Ecken von V' und allen Kanten von G besteht, deren Endecken in V' sind, wird als $G[V']$ bezeichnet. Er heißt der durch V' *induzierte* Untergraph von G.

Definition 1.3.5 Sei $G = (V, E, \varphi)$ ein Graph und E' nicht leere Teilmenge von E. Der Untergraph von G, der aus allen Ecken, die mit den Kanten aus E' inzident sind und allen Kanten von E' besteht, wird mit $G[E']$ bezeichnet. Er heißt der durch E' *kanteninduzierte* Untergraph von G.

1.4 Inzidenz- und Adjazenzmatrix

Definition 1.4.1 Sei G gegeben mit $V = \{v_1, \ldots, v_\nu\}$ und $E = \{e_1, \ldots, e_\varepsilon\}$. Die *Inzidenzmatrix* von G ist $M(G) = (m_{ij})$, wobei die Zahl m_{ij} angibt, wie oft v_i und e_j indizent sind. Dabei zählt eine Schlinge als 2-fache Inzidenz.

Definition 1.4.2 Sei G gegeben mit $V = \{v_1, \ldots, v_\nu\}$. Die *Adjazenzmatrix* (*Nachbarschaftsmatrix*) von G ist $A(G) = (a_{ij})$, wobei die Zahl a_{ij} angibt, durch wieviele Kanten die Ecken v_i und v_j adjazent sind.

Bemerkung 1.4.1 Die Adjazenzmatrix ist symmetrisch, es muß also nur eine Dreiecksmatrix gespeichert werden.

1.5 Eckengrade

Definition 1.5.1 Der *Eckengrad* $d(v)$ einer Ecke v von G ist die Anzahl der Kanten, mit denen v inzident ist. Auch hier zählt eine Schlinge als doppelte Inzidenz. Man bezeichnet:

a) $\delta(G) = min\{d(v) : v \in V\}$

b) $\Delta(G) = max\{d(v) : v \in V\}$.

Satz 1.5.1 Die Summe der Eckengrade eines Graphen G ist gleich der doppelten Kantenzahl von G:

$$\sum_{v \in V} d(v) = 2\varepsilon.$$

Korollar 1.5.1 In jedem Graphen ist die Anzahl der Ecken mit ungeradem Eckengrad eine gerade Zahl.

2 Wege, Zykel und kürzeste Wege

2.1 Wege

Definition 2.1.1 Eine *Kantenfolge* K von G ist eine Folge $K = v_0 e_1 v_1 \dots e_k v_k$, deren Ausdrücke abwechselnd Ecken und Kanten sind, sodaß für $1 \leq i \leq k$ die Endecken von e_i genau v_{i-1} und v_i sind.

Definition 2.1.2 Eine Kantenfolge L heißt *Linie*, wenn die Kanten alle voneinander verschieden sind.

Definition 2.1.3 Eine Linie W heißt *Weg*, wenn die Ecken alle voneinander verschieden sind.

Bemerkung 2.1.1 In einem schlichten Graphen ist eine Kantenfolge, Linie, oder ein Weg bereits durch die Angabe der Ecken eindeutig bestimmt. Man nennt v_0 den *Anfang*, v_k das Ende und k die Länge von K. Unter einem (u, v)-Weg versteht man einen Weg mit Anfang u und Ende v.

Definition 2.1.4 Zwei Ecken u und v heißen *zusammenhängend* in einem Graphen G, wenn es einen (u, v)-Weg in G gibt. Sind alle Eckenpaare von G zusammenhängend, dann heißt G zusammenhängend.

Bemerkung 2.1.2 Der Zusammenhang ist eine reflexive, symmetrische, tranitive Relation auf der Menge der Ecken. Die Äquivalenzklassen heißen *Zusammenhangskomponenten*.

2.2 Zykel

Definition 2.2.1 Eine Kantenfolge, deren Anfangs- und Endecke identisch ist, heißt *geschlossen*. Jede Ecke außer der Anfangs- und Endecke wird als *innere Ecke* bezeichnet. Eine geschlossene Linie, deren innere Ecken verschieden sind, heißt *Zykel* (*Kreis*). Ein Zykel der Länge k heißt auch *k-Zykel*. Ist k gerade, dann heißt der Zykel *gerade*, ansonsten heißt er *ungerade*.

Satz 2.2.1 Ein Graph G ist genau dann bipartit, wenn G keine ungeraden Zykel hat.

2.3 Kürzeste Wege

Definition 2.3.1 Sei G ein Graph. Eine Funktion $\omega : E \to \mathbb{R}^+$, die jeder Kante aus E eine positive reelle Zahl zuordnet, heißt *Gewichtsfunktion*.

Definition 2.3.2 In einem gewichteten Graphen versteht man unter der *Länge* eines (u_0, v_0)-Weges die Summe der Gewichte über den beschrittenen Kanten.

Problem 2.3.1 Sei G ein zusammenhängender und schlichter Graph mit Gewichtsfunktion $\omega : E \to \mathbb{R}^+$, $u_0, v_0 \in V(G)$. Bestimme einen kürzesten Weg von u_0 nach v_0.

Algorithmus 2.3.1 (Dijkstra) Zur Bestimmung eines kürzesten Weges von u_0 nach v_0 dient folgender Algorithmus:

a) Setze $l(u_0) = 0$ und $l(u_i) = \infty$ für $i \neq 0$, $S_0 := \{u_0\}$, $i := 0$.

b) Für jedes $v \in \bar{S}_i$ ersetze $l(v)$ durch $min\{l(v), l(u_i) + \omega(u_i, v)\}$, es werden dabei alle $u_i \in S$ berücksichtigt. Berechne $min\{l(v) : v \in \bar{S}_i\}$ und bezeichne die Ecke, bei der das Minimum angenommen wird mit u_{i+1}. Setze $S_{i+1} := S_i \cup \{u_{i+1}\}$.

c) Stoppe, wenn $i = \nu - 1$.
Wenn $i < \nu - 1$, setze $i := i + 1$ und gehe nach b).

Bemerkung 2.3.1 (Komplexität) Im i-ten Schritt werden zur Bestimmung der Minima $\nu - i - 1$ Summen und zweimal $\nu - i - 1$ Vergleiche errechnet. Also werden

$$3 \cdot \sum_{i=0}^{\nu-1} \nu - i - 1 \quad = \quad 3 \cdot \left(\frac{\nu \cdot (\nu - 1)}{2} \right) \qquad \text{Schritte (Operationen)}$$

durchgeführt. Der Algorithmus ist also polynomial in der Größenordnung $O(\nu^2)$. Der Zeitaufwand eines Algorithmus hängt von den Eingabedaten und der Anzahl der Rechenschritte ab.

2.4 Komplexität

Definition 2.4.1 Seien $f, g : \mathbb{N}_0 \to \mathbb{R}^+$ gegeben. Die Funktion $f(n)$ ist in $O(g(n))$, d.h. in der Größenordnung von $g(n)$, wenn es eine Konstante $c \in \mathbb{R}^+$ gibt, sodaß $f(n) \leq c \cdot g(n)$ für alle $n \geq n_0$.

Definition 2.4.2 Ist f ein Polynom $f(x) = a_k x^k + \ldots + a_1 x + a_0$, so läßt sich f mit der höchsten auftretenden Potenz abschätzen:

$$f(n) = a_k n^k + \ldots + a_1 n + a_0 \quad \leq \quad \sum_{i=1}^{k} a_i n^k$$

mit $c = \sum a_i$ gilt also $f(n) \in O(n^k)$.

Definition 2.4.3 Seien $f, g : \mathbb{N}_0 \to \mathbb{R}^+$ monoton wachsende Funktionen. f wächst *asymptotisch* nicht schneller, als die Funktion g, wenn gilt:

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = c \quad \text{für eine Konstante } \; c > 0.$$

Definition 2.4.4 Ein Algorithmus heißt *polynomial*, wenn seine Rechenzeit durch ein Polynom abgeschätzt werden kann. Andernfalls heißt der Algorithmus *exponentiell.*

2.5 Digraphen

Definition 2.5.1 Ein *gerichteter Graph* oder *Digraph* (*directed graph*) ist ein Tripel $D = (V(D), A(D), \varphi_D)$, wobei $V(D)$ eine nichtleere Menge von Ecken, $A(D)$ eine von $V(D)$ disjunkte Menge von Bögen (arc) und $\varphi_D : A(D) \to V(D)^2$ eine Inzidenzfunktion ist, die jedem Bogen ein *geordnetes* Paar bestehend aus einer Anfangs- und Endecke zuordnet.

