Teil VI

Graphen und Optimierung

21

Graphen

Die Graphentheorie ist ein wichtiges Instrument, um komplexe Probleme in unterschiedlichen Wissenschaftsbereichen zu bearbeiten. Der interdisziplinäre Charakter der Graphentheorie rührt von der Struktur von Graphen her. Graphentheorie ist immer dann anwendbar, wenn ein Problem zu untersuchen ist, in dem Paare von Objekten in Relation stehen. Beispiele hierfür sind Wegenetze, elektrische Netzwerke und Flussdiagramme. In diesem Kapitel werden die Grundbegriffe der Graphentheorie erörtert.

21.1 Grundbegriffe

Ein *Graph* ist ein Paar $G = (V, E)$, bestehend aus einer nichtleeren Menge V und einer Menge E von 2-Teilmengen von V. Die Elemente von V heißen *Knoten* und die Elemente von E *Kanten*. Eine Kante $e = \{u, v\}$ wird auch als ein Wort $e = uv$ (oder $e = vu$) geschrieben. Liegt eine Kante $e = uv$ vor, dann *inzidieren* u und v mit e, u und v sind *adjazent* (oder *direkt verbunden*) und u und v bilden die *Endknoten* von e.

Im Folgenden werden nur *endliche* Graphen betrachtet, also Graphen mit endlicher Knotenmenge. Die Anzahl der Knoten eines Graphen G heißt *Ordnung* von G und die Anzahl der Kanten von G heißt *Größe* von G.

Ein Graph kann durch ein *Diagramm* dargestellt werden, in dem die Knoten durch Punkte der Zeichenebene und die Kanten durch stetige Streckenzüge repräsentiert werden.

Beispiel 21.1. Der Graph G mit der Knotenmenge $V = \{v_1, \ldots, v_4\}$ und der Kantenmenge $E = \{v_1v_3, v_2v_3, v_2v_4, v_3v_4\}$ wird durch das Diagramm in Abb. 21.1 dargestellt.

Ein Graph $G = (V, E)$ hat weder *Schlingen* noch *Mehrfachkanten*. Schlingen sind 1-Teilmengen von V, also Kanten, die nur mit einem Knoten inzidieren. Mehrfachkanten sind Multimengen von 2-Teilmengen von V, also mehrfache Verbindungen zwischen zwei Knoten.

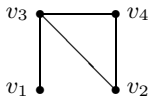


Abb. 21.1. Ein Diagramm des Graphen in 21.1.

Sei $G = (V, E)$ ein Graph. Die Anzahl der mit einem Knoten $v \in V$ inzidierenden Kanten wird der *Grad* von v genannt und mit $d(v)$ bezeichnet. Ist $d(v) = 0$, so heißt v *isoliert*. Haben alle Knoten in G den gleichen Grad k, so wird G als *k-regulär* bezeichnet.

Satz 21.2. (Handschlagslemma) *Für jeden Graphen $G = (V, E)$ gilt*

$$\sum_{v \in V} d(v) = 2|E|. \tag{21.1}$$

Beweis. Jede Kante wird in der Summe auf der linken Seite zweimal gezählt, je einmal für jeden inzidierenden Knoten. □

Korollar 21.3. *In jedem Graphen ist die Anzahl der Knoten mit ungeradem Grad stets gerade.*

Beweis. Nach dem Handschlagslemma ist die Summe aller Grade gerade. Durch Subtrahieren der Summe aller geraden Grade ergibt sich eine gerade Zahl, die die Summe aller ungeraden Grade darstellt. Also muss die Summe aller ungeraden Grade eine gerade Anzahl von Summanden haben. □

Beispiel 21.4. Können 333 Telefone so zusammengeschaltet werden, dass jedes Telefon mit drei Telefonen direkt verbunden ist? Die Antwort ist zu verneinen, denn die Summe der Grade des Telefonnetzes wäre ungerade $(333 \cdot 3)$.

Die *Gradfolge* eines Graphen G ist die absteigend sortierte Folge der Grade aller Knoten in G. Beispielweise hat der Graph in Abb. 21.1 die Gradfolge $(3, 2, 2, 1)$. Umgekehrt gehört nicht zu jeder absteigenden Folge natürlicher Zahlen ein Graph. Etwa gibt es keinen Graphen mit der Gradfolge $(5, 3, 2, 2, 2, 1)$, denn die Summe der Grade ist ungerade.

Teilgraphen

Sei $G = (V, E)$ ein Graph. Ein *Teilgraph* von G ist ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$. Ein Teilgraph G' von G kann als der von seiner Kantenmenge E' *induzierte Teilgraph* von G angesehen werden. Ein Teilgraph G' von G wird als der durch seine Knotenmenge V' *induzierte Teilgraph* von G bezeichnet, wenn jede Kante in G, die zwei Knoten in G' verbindet, zu G' gehört.