Bemerkung 2.5.1 Zu jedem Digraphen D gibt es den zugrundeliegenden Graphen G. Deshalb heißt D auch eine *Orientierung* von G.

Definition 2.5.2 Eine *gerichtete Kantenfolge* (*directed walk*) K ist eine Folge $v_0 a_1 v_1 \ldots a_k v_k$ von abwechselnd Ecken v_i und Bögen a_i, sodaß v_{i-1} die Anfangsecke und v_i die Endecke des Bogens a_i für $1 \leq i \leq k$ ist.
Eine *gerichtete Linie* (*directed trail*) L ist eine Kantenfolge, die im zugrundeliegenden Graphen G eine Linie ist.
Ein *gerichteter Weg* (*directed path*) L ist eine gerichtete Linie, die im zugrundeliegenden Graphen G ein Weg ist.

Definition 2.5.3 In einem Digraphen D heißt eine Ecke v erreichbar von einer Ecke u, wenn es einen gerichteten Weg von u nach v gibt. Zwei Ecken heißen *dizusammenhängend* (*diconnected*) in D, wenn jede der beiden Ecken von der anderen aus erreichbar ist.

Definition 2.5.4 Der *Eingangsgrad* (*indegree*) $d^-(v)$ einer Ecke v des Digraphen D ist die Anzahl der Bögen, die v als Endecke haben.
Der *Ausgangsgrad* (*outdegree*) $d^+(v)$ ist analog definiert.

Satz 2.5.1 In jedem Digraphen ist die Summe der Eingangsgrade aller Ecken gleich der Summe der Ausgangsgrade aller Ecken gleich der Anzahl der Bögen des Digraphen:

$$\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = \varepsilon.$$

Definition 2.5.5 Eine Orientierung eines vollständigen Graphen heißt *Turnier* (*tournament*).

3 Aufspannende Bäume

3.1 Bäume

Definition 3.1.1 Ein Graph G heißt *azyklisch* (oder ein *Wald*), wenn G keine Zyklen hat. Ist G azyklisch und zusammenhängend, dann heißt G *Baum*.

Bemerkung 3.1.1 Je zwei Ecken eines Baumes sind durch genau einen Weg verbunden, da ansonsten Zykel vorkommen würden.

Satz 3.1.1 Ist G ein Baum, dann gilt:

$$\varepsilon = \nu - 1.$$

Korollar 3.1.1 Jeder Baum G hat mindestens zwei Ecken vom Eckengrad 1, falls die Gesamtanzahl der Ecken $\nu(G) \geq 2$ ist.

Definition 3.1.2 Eine Kante e heißt *Schnittkante* von G, wenn die Ungleichung $\omega(G-e) > \omega(G)$ gilt, wobei $\omega(G)$ die Anzahl der Zusammenhangskomponenten von G ist.

Definition 3.1.3 Ein aufspannender Untergraph T von G, der ein Baum ist, ist ein *aufspannender Baum* von G.

Satz 3.1.2 Jeder zusammenhängende Graph G besitzt einen aufspannenden Baum.

Korollar 3.1.2 Ist G zusammenhängend, dann gilt:

$$\varepsilon \geq \nu - 1.$$

3.2 Anzahl der Bäume

Bezeichnung 3.2.1 Die Anzahl der aufspannenden Bäume eines Graphen G bezeichnet man mit $\tau(G)$.

Definition 3.2.1 Der Graph $G * e$ heißt *durch Kontraktion der Kante e aus G entstanden*, wenn man e aus G herausnimmt und die Endecken von e identifiziert.

Satz 3.2.1 Ist e eine Kante mit zwei verschiedenen Ecken, dann ist die Anzahl der aufspannenden Bäume

$$\tau(g) = \tau(G-e) + \tau(G * e).$$

Satz 3.2.2 (Cayley, 1889) Ein vollständiger Graph K_n mit n Ecken hat

$$\tau(K_n) = n^{n-2}$$

aufspannende Bäume.

3.3 Der Algorithmus von KRUSKAL

Problem 3.3.1 Gegeben ist ein zusammenhängender Graph G mit Gewichtsfunktion $\omega : E \to \mathbb{R}^+$. Gesucht ist ein aufspannender Baum von G mit minimalem Gewicht.

Algorithmus 3.3.1 (KRUSKAL) Zur Bestimmung eines aufspannenden Baumes mit geringstem Gewicht dient folgender Algorithmus:

a) Ordne die Kanten nach aufsteigendem Gewicht unter Benutzung eines Sortieralgorithmus $E = (a_1, \dots, a_\varepsilon)$ und setze $e_1 := a_1$.

b) Sind die Kanten $e_1, \dots, e_i$ bereits gewählt, so wähle $e_{i+1} := a_j$ aus $E \setminus \{e_1, \dots, e_i\}$ derart, daß:

   i) $G[(e_1, \dots, e_i)]$ azyklisch

   ii) $\omega(e_{i+1})$ so klein, als möglich

c) Stop, wenn Schritt b) nicht weiter durchgeführt werden kann.

Bemerkung 3.3.1 In Schritt b) benötigt man eine geeignete Unterprozedur, die die Anfrage $G[(e_1, \dots, e_i)]$ azyklisch durchführt. Man führt eine Markierung l ein, mit der bei jedem Schritt festgestellt werden kann, ob zwei Ecken zu der selben Komponente von $G[S]$ gehören. $l(v) = k$, falls v zur k-ten Komponente gehört. Wird nun a_j mit Endecken v_i und v_{i+1} hinzugenommen, dann gilt gerade $l(v) = k$.

Bemerkung 3.3.2 (Komplexität) Setzt man beispielsweise Bubblesort in Schritt a) ein, dann ist dieser in $O(\varepsilon^2)$. Für Schritt b) schätzt man $O(\nu^2)$ ab und erhält insgesamt $O(n^2)$ mit beispielsweise n Kanten.

Bemerkung 3.3.3 Man unterscheidet zwei Arten der Korrektheit:

a) Korrektheit eines Algorithmus:
Man zeigt dies mit Hilfe von (formalen) Beweisen

b) Korrektheit eines Programms:
Man unterscheidet hier partielle, nämlich ob ein Programm nach endlich vielen Schritten der Iteration endet. (Theorie von DANA SCOTT)
Es gibt keine Theorie für die Korrektheit eines Programms.

Satz 3.3.1 (Korrektheit) Jeder aufspannende Baum $T_1 = G[\{e_1, \dots, e_s\}]$, der durch den Algorithmus von KRUSKAL konstruiert wird, ist von minimalem Gewicht.

4 Bäume und Cluster

4.1 Datenanalyse

Bezeichnung 4.1.1 Man faßt die Objekte der Clusteranalyse in eine Datenmatrix der Größe $(m \times l)$ zusammen, wobei m die Anzahl der Objekte und l die Anzahl der Variablen ist:

$$X = \begin{pmatrix} x_{11} & \dots & x_{1l} \\ \vdots & & \vdots \\ x_{m1} & \dots & x_{ml} \end{pmatrix}$$

Sei U eine Menge von Elementen, die später aus Objekten besteht und die durch jeweils eine Zeile der Datenmatrix gegeben sind.

Definition 4.1.1 Eine Abbildung $d : U \times U \to \mathbb{R}$ heißt *Abstandsfunktion*, wenn für $x, y \in U$ und d_0 feste reelle Zahl gilt:

a) $d(x, y) \geq d_0$

b) $d(x, x) = d_0$

c) $d(x, y) = d(y, x)$

Definition 4.1.2 Eine Abstandsfunktion $d : U \times U \to \mathbb{R}$ heißt *Metrik*, wenn für alle $x, y, z \in U$ gilt:

d) aus $d(x, y) = d_0$ folgt $x = y$

e) $d(x, z) \leq d(x, y) + d(y, z)$ (Dreiecksungleichung)

Bezeichnung 4.1.2 Für die Vektoren $x, y \in \{0, 1\}^n$ setzt man:

$$\alpha = \sum_{k=1}^{n} min\{x_k, y_k\} \qquad \text{Anzahl der Positionen für } x = y = 1$$

$$\beta = \sum_{k=1}^{n} x_k - \alpha \qquad \text{Anzahl der Positionen für } x = 0 \text{ und } y = 1$$

$$\gamma = \sum_{k=1}^{n} y_k - \alpha \qquad \text{Anzahl der Positionen für } x = 1 \text{ und } y = 0$$

$$\delta = n - (\alpha + \beta + \gamma) \qquad \text{Anzahl der Positionen für } x = y = 0$$

Bemerkung 4.1.1 Mit Hilfe der in Bezeichnung 4.1.2 definierten Werte lassen sich sehr viele Abstandsfunktionen definieren:

a) $\frac{\alpha}{n}$ (Russel, Rao)

b) $\frac{\alpha}{\alpha+\beta+\gamma}$ (Jaccard)

c) $\frac{\alpha+\beta}{n}$ (simple matching)

Definition 4.1.3 Die besondere Abstandsfunktion $1 - \frac{\alpha}{\alpha+\beta+\gamma}$, die einen idealen Kompromiß darstellt heißt *Tanimotokoeffizient*.

4.2 Grundbegriffe über Clustering

Definition 4.2.1 Sei $M = \{x_1, \ldots, x_m\}$ eine nichtleere endliche Menge der zu gruppierenden Objekte. Eine Partition P_n der Länge n ist eine Zerlegung von M in nichtleere Teilmengen $C_1, \ldots, C_n$ für die gilt:

a) $\cup_{j=1}^{n} C_j = M$

b) $C_i \cap C_j = \emptyset$ für alle $i \neq j$, $i, j = 1, \ldots, n$

Die Teilmengen heißen Cluster.

Grundaufgabe 4.2.1 Sei M die Menge der zu gruppierenden Objekte. Suche eine Clusterzerlegung $\{C_1, \ldots, C_n\}$ in M, in n Cluster ($n \leq |M|$), sodaß eine vorgegebene Zielfunktion $Z(C_1, \ldots, C_n)$ opitmiert wird.

Grundaufgabe 4.2.2 Sei $M = \{x_1, \ldots, x_m\}$ eine Menge von Objekten. Suche eine Zahl n und eine Clusterzerlegung $\{C_1, \ldots, C_n\}$, sodaß eine vorgegebene Zielfunktion unter Restriktionen optimiert wird.

4.3 Hierarchische Clusterverfahren

Definition 4.3.1 Unter einem *hierarchischen Clusteralgorithmus* versteht man ein Verfahren, das für m gegebene Objekte eine Folge von Partitionen der Länge $n_1, \ldots, n_k$ erzeugt, wobei $1 = n_0 < n_1 < \ldots < n_k \leq m$.

Definition 4.3.2 Ein hierarchischer Clusteralgorithmus heißt *divisiv*, wenn aus Clustern der Stufe j solche von Stufe $j+1$ entstehen, indem einer oder mehrere Cluster der Stufe j aufgespalten werden.

Definition 4.3.3 Man nennt einen hierarchischen Clusteralgorithmus *agglomerativ*, wenn der Prozeß durch Schmelzung der Cluster geschieht.

4.4 Algorithmen mit Hilfe von Schwellwerten

Definition 4.4.1 Eine *Proximity-Matrix* (*Entfernungsmatrix*) ist eine symmetrische $(n \times n)$-Matrix $D = (d_{ij})$, wobei die Abstandsfunktion von Definition 4.1.2 stammt. Man unterscheidet den *Ähnlichkeitsindex* (*similarity*) und den *Unähnlichkeitsindex* (*dissimilarity*). Wegen 4.1.2 setzen wir:

a) $d_{ii} = 0$ für alle i (Unähnlichkeitsindex)

b) $d_{ii} \geq \max_k d_{ik}$ (Ähnlichkeitsindex)

Definition 4.4.2 Ein *Schwellwertgraph* (*engl. threshold graph*) ist ein schlichter Graph mit n Ecken. Jede Ecke stellt ein Objekt dar. Ein Schwellwertgraph ist für jede Unähnlichkeitsmatrix definiert.

Algorithmus 4.4.1 Der Algorithmus single-link:

a) Beginne mit dem disjunkten Cluser $C(0)$ und dem Schwellwertgraphen $G(0)$, welcher keine Kanten enthält.

b) Bilde den Schwellwertgraphen $G(k)$. Wenn die Anzahl der maximalen, zusammenhängenden Untergraphen in $G(k)$ kleiner ist, als die Anzahl der Cluster $C(m)$, dann bilde einen neuen Cluster $C(m+1)$.

c) Wenn der Schwellwertgraph $G(k)$ aus einem einzelnen zshg. Graphen besteht, dann Stop. Andernfalls setze $k := k+1$ und gehe nach b).

Definition 4.4.3 Eine Untermenge S eines (schlichten) Graphen G heißt eine *Clique*, wenn der durch S induzierte Untergraph $G(S)$ vollständig ist.

Algorithmus 4.4.2 Der Algorithmus complete-link:

a) wie in Algorithmus 4.4.1.

b) Bilde den Schwellwertgraphen $G(k)$. Wenn zwei Cluster $C(m)$ eine Clique bilden, dann verschmelze die zwei Cluster zu einem und setze $C(m+1)$.

c) Wenn $k = \frac{n(n-1)}{2}$, dann ist $G(k)$ ein vollständiger Graph mit n Ecken, dann Stop. Andernfalls setze $k := k+1$ und gehe nach b).

Bemerkung 4.4.1 Das Klassifikationsergebnis divisiver und agglomerativer Verfahren läßt sich als *Dendrogramm* darstellen, aus dem abgelesen werden kann, welche Cluster geteilt bzw. zusammengeführt wurden.

4.5 Bäume und Cluster

Algorithmus 4.5.1 Der graphentheoretische Algorithmus für single-link:

a) Beginne mit einer disjunkten Clustermenge, d.h. mit jedem Objekt in seinem Cluster. Betrachte einen minimalen aufspannenden Baum (minimal spanning tree (MST)). Wiederhole die Schritte b) und c) bis die gewünschte Anzahl von Cluster erreicht wird.

b) Vereinige zwei Cluster, durch Verbinden der MST-Kante mit dem kleinsten Gewicht.

c) Setze das Gewicht der ausgewählten Kante auf ∞.

Algorithmus 4.5.2 (Zahn) Der Algorithmus von Zahn entdeckt im Sinne der künstlichen Intelligenz und Mustererkennung inkonsistente Kanten:

a) Konstruiere den MST für n Mengen (Muster (pattern)).

b) Indentifiziere die inkonsistenten Kanten in MST.

c) Entferne die inkonsistenten Kanten, welche die zusammenhängenden Komponenten bilden. Diese bilden eine Clusterzerlegung.

4.6 Fuzzy-Mengen

Bemerkung 4.6.1 Sei E eine Grundmenge und $A \subseteq E$. Wir bezeichnen mit dem Symbol $\in$ ein Element x aus der Menge A: $x \in A$. Mit Hilfe der charakteristischen Funktion können wir schreiben:

$$\mu_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

Damit können wir alle mengentheoretischen Operationen beschreiben.

Bemerkung 4.6.2 Wir benutzen die charakteristische Funktion, um *Fuzzy-Mengen* zu beschreiben. fuzzy $\hat{=}$ unscharf, crisp $\hat{=}$ scharf.

Definition 4.6.1 Sei E eine Grundmenge und μ eine Funktion von E in das Einheitsintervall $[0, 1]$. Dann heißt die Menge A aller Paare $(x, \mu(x))$ eine *Fuzzy-Menge über E*. μ heißt *Zugehörigkeitsfunktion* (*membership*) oder *Grad der Zugehörigkeit* (*Grad der Wahrheit*).

Bemerkung 4.6.3 Die Gleichheitsrelation klassischer Mengen A, B ist:

$$A = B \;\Leftrightarrow\; \text{für alle } a \in A \text{ folgt } a \in B \text{ und für alle } b \in B \text{ folgt } b \in A.$$

Dabei ist $\in$ ein Zugehörigkeitsgrad von 1.

Definition 4.6.2 Sei E eine Grundmenge und A, B Fuzzy-Mengen mit den Zugehörigkeitsfunktionen μ_A und μ_b. Dann gilt:

$$A = B \;\Leftrightarrow\; \mu_A(x) = \mu_B(x).$$

Definition 4.6.3 Sei E eine Grundmenge und A, B Fuzzy-Mengen über E mit μ_A und μ_B. Dann gilt:

$$A \subseteq B, \text{ wenn } \mu_A(x) \leq \mu_B(x) \text{ für alle } x \in E.$$

Bemerkung 4.6.4 Sei E eine Grundmenge. Die Negation einer klassischen Menge $A \subseteq E$ ist definiert durch:

$$\bar{A} := \{x \mid x \in E, x \notin A\}.$$

Definition 4.6.4 Die Negation einer Fuzzy-Menge B ist definiert durch:

$$\bar{B} := \{\mu_{\bar{B}} \mid \mu_{\bar{B}}(x) := 1 - \mu_B(x)\}.$$

Definition 4.6.5 Wir definieren die Vereinigung und den Durchschnitt zweier Fuzzy-Mengen A und B punktweise durch:

$$A \vee B := max(\mu_A, \mu_B) \qquad A \wedge B := min(\mu_A, \mu_B).$$

4.7 Fuzzy-Clusterung

Bemerkung 4.7.1 Für Fuzzy-Cluster ist die Zugehörigkeitsfunktion f_{C_k} nicht negativ und summiert sich auf jedem Muster zu $f_{C_k}(x) \geq 0$ oder $f_{C_k}(x) = 1$. Man braucht eine Kriteriumsfunktion φ_f, die den Fuzzy-Grad einer Partition definiert.