Beispiel 21.5. In Abb. 21.2 ist ein Graph G mitsamt zweier seiner Teilgraphen G_1 und G_2 dargestellt. Der Teilgraph G_2 wird von der Knotenmenge $\{b, c, d\}$ induziert, der Teilgraph G_1 nicht.

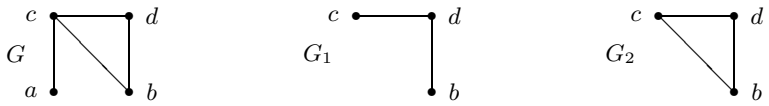


Abb. 21.2. Ein Graph G mit zwei Teilgraphen.

Isomorphismen

Seien $G = (V, E)$ und $G' = (V', E')$ Graphen. Eine Abbildung $\phi : V \to V'$ heißt ein *Isomorphismus* von G auf G', wenn ϕ bijektiv ist und für alle $u, v \in V$ gilt $uv \in E$ genau dann, wenn $\phi(u)\phi(v) \in E'$. Zwei Graphen G und G' heißen *isomorph*, wenn es einen Isomorphismus von G auf G' gibt.

Beispiel 21.6. Die beiden Graphen in Abb. 21.3 sind isomorph, denn ein Isomorphismus ist durch $\phi(a) = 1$, $\phi(b) = 2$, $\phi(c) = 3$ und $\phi(d) = 4$ gegeben.

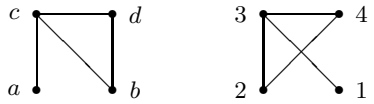


Abb. 21.3. Zwei isomorphe Graphen.

Isomorphe Graphen haben gleiche Knotenanzahl (Ordnung), Kantenanzahl (Größe) und Gradfolge. Anders ausgedrückt sind zwei Graphen nicht isomorph, wenn sie unterschiedliche Ordnung, Größe oder Gradfolge haben. Es gibt nichtisomorphe Graphen mit gleicher Gradfolge (Abb. 21.4).

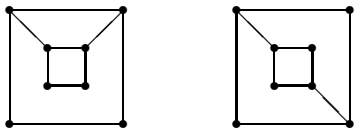


Abb. 21.4. Zwei nichtisomorphe Graphen mit gleicher Gradfolge.

Sei G ein Graph. Ein Isomorphismus von G auf G wird auch ein *Automorphismus* von G genannt.

Satz 21.7. *Die Menge aller Automorphismen eines Graphen bildet unter Komposition eine Gruppe.*

Die Gruppe aller Automorphismen von G wird die *Automorphismengruppe* von G genannt und mit $\mathrm{Aut}(G)$ bezeichnet.

Beispiel 21.8. Die Automorphismengruppe eines Quadrats (Abb. 21.5) ist nach 15.30 die Diedergruppe D_4, bestehend aus vier Drehungen id, (1234), $(1234)^2 = (13)(24)$, $(1234)^3 = (1432)$ und vier Spiegelungen $(12)(34)$, $(14)(23)$, (13), (24).

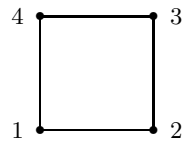


Abb. 21.5. Ein Quadrat.

21.2 Wege, Kreise und Zusammenhang

Sei $G = (V, E)$ ein Graph. Eine Folge $W = (v_0, \ldots, v_k)$ von Knoten $v_i \in V$ heißt ein *Weg* in G, wenn für alle i, $1 \leq i \leq k$, gilt $v_{i-1}v_i \in E$. Der Knoten v_0 ist der *Startknoten* und der Knoten v_k der *Endknoten* von W. Die *Länge* von W ist n, die Anzahl seiner Kanten. Ein Weg W heißt *einfach*, wenn W keinen Knoten mehrfach enthält.

Beispiel 21.9. Der Graph in Abb. 21.6 enthält etwa die einfachen Wege (s, a, d, g, f, i, t) und (s, a, b, e, h, i, t) der Länge 6.

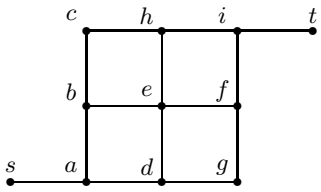


Abb. 21.6. Ein Wegenetz.

Ein *Kreis* in G ist ein Weg in G, in dem Start- und Endknoten identisch sind. Ein Kreis heißt *einfach*, wenn er (vom Start- und Endknoten abgesehen) keinen Knoten mehrfach enthält. Eine hin und zurück durchlaufene Kante uv ergibt einen einfachen Kreis (u, v, u) der Länge 2.