Algorithmus 4.7.1 Ein Algorithmus zur Fuzzy-Clusterung:

a) Wähle eine Anfangspartition $\{C_k\}_{k=1}^{K}$. Wiederhole Schritte b) bis d) bis die Clusterzugehörigkeit sich stabilisiert.

b) Berechne die Zugehörigkeitsfunktion $\{f_{C_k}(x_j)\}$.

c) Berechne die Kriteriumsfunktion φ_f.

d) Klassifiziere die Muster (pattern) um die Kriteriumsfunktion φ_f zu verbessern.

4.8 Begriffliche Clusterung

Algorithmus 4.8.1 (Stern-Methode) Die Stern-Methode versucht, Mengen von Beobachtungen gegeneinander abzugrenzen:

a) Wahl von k Ausgangsbeobachtungen.

b) Bestimmung des Sterns für jede Ausgangsbeobachtung gegen die anderen.

c) Konstruktion einer (möglichst) disjunkten Abdeckung.

d) Evaluierung: wenn das Kriterium erfüllt ist, alle Charakterisierungen ausgeben und zur nächsten Ebene übergehen.

5 Planare Graphen

5.1 Grundbegriffe

Bemerkung 5.1.1 Unter einer *Ebene* verstehen wir im folgenden unsere Anschauungsebene, die durch den Vektorraum $\mathbb{R} \times \mathbb{R}$ koordinatisiert werden kann. Unter einem *ebenen Graphen* G verstehen wir einen in die Ebene eingezeichneten Graphen, bei dem sich die Kanten nur in den Ecken schneiden.

Definition 5.1.1 Ein Graph heißt *planar*, wenn G isomorph zu einem ebenen Graphen ist. Die Darstellung von G als ebenen Graph nennt man auch *planare Einbettung*.

Definition 5.1.2 Eine *Jordankurve* J ist eine stetige, sich selbst nicht schneidende Kurve, deren Anfangs- und Endpunkt identisch sind. Die Jordankurve J zerlegt E in zwei disjunkte, offene Teilmengen, die mit $int(J)$ und $ext(J)$ bezeichnet werden. Der topologische Abschluß ist mit $Int(J)$ und $Ext(J)$ bezeichnet.

Satz 5.1.1 (Jordan'scher Kurvensatz) Sei J eine Jordankurve. Jede Kurve, die einen Punkt $a \in int(J)$ mit einem Punkt $b \in ext(J)$ verbindet, schneidet die Jordankurve J.

Satz 5.1.2 Der vollständige Graph K_5, sowie der vollständige bipartite Graph $K_{3,3}$ sind nicht planar.

Definition 5.1.3 Ein Graph G heißt *einbettbar* in einer Fläche S, wenn G isomorph zu einem Graphen H in S ist, dessen Kanten sich nur in den Ecken schneiden.

Bemerkung 5.1.2 So lassen sich beispielsweise der K_5 auf einem Torus und der $K_{3,3}$ auf dem Möbiusband einbetten. Jeder planare Graph läßt sich mit Hilfe der stereographischen Projektion in eine Sphäre (Kugeloberfläche) einbetten und umgekehrt.

5.2 Euler'sche Formel

Algorithmus 5.2.1 Ist ein planarer Graph G gegeben, läßt sich der zu G *duale Graph* $G^\star$ folgendermaßen konstruieren:

a) Jedem Gebiet f von G entspricht eine Ecke $f^\star$ von $G^\star$.
b) Jeder Kante e von G entspricht eine Kante $e^\star$ von $G^\star$.
c) Zwei Ecken $f^\star$ und $g^\star$ sind in $G^\star$ durch eine Kante $e^\star$ genau dann verbunden, wenn die Gebiete f und g von G getrennt waren.

Folgerung 5.2.1 Es gilt $\nu(G^\star) = \varphi(G)$, wobei $\varphi(G)$ die Anzahl der Gebiete von G ist. G und $G^\star$ besitzen die selbe Kantenanzahl, also $\delta(G^\star) = \varepsilon(G)$ und für alle Gebiete f von G, wobei $d_G(f)$ die Anzahl der Kanten ist, mit denen f inzidiert gilt $d_G(f^\star) = d_G(f)$.

Satz 5.2.1 Ist G ein planarer Graph, dann gilt:

$$\sum_{f \in F} d(f) = 2\varepsilon,$$

wobei F die Menge der Gebiete von G bezeichnet.

Satz 5.2.2 Ist G ein zusammenhängender, planarer Graph, dann gilt:

$$\nu - \varepsilon + \varphi = 2,$$

wobei φ die Anzahl der Gebiete von G bezeichnet.

Bemerkung 5.2.1 Die Formel aus Satz 5.2.2 hängt mit der Euler'schen Polyederformel $\nu - \varepsilon + \varphi = 2$ zusammen.

Korollar 5.2.1 Ist G ein schlichter, planarer Graph mit mindestens 3 Ecken, dann gilt:

$$\varepsilon \leq 3 \cdot \nu - 6.$$

Korollar 5.2.2 Ist G schlicht und planar, so gilt für den kleinsten Eckengrad δ von G, daß $\delta \leq 5$.

Satz 5.2.3 Der vollständige Graph K_5 und der vollständige bipartite Graph $K_{3,3}$ sind nicht planar.

Definition 5.2.1 Wird in einem Graphen G eine Kante uv durch zwei Kanten ua und av ersetzt, wobei die Ecke a zu G neu hinzugefügt wird, dann heißt der dadurch neu entstandene Graph $G^\star$ eine *Unterteilung*. Ferner heißt jeder Graph $G^\star$ eine Unterteilung von G, der durch endlich viele Unterteilungen aus G entstanden ist.

Satz 5.2.4 (Kurakowski) Ein Graph G ist genau dann planar, wenn G keine Unterteilung des K_5 und des $K_{3,3}$ enthält.

6 Euler'sche Graphen

6.1 Grundbegriffe

Definition 6.1.1 Eine *Eulertour* von G ist eine geschlossene Linie, die jede Kante von G enthält. Ein Graph G heißt *euler'sch*, wenn G eine Eulertour besitzt.

Satz 6.1.1 Ein nichtleerer, zusammenhängender Graph G ist genau dann euler'sch, wenn G keine Ecken von ungeradem Eckengrad hat.

6.2 Das chinesische Briefträgerproblem

Problem 6.2.1 Wie kann man eine Route finden, die alle Straßen eines Bezirks enthält und von minimaler Länge ist.

Algorithmus 6.2.1 (Fleury) Der Algorithmus von Fleury besteht aus folgenden Schritten:

a) Wähle eine Ecke v_0 und setze $L_0 := v_0$.
b) Sei die Linie $L_i = v_0 e_1 v_1 \dots e_i v_i$ bereits gefunden. Dann wähle eine Kante e_{i+1} aus $E \setminus \{e_1, \dots, e_i\}$, sodaß:
   i) e_{i+1} inzidiert mit v_i.
   ii) e_{i+1} ist keine Schnittkante von $G_i = G \setminus \{e_1, \dots, e_i\}$ außer, es gibt keine Alternative.
c) Stop, wenn Schritt b) nicht weiter durchgeführt werden kann.

Bemerkung 6.2.1 Der Algorithmus von Fleury konstruiert eine Linie. Wenn der Graph G euler'sch ist, konstruiert er eine Eulertour. In diesem Fall löst er das Problem. Wenn G nicht euler'sch ist, muß man einen neuen Graphen$D$ betrachten, bei dem die Kanten doppelt sind.

Bemerkung 6.2.2 Der Algorithmus von Fleury ist polynomial. Es gibt ähnliche polynomiale Algorithmen für nicht euler'sche Graphen.

6.3 Konstruktion einer Computertrommel

Problem 6.3.1 Kann man eine Trommel so konstruieren, daß 2^n verschiedene Positionen der Trommel durch k Kontakte bestimmt werden können ?

Bemerkung 6.3.1 Man sieht, daß k Kontakte ein k-Tupel aus $\{0, 1\}$ abgreifen. Es gibt 2^k derartige Tupel. Wenn also 2^n Positionen durch verschiedene k-Tupel bestimmt werden sollen, dann muß gelten $2^k \geq 2^n$, d.h. $k \geq n$.

Bemerkung 6.3.2 Definiere einen Digraphen D_n folgendermaßen. Die Ecken von D_n sind $(n-1)$-Tupel $p_1 p_2 \dots p_{n-1}$ mit $p_i \in \{0, 1\}$. Es gibt einen Bogen mit Anfang $p_1 \dots p_{n-1}$ und Ende $q_1 \dots q_{n-1}$ genau dann, wenn $p_{i+1} = q_i$, $i = 1, \dots, n-2$. Ferner werden die Bögen mit den Gewichten $p_1 \dots p_{n-1} q_{n-1}$ versehen.

Bemerkung 6.3.3 D_n ist nach Konstruktion zusammenhängend, jede Ecke hat den Eingangsgrad 2 und den Ausgangsgrad 2. Man kann zeigen, daß D_n eine Eulertour besitzt und sich $k = n$ ergibt.

7 Hamilton'sche Graphen

7.1 Theorie der Hamilton'schen Graphen

Definition 7.1.1 Ein Weg, der jede Ecke eines Graphen G enthält, heißt *Hamiltonweg*. Ein *Hamiltonzykel* ist ein Zykel, der alle Ecken von G enthält. Ein Graph heißt *hamilton'sch*, wenn G einen Hamiltonzykel enthält.

Bemerkung 7.1.1 Eine Charakterisierung der hamilton'schen Graphen durch Eckengrade o.ä. ist bis jetzt noch nicht gelungen.

Satz 7.1.1 Ist G hamilton'sch, dann gilt für jede nichtleere, echte Untermenge $S \subset V$ für die Anzahl der Zusammenhangskomponenten $\omega(G - S) \leq |S|$.

Satz 7.1.2 Ist G schlicht, $\nu \geq 3$ und $\delta \geq \frac{\nu}{2}$, dann ist G hamilton'sch.

Satz 7.1.3 Sei G ein schlichter Graph und seien u, v zwei nicht benachbarte Ecken von G mit $d(u) + d(v) \geq \nu$. G ist genau dann hamilton'sch, wenn $G + uv$ hamilton'sch.

Bemerkung 7.1.2 Es gibt kein Kriterium für das Hamilton'schsein in Eckengraden etc. (im Vergleich zu euler'schen Graphen).

7.2 Travelling Salesman Problem (TSP)

Bemerkung 7.2.1 Ein Handelsvertreter wünscht, eine Folge von Städten zu besuchen und danach wieder zu seinem Ausgangspunkt zurückzukehren. Wenn die Entfernungen zwischen den Städten gegeben sind, soll ein Reiseweg gesucht werden, der jede Stadt genau einmal enthält und eine minimale Gesamtlänge hat.

Problem 7.2.1 (Graphentheoretische Version) Gegeben ist ein gewichteter, vollständiger Graph. Gesucht ist ein Hamiltonzykel von kleinstem Gewicht.

Bemerkung 7.2.2 Es ist bisher ein ungelöstes Problem, ob es einen Algorithmus polynomialer Komplexität gibt, der einen optimalen Zykel findet.

Bezeichnung 7.2.1 Ein Algorithmus heißt *heuristisch*, wenn er nicht notwenigerweise eine formale Garantie der Güte der gefundenen Lösung bietet. Ein Algorithmus heißt *Näherungsverfahren*, wenn sich die Güte der Lösung abschätzen läßt.

Bezeichnung 7.2.2 Sei P ein Optimierungsproblem mit einer Kostenfunktion C. Sei I ein Beispiel des Problems P. Sei A ein Algorithmus, der für jedes Beispiel I eine zulässige Lösung $f_A(I)$ liefert. Dann bezeichnen wir eine optimale Lösung mit $\hat{f}(I)$.

Definition 7.2.1 Ein Algorithmus A heißt ein ε-Näherungsverfahren für P, wenn es ein $\varepsilon \geq 0$ gibt, sodaß:

$$\frac{|cf_A(I) - c\hat{f}(I)|}{c\hat{f}(I)} \leq \varepsilon \quad \text{für alle Beispiele } I.$$

Definition 7.2.2 Wenn die Dreiecksungleichung für die Entfernungen des TSP gilt, dann bezeichnen wir es als Δ-TSP.

Algorithmus 7.2.1 Der Algorithmus „Baum“ für das Δ-TSP besteht aus folgenden Schritten:

a) Finde einen aufspannenden Baum T minimalen Gewichts für den Graphen mit der Entfernungsmatrix (d_{ij}).

b) Verdopple die Kanten von T und erhalte den Multigraphen $K(T)$.

c) Finde einen Eulerweg in $K(T)$ und bestimme daraus eine Hamiltontour.

Satz 7.2.1 Der oben beschriebene Algorithmus „Baum“ ist ein 1-Näherungsverfahren für das Δ-TSP.

7.3 Komplexität

Satz 7.3.1 Unter der Voraussetzung, daß P $\neq$ NP gilt, gibt es für kein $\varepsilon > 0$ ein ε-Näherungsverfahren für das (allgemeine) TSP.

7.4 Das TSP und die Methoden der künstlichen Intelligenz

Algorithmus 7.4.1 Der Zykelverbesserungsalgorithmus besteht aus folgenden Schritten:

a) Sei ein Hamiltonzykel $C = v_1 \dots v_\nu v_1$ gegeben.

b) Durch folgende Modifikation wird C verbessert:
Für alle i, j mit $1 < i + 1 < j < \nu$ erhält man einen neuen Zykel $C_{ij} = v_1 v_2 \dots v_i v_j v_{j-1} \dots v_{i+1} v_{j+1} \dots v_\nu v_1$, wenn nun für i, j gilt: $\omega(v_i v_j) + \omega(v_{i+1} v_{j+1}) < \omega(v_i v_{i+1}) + \omega(v_j v_{j+1})$.

c) Stop nach einer vorgegebenen Anzahl von Durchläufen.

8 Neuronale Netze und das TSP

8.1 Struktur eines neuronalen Netzes

Definition 8.1.1 Seien $e_1, \ldots, e_n$ die Eingangsgrößen, $e_i \in \{0,1\}$ und seien $w_1, \ldots, w_n$ Gewichte (Synapsenwerte), $w_i \in \mathbb{R}^+$ eines Neurons.
Sei ferner $net := \sum w_i e_i$ der kumulierte Input. Dann ist $a := f(net - \theta)$ der Ausgabewert des Neurons, wobei

$$f(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x \leq 0. \end{cases}$$

θ heißt *Schwellwert*, f heißt *Transferfunktion*. Ein Zusatzeingang steht stets auf 1, hat das Gewicht $-\theta$ und heißt *Bias* (*Schiefe*).

Bemerkung 8.1.1 Bei Definition 8.1.1 dürfen auch andere Transferfunktionen verwendet werden.

Definition 8.1.2 Ein *einschichtiges* neuronales Netz ist durch eine $n \times m$-Matrix M und eine Transferfunktion f gegeben, sodaß jedem Inputvektor e ein Outputvektor a zugeordnet wird gemäß:

$$a = f(net - \theta) \quad \text{mit } net = M_e.$$

8.2 Lernen

Definition 8.2.1 (Deltaregel) Sei Δa_j die Differenz des gewünschten Outputs und dem zur Zeit des Lernens wirklich erzielten Outputs. Dann wird das Gewicht w_{ij} vom Input e_i zum Output a_j folgendermaßen geändert:

$$\Delta w_{ij} = \sigma \cdot l_i \cdot \Delta a_j \qquad \sigma \text{ beliebige Konstante.}$$

8.3 Netzwerktopologie

Definition 8.3.1 Sei e Eingabevektor, a Ausgabevektor und seien $h_1, h_2, \ldots$ Hilfsvektoren. Sei f eine Transferfunktion und $W_1, W_2, \ldots$ Matrizen. Dann errechnet ein n-stelliges neuronales Netz aus e den Vektor a durch:

$$\begin{aligned} h_1 &= f(W_1 e) \\ h_2 &= f(W_2 h_1) \\ \vdots &\quad \vdots \\ a &= f(W_n h_{n-1}) \end{aligned}$$

8.4 Das Perzeptron

Problem 8.4.1 Aufgabe ist es, die Gewichte so zu bestimmen, daß die Fehler minimal werden:

$$E = \frac{1}{2} \sum_i (z_i - a_i)^2.$$

Satz 8.4.1 Seien $W_1 = (w_{ij}^1)$ und $W_2 = (w_{ij}^2)$ die Matrizen für ein 2-stufiges Perzeptron. Ein Lernschritt bestehe darin, daß die Gewichte geändert werden durch:

a) $\Delta w_{ij}^1 = \alpha \cdot \varepsilon_i \cdot h$

b) $\Delta w_{ij}^2 = \alpha \sum_{m=1}^{n} e_m \cdot w_{mi}^1 \cdot e_j$.