Beispiel 21.10. Der Graph in Abb. 21.6 besitzt etwa die einfachen Kreise (a, b, c, h, e, d, a) und (a, b, e, f, g, d, a) der Länge 6.

Zusammenhang

Sei $G = (V, E)$ ein Graph. Zwei Knoten $u, v \in V$ heißen *verbunden* in G, kurz $u \equiv_G v$, wenn $u = v$ oder es einen Weg von u nach v gibt. Sind je zwei Knoten in G verbunden, so heißt G *zusammenhängend*.

Lemma 21.11. *Sei $G = (V, E)$ ein Graph. Die Verbindbarkeit $\equiv_G$ in G ist eine Äquivalenz auf V.*

Die Äquivalenzklassen der Verbindbarkeit bilden nach Satz 5.5 eine Partition von V. Die von den Äquivalenzklassen aufgespannten Teilgraphen heißen *Komponenten* von G. Ist G zusammenhängend, so existiert nur eine Komponente.

Beispiel 21.12. Der Graph in Abb. 21.7 besteht aus zwei Komponenten, die von den Äquivalenzklassen $\{a, b\}$ und $\{c, d, e\}$ aufgespannt werden.

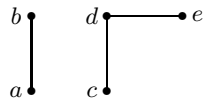


Abb. 21.7. Ein Graph mit zwei Komponenten.

Satz 21.13. *Sei $G = (V, E)$ ein zusammenhängender Graph und $e \in E$ eine auf einem einfachen Kreis in G liegende Kante. Der durch Streichen von e entstehende Teilgraph von G ist ebenfalls zusammenhängend.*

Beweis. Es bezeichne G' den durch Streichen von e entstehenden Teilgraphen von G. Seien $u, v \in V$. Da G zusammenhängend ist, gibt es einen Weg W in G von u nach v. Benutzt der Weg die Kante e nicht, so ist W auch ein Weg in G'. Andernfalls kann W so abgeändert werden, dass anstatt der Kante e der Rest des Kreises benutzt werden, auf dem e nach Voraussetzung liegt. □

Abstände

Sei $G = (V, E)$ ein Graph. Für je zwei Knoten $u, v \in V$ wird der *Abstand* $d_G(u, v)$ zwischen u und v in G definiert durch

$$d_G(u,v) = \begin{cases} 0 & \text{wenn } u = v, \\ \infty & \text{wenn } u \text{ und } v \text{ nicht verbunden,} \\ l & \text{wenn } l \text{ minimale Länge eines Weges in } G \text{ von } u \text{ nach } v. \end{cases} \tag{21.2}$$

Satz 21.14. *Ist G ein Graph, dann definiert der Abstand d_G eine Metrik auf G.*

Beweis. Wir zeigen nur die Dreiecksungleichung. Seien $u, v, w \in V$. Aus kürzesten Wegen zwischen u und v sowie v und w entsteht ein Weg der Länge $d_G(u, v) + d_G(v, w)$ zwischen u und w. Für den kürzesten Weg zwischen u und w gilt definitionsgemäß $d_G(u, w) \leq d_G(u, v) + d_G(v, w)$. □

Bäume

Ein Graph heißt *kreisfrei*, wenn er keinen einfachen Kreis der Länge ≥ 3 besitzt. Einfache Kreise der Länge 2 werden durch hin und zurück durchlaufene Kanten beschrieben und existieren in jedem Graphen mit wenigstens einer Kante. Ein kreisfreier Graph heißt ein *Wald*. Ein zusammenhängender Wald heißt ein *Baum* (Abb. 21.8).

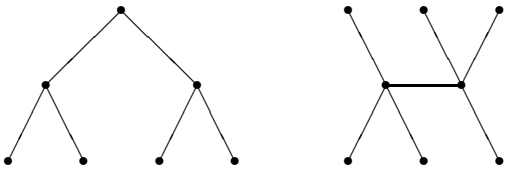


Abb. 21.8. Ein Wald mit zwei Bäumen.

Satz 21.15. *Ein Baum enthält mindestens zwei Knoten vom Grad 1.*

Beweis. Sei G ein Graph. Seien Knoten u und v in G so gewählt, dass der Abstand $d_G(u,v)$ maximal ist. Sei $W = (u, v_1, \ldots, v_{k-1}, v)$ ein kürzester Weg in G von u nach v. Angenommen, u hätte neben v_1 noch einen weiteren adjazenten Knoten w. Dann gilt nach Voraussetzung $d_G(w,v) \leq d_G(u,v)$. Also gibt es einen kürzesten Weg von w nach v, der nicht durch u führt. Folglich gibt es widersprüchlicherweise einen einfachen Kreis der Länge ≥ 3 in G. Somit hat u den Grad 1 und aus Symmetriegründen auch