Dann wird die Fehlerfunktion

$$E = \frac{1}{2} \sum_{i=1}^{n} (z_i - a_i)^2$$

verkleinert und ein lokales Minimum angestrebt.

Algorithmus 8.4.1 Der Lernalgorithmus bei Backpropagation besteht aus folgenden Schritten:

a) Bestimme die Gewichte mit Zufallszahlen.

b) Gebe ein zufälliges Ein- und Ausgabemuster der zu lernenden Funktion vor, berechne die Belegung h_j der versteckten Schicht.

c) Für die so vorgegebenen Eingabewerte e_j und die Zielwerte z_i korregiere Gewichte entsprechend der beiden Korrekturformeln aus Satz 8.4.1.

d) Fahre bei b) fort.

8.5 Hopfield-Netz

Bemerkung 8.5.1 Die bisher betrachteten Netze waren alle ohne Rückkoppelung. Beim *Hopfield-Netz* sind alle n Neuronen miteinander verbunden.

Definition 8.5.1 Die Gewichte w_{ij} ergeben sich nach Neuron i und Neuron j mit den Eigenschaften:

a) $w_{ii} = 0$ für alle i

b) $w_{ij} = w_{ji}$ für alle i, j.

Beim Hopfield-Netz wählt man oft $\{+1, -1\}$ statt $\{0, 1\}$. Belegt man *alle* Neuronen des Netzes mit Eingabewerten e_j, dann berechnet das Netz:

c) $a_j = f(\sum_j w_{ij} \cdot e_j - \delta_i)$, wobei δ_i sind Schwellwerte und einer Transferfunktion f.

Nun legt man die berechneten Werte a_j wieder ans Netz und bekommt Iterationen. Konvergiert das Iterationsverfahren, betrachtet man die Grenzwerte, die keine Veränderungen bewirken als Ausgaben.

d) $a^{(p+1)} = f(W \cdot a^{(p)} - \delta)$
$a^{(0)} = e$.

Definition 8.5.2 Ein Hopfield-Netz ist gegeben durch eine symmetrische Matrix W, deren Diagonalelemente 0 sind, einen Schwellwertvektor δ und die Signumfunktion als Transferfunktion. Ein Vektor a heißt ein *Muster* für das Hopfield-Netz, wenn gilt:

$$a = f(W \cdot a - \delta).$$

Definition 8.5.3 Gegeben sei ein Hopfield-Netz mit den Gewichten w_{ij} und den Schwellwerten δ_i. Dann heißt die Funktion

$$H = -\frac{1}{2} \sum_i \sum_j w_{ij} \cdot a_i \cdot a_j \; + \; \sum_k \delta_k \cdot a_k$$

Energie- oder Hamiltonfunktion.

Bemerkung 8.5.2 Man berechnet die Hamiltonfunktion für alle möglichen Eingabewerte und erhält eine *Energielandschaft* mit Minima und Maxima.

Satz 8.5.1 Die Muster eines Hopfield-Netzes sind die Minima der Energiefunktion. Falls nur strikte Minima existieren, dann konvergiert die Iterationsfolge

$$\begin{aligned} a^{(p+1)} &= f(W \cdot a^{(p)} - \delta) \\ a^{(0)} &= e. \end{aligned}$$

8.6 Simulated Annealing (Simulierte Kühlung)

Bemerkung 8.6.1 Die Hopfield-Iteration wird so umgebaut, daß ein Algorithmus für das Finden eines globalen Minimums entsteht. Wir benutzen:

$$y(r) = \frac{1}{1 + e^{\frac{-r}{T}}},$$

wobei T (die *Temperatur*) ein beliebiger Parameter ist. Wählt man t sehr groß, können alle Zustände des Netzes angenommen werden. Das Verfahren bleibt nicht mehr in einem lokalen Minimum hängen, wobei es häufig zur Verschlechterung der Zielfunktion kommt. Man beginnt mit einem großen T und läßt T langsam kleiner werden (*abkühlen*).

Algorithmus 8.6.1 Der Algorithmus für das Simulated Annealing besteht aus folgenden Schritten:

a) Wähle ein zufälliges Neuron m aus.
b) Berechne $r = \sum_j w_{mj} \cdot a_j - \delta_m$.
c) Berechne $p = \frac{1}{1+e^{\frac{-r}{T}}}$.
d) Wähle eine Zufallszahl $0 \leq z \leq 1$ und setze $a_m := \begin{cases} 1 & \text{falls } z \leq p \\ 0 & \text{falls } z > p. \end{cases}$
e) Gehe zu Schritt a).

8.7 Illustration für das TSP

Bemerkung 8.7.1 Die Neuronenaktivität beschreiben wir durch $a(i,j)$, wobei i die Zeile und j die Spalte bezeichnet. Beim TSP ergeben sich die Restriktionen aus der Tatsache, daß jede Zeile und jede Spalte genau ein Neuron aktiv sein muß. Es muß also gelten:

$$S_1 = \sum_{\substack{i,j,k \\ j \neq k}} a(i,j) \cdot a(j,k) = 0$$

$$S_2 = \sum_{\substack{i,j,k \\ i \neq j}} a(i,k) \cdot a(j,k) = 0.$$

Weiterhin müssen genau N Neuronen aktiv sein:

$$S_3 = \left(\sum_{i,j} a(i,j) - N \right)^2 .$$

Bezeichne ferner $d(s,t)$ die Entfernungen der Städte s und t. Die Tour ergibt sich in ihrer Länge zu:

$$S = \sum_{i,s,t} d(s,t) \cdot a(i,s) \cdot a(i+1,t).$$

Insgesamt hat man zu minimieren:

$$S = aS_1 + bS_2 + cS_3 + dS_4 \quad \text{für } a,b,c,d > 0.$$

Nun setzt man die Gewichte des Netzes an, daß S der Energie des Netzes entspricht, also:

$$S = -\frac{1}{2} \sum_i \sum_j w_{ij} \cdot a_i \cdot a_j \;-\; \sum_i \delta_i \cdot a_i \quad \text{mit } i = k + l.$$

Dann läßt man das Netz die optimale Tour suchen.

9 Matchings

9.1 Grundbegriffe

Definition 9.1.1 Eine Untermenge M der Kantenmenge E eines Graphen G heißt *Matching* (*Paarung*), wenn jede Kante aus M zwei verschiedene Ecken hat und je zwei Kanten nicht benachbart sind, d.h. sie haben keine gemeinsame Ecke.

Definition 9.1.2 Ein Matching M *sättigt* eine Ecke v (v heißt *M-gesättigt*), wenn eine Kante aus M mit v inzidiert. Ein Matching heißt *perfekt*, wenn jede Ecke von G M-gesättigt ist.

Definition 9.1.3 M ist ein *maximales* Matching, wenn es kein Matching $M^\star$ gibt mit $|M^\star| > |M|$.

Definition 9.1.4 Sei M ein Matching von G. Ein *M-alternierender Weg W* ist ein Weg in G, dessen Kanten abwechselnd in M und in $E \setminus M$ sind. Ein *M-Verbesserungsweg* ist ein M-alternierender Weg, dessen Anfangs- und Endecke M-ungesättigt ist.

Satz 9.1.1 Ein Matching M von G ist genau dann maximal, wenn G keinen M-Verbesserungsweg enthält.

9.2 Heiratssatz

Bezeichnung 9.2.1 Sei $S \subseteq V$ eine Menge von Ecken in G. Die *Nachbarschaftsmenge* $N(S)$ von S besteht aus allen Ecken, die zu den Ecken von S benachbart sind.

Satz 9.2.1 (Hall) Sei G ein bipartiter Graph mit einer Bipartition (X, Y). G besitzt genau dann ein Matching, das jede Ecke von X sättigt, wenn gilt:

$$|N(S)| \geq |S| \quad \text{für alle } S \subseteq X.$$

Definition 9.2.1 G heißt k-regulär, wenn jede Ecke von G den Eckengrad k hat.

Satz 9.2.2 (Heiratssatz) Ist G ein k-regulärer bipartiter Graph mit $k > 0$, dann besitzt G ein perfektes Matching.

9.3 Ungarische Methode

Problem 9.3.1 In einer Fabrik gibt es n Arbeiter $x_1, \ldots, x_n$, die für n Arbeiten $y_1, \ldots, y_n$ eingesetzt werden können, wobei jeder Arbeiter für eine oder mehrere Arbeiten geeignet ist. Kann man die Arbeiter so einsetzen, daß jeder eine Arbeit bekommt, für die er qualifiziert ist.

Problem 9.3.2 (Mathematisierung) Gegeben ist ein Graph G mit einer Bipartition (X, Y), wobei $X = \{x_1, \ldots, x_n\}$ und $Y = \{y_1, \ldots, y_n\}$. Eine Ecke x_i ist mit einer Ecke y_j genau dann verbunden, wenn der Arbeiter x_i für die Arbeit y_j qualifiziert ist.

Algorithmus 9.3.1 Der Algorithmus der Ungarischen Methode besteht aus folgenden Schritten:

a) Beginne mit einem beliebigen Matching M (evtl. mit einer Kante).
b) Sättigt M alle Ecken in X, dann Stop.
Andernfalls wähle eine M-ungesättigte Ecke $u \in X$.
c) Suche einen M-Verbesserungsweg P mit Anfang u. Existiert kein solches P, dann Stop: Es gibt kein perfektes Matching.
Andernfalls ergibt $\Delta(M, E(P))$ ein vergrößertes Matching $M^\star$. Setze $M := M^\star$ und gehe zu b).

Bezeichnung 9.3.1 In Schritt b) wird ein Baum entwickelt, der auch *ungarischer Baum* heißt.

Definition 9.3.1 Ein ungarischer Baum von u ist ein Baum H mit:

a) $u \in V(H)$
b) für jede Ecke v von H ist der (u, v)-Weg ein M-alternierender Weg.

Bei dieser Prozedur läßt man den Baum von der Wurzel ausgehend wachsen.

9.4 Optimale Zuordnung

Problem 9.4.1 Betrachte einen gewichteten, vollständigen, bipartiten Graphen mit Bipartition (X, Y) mit $X = \{x_1, \ldots, x_n\}$ und $Y = \{y_1, \ldots, y_n\}$ und die Matrix (w_{ij}), wobei w_{ij} das Gewicht der Kante $x_i y_j$ ist. Finde ein perfektes Matching maximalen Gewichts als *optimales Matching.*

Bemerkung 9.4.1 Es sind mindestens drei Methoden zur Lösung des Problems bekannt:

a) Lösung des sog. *Zuordnungsproblems* in der linearen Optimierung mit Hilfe des *Simplex-Verfahrens* führt schnell zu einer Lösung:

$$max \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

$$\text{Restriktionen}: \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n$$

$$x_{ij} = \begin{cases} 1 & \text{falls der Arbeiter } i \text{ den Job } j \text{ bekommt} \\ 0 & \text{sonst} \end{cases}$$

b) Der Algorithmus von KUHN-MUNKRES, der die Idee einer *Eckenmarkierung* verfolgt, was im folgenden behandelt wird.

c) Die Betrachtung teilweise geordneter Mengen durch Hinzunahme von Präferenzen.

Bezeichnung 9.4.1 Die Abbildung $l : X \cup Y \to \mathbb{R}^+$ heißt eine *zulässige Eckenmarkierung*, wenn für alle $x \in X$ und alle $y \in Y$ gilt:

$$l(x) + l(y) \geq w(x, y).$$

Bemerkung 9.4.2 Es gibt stets eine zulässige Eckenmarkierung, nämlich:

$$l(x) = \max_{y \in Y} w(x, y) \quad \text{und} \quad l(y) = 0.$$

Bezeichnung 9.4.2 Ist l eine zulässige Eckenmarkierung, dann bezeichnet E_l die Menge aller Kanten xy für die $l(x) + l(y) = w(x, y)$ gilt, also ist:

$$E_l := \{xy \mid l(x) + l(y) = w(x, y)\}.$$

Der durch E_l aufgespannte Untergraph heißt *Gleichheitsgraph* G_l, der zur Markierung l gehört.

Satz 9.4.1 Sei eine zulässige Eckenmarkierung von G gegeben. Enthält G_l ein perfektes Matching $M^\star$, dann ist $M^\star$ optimal für G.

Algorithmus 9.4.1 Der Algorithmus von KUHN-MUNKRES besteht aus folgenden Schritten:

a) Beginne mit einer zulässigen Eckenmarkierung. Bestimme G_l und wähle ein Matching M in G.

b) Ist jede Ecke in G_l M-gesättigt, dann ist M ein perfektes Matching wegen $|X| = |Y|$ und nach obigem Satz ein optimales Matching für G, also Stop.
Ansonsten sei u eine M-ungesättigte Ecke. Setze $S = \{u\}$ und $T = \emptyset$.

c) Ist $N(S) \supset T$, gehe nach d).
Ansonsten gilt $N(S) = T$. Berechne $\alpha_l = \min_{\substack{x \in S \\ y \notin T}} \{l(x) + l(y) = w(x, y)\}$
und erhalte $\hat{l}$ als zulässige Eckenmarkierung mit:

$$\hat{l}(v) = \begin{cases} l(v) - \alpha_l & \text{falls } v \in S \\ l(v) + \alpha_l & \text{falls } v \in T \\ l(v) & \text{sonst} \end{cases}$$

d) Wähle ein $y \in N(S) \setminus T$. Entscheide wie bei der ungarischen Methode, ob y M-gesättigt oder M-ungesättigt ist. Ist y M-gesättigt mit $xy \in M$, ersetze S durch $S \cup \{x\}$ und T durch $T \cup \{y\}$ und gehe zu c).
Andernfalls haben wir einen M-Verbesserungsweg P in G_l.
Bilde $\hat{M} := \Delta(M, E(P))$ und gehe mit $M := \hat{M}$ zu b).

9.5 Überdeckung

Definition 9.5.1 Eine *Überdeckung* eines Graphen G ist eine Untermenge K von Ecken, sodaß jede Kante von G mit mindestens einer Ecke von K inzident ist. Eine Überdeckung K heißt *minimal*, wenn es keine Überdeckung K' gibt, sodaß $|K'| < |K|$.

Bemerkung 9.5.1 Ist K eine Überdeckung von G und M ein Matching, dann enthält K wenigstens jeweils eine Endecke der Kanten von M. Daher gilt $|M| \leq |K|$. Dies gilt sogar für ein maximales Matching $M^\star$ und eine minimale Überdeckung $\bar{K}$.

Satz 9.5.1 Sei M ein Matching und K eine Überdeckung, sodaß $|M| = |K|$. Dann ist M ein maximales Matching und K eine minimale Überdeckung.

Satz 9.5.2 (König, 1931) In einem bipartiten Graphen ist die Anzahl der Kanten eines maximalen Matchings gleich der Anzahl der Ecken einer minimalen Überdeckung.

10 Eckenfärbungen

10.1 Die chromatische Zahl

Problem 10.1.1 (Vierfarbenproblem) Kann man eine Landkarte mit vier Farben so färben, daß benachbarte Länder verschiedene Farben tragen ?
Mit anderen Worten: Kann man die Ecken eines planaren Graphen so färben, daß benachbarte Ecken verschiedene Farben tragen ?

Definition 10.1.1 Eine k-*Eckenfärbung* eines Graphen G ist eine Zuordnung $f : V(G) \to \{1, \ldots, k\}$ der Ecken von G auf k Farben derart, daß benachbarte Ecken verschiedene Farben tragen. Ein Graph G heißt k-*färbbar*, wenn G eine k-Eckenfärbung besitzt. Die *chromatische Zahl* $\chi(G)$ ist die kleinste Zahl k für die G k-färbbar ist. G heißt auch k-*chromatisch*.

Bemerkung 10.1.1 G ist genau dann 1-färbbar, wenn für die Kantenmenge $E(G) = \emptyset$ gilt. G ist genau dann 2-färbbar, wenn G bipartit ist.

Definition 10.1.2 G heißt ein kritischer Graph, wenn für alle echten Untergraphen H gilt:

$$\chi(H) < \chi(G).$$

Bemerkung 10.1.2 Sei G k-färbbar. Wir fassen die gleichgefärbten Ecken zu Äquivalenzklassen $V_1, \ldots, V_k$ zusammen und sprechen von der k-Färbung $(V_1, \ldots, V_k)$.

Satz 10.1.1 Ist G k-kritisch, dann gilt für den kleinsten Eckengrad $\delta \geq k - 1$.

Korollar 10.1.1 Jeder k-chromatische Graph G hat mindestens k Ecken vom Eckengrad $\delta \geq k - 1$.

Korollar 10.1.2 Für jeden Graphen G mit dem größten Eckengrad Δ gilt: $\chi(G) \leq \Delta + 1$.

10.2 Der Fünffarbensatz

Satz 10.2.1 Jeder planare Graph ist 5-färbbar.

10.3 Eine Heuristik für Eckenfärbungen

Problem 10.3.1 Es soll ein Stundenplan für alle Vorlesungen erstellt werden. Dabei dürfen zwei Vorlesungen nicht zur gleichen Zeit stattfinden, wenn es eine Studentengruppe gibt, die beide Vorlesungen besuchen möchte.
Mit anderen Worten: Ein Graph wird konstruiert, dessen Ecken dem verschiedenen Vorlesungen entsprechen. Dürfen zwei Vorlesungen nicht gleichzeitg stattfinden, werde die jeweiligen Ecken durch Kanten verbunden. Die Eckenfärbung des Graphen ist dann eine Lösung des Problems.

Bemerkung 10.3.1 Für die Eckenfärbung ist kein Algorithmus polynomialer Komplexität bekannt.

Algorithmus 10.3.1 Der Greedy-Algorithmus zur Eckenfärbung besteht aus folgenden Schritten:

a) Ordne die Ecken des Graphen $u_1, \ldots, u_n$.

b) Färbe die Ecke u_1 mit der Farbe v_1.

c) Seien die Ecken $u_1, \ldots, u_{i-1}$ schon gefärbt. Ist u_i mit keiner Ecke verbunden, dann färbe u_i mit v_1. Ist u_i mit Ecken der Farben $v_1, \ldots, v_r$ verbunden, färbe u_i mit der Farbe v_j, wobei j die kleinste Zahl ist, die bei $v_1, \ldots, v_r$ nicht vorkommt.

d) Stop.

10.4 Chromatische Polynome

Satz 10.4.1 Ist G schlicht, dann ist

$$\pi_k(G) = \pi_k(G - e) - \pi_k(G \cdot e).$$

Korollar 10.4.1 Für jeden Graphen G ist $\pi_k(G)$ ein Polynom in k vom Grad ν mit ganzzahligen Koeffizienten beginnend mit dem Term k^ν beginnend und endend mit konstantem Term 0. Die Koeffizienten von $\pi_k(G)$ haben verschiedene Vorzeichen.

Definition 10.4.1 $\pi_k(G)$ heißt das *chromatische Polynom.*

10.5 Vergleichbarkeitsgraph

Definition 10.5.1 Sei $(P, \leq)$ eine teilweise geordnete Menge (poset). Der *Vergleichbarkeitsgraph* G von $(P, \leq)$ hat die Elemente von P als Ecken. Zwei Ecken x, y sind genau dann mit einer Kante verbunden, wenn $x \leq y$ oder $y \leq x$ ist.

Bezeichnung 10.5.1 Eine *Clique* ist ein vollständiger Untergraph. Die Zahl $w(G)$ ist die Anzahl einer maximalen Clique. Die Zahl $\theta(G)$ ist die Anzahl der Cliquen, die bei einer maximalen Zerlegung auftreten. Die Zahl $\alpha(G)$ ist die Anzahl der Ecken einer maximalen, unabhängigen Menge.

Lemma 10.5.1 Für jeden Graphen G gilt:

a) $\chi(G) \geq w(G)$

b) $\alpha(G) \leq \theta(G)$

c) $\alpha(G) = w(\bar{G})$

d) $\theta(G) = \chi(\bar{G})$,

wobei $\bar{G}$ der komplementäre Graph zu G ist.

Definition 10.5.2 Ein Graph G heißt *perfekt*, wenn jeder kanteninduzierte Untergraph H die Bedingung erfüllt:

$$\alpha(H) = \theta(H).$$

Satz 10.5.1 (Berge) G ist genau dann perfekt, wenn für alle kanteninduzierten Untergraphen gilt:

$$w(H) = \chi(H).$$

Satz 10.5.2 (Berge) G ist genau dann perfekt, wenn in G und $\bar{G}$ jeder ungerade Zykel der Länge ≤ 5 eine Sehne besitzt.

11 Kantenfärbung

11.1 Kantenchromatische Zahl

Definition 11.1.1 Eine Farbverteilung für die Kanten eines Graphen ist ein Zuordnung $f : E(G) \to \{1, \ldots, k\}$ auf k Farben.

Definition 11.1.2 Eine k-Kantenfärbung von G ist eine k-Farbverteilung für die Kanten, sodaß benachbarte Kanten verschiedene Farben tragen. Dabei heißen zwei Kanten benachbart, wenn sie eine Ecke gemeinsam haben. Die *kantenchromatische Zahl* $\chi^\star(G)$ ist die kleinste Zahl k für die G k-kantenfärbbar ist. G heißt *k-kantenchromatisch*, wenn $\chi^\star(G) = k$.

Bemerkung 11.1.1 Bei jeder Kantenfärbung müssen alle Kanten, die in einer Ecke inzident sind, verschiedene Farben tragen. Es gilt also:

$$\chi^\star(G) \geq \Delta(G),$$

wobei $\Delta(G)$ der größte Eckengrad von G ist.

Bemerkung 11.1.2 Eine k-Kantenfärbung kann man als Partition der Kantenmenge $E(G)$ ansehen $(E_1, \ldots, E_k)$. Dabei ist E_i die Menge aller Kanten der Farbe i, $i = 1, \ldots, k$. Da die Kanten gleicher Farbe nicht benachbart sind, stellt jede Menge E_i ein Matching dar.

Satz 11.1.1 Der vollständige bipartite Graph $K_{m,n}$ mit $m \geq n$ hat die kantenchromatische Zahl $\chi^\star(K_{m,n}) = \Delta$.

Satz 11.1.2 Ein bipartiter Graph hat die kantenchromatische Zahl $\chi^\star = \Delta$.

Bemerkung 11.1.3 Für schlichte Graphen zeigt Satz 11.1.1, daß es eine große Klasse mit $\chi^\star = \Delta$ gibt. Der schlichte vollständige Graph K_3 hat wie jeder ungerade Zykel die kantenchromatische Zahl $\chi^\star = \Delta + 1$.

Satz 11.1.3 (Vizing, 1964) Ist G schlicht, so ist entweder $\chi^\star(G) = \Delta$ oder $\chi^\star(G) = \Delta + 1$.

11.2 Ramseyzahlen

Definition 11.2.1 Eine Untermenge $U \subseteq V(G)$ heißt *unabhängig*, wenn keine zwei verschiedenen Ecken in U benachbart sind. Eine unabhängige Menge heißt *maximal*, wenn es keine unabhängige Menge $U^\star$ gibt mit $|U^\star| > |U|$.

Definition 11.2.2 Eine Untermenge S heißt *Clique*, wenn der durch S induzierte Untergraph $G(S)$ vollständig ist.

Bemerkung 11.2.1 S ist genau dann eine Clique, wenn S eine unabhängige Menge des komplementären Graphen $C(G)$ ist.

Bemerkung 11.2.2 Ramsey zeigte, daß eine kleinste Zahl existiert, sodaß jeder Graph mit $r(k,l)$ Ecken entweder eine Clique mit k Ecken oder eine unabhängige Menge mit l Ecken enthält. Dabei gilt:

a) $r(1,l) = r(k,1) = 1$

b) $r(2,l) = l$

c) $r(k,2) = k$

Satz 11.2.1 Ist $k \geq 2$ und $l \geq 2$, so gilt:

$$r(k,l) \leq r(k,l-1) + r(k-1,l).$$

Korollar 11.2.1 Sei $k \geq 2$, $l \geq 2$ und seinen $r(k,l-1)$ und $r(k-1,l)$ gerade Zahlen, dann gilt:

$$r(k,l) < r(k,l-1) + r(k-1,l).$$

Satz 11.2.2 (Erdös, 1943) Es gilt $r(k,k) \geq 2^{\frac{k}{2}}$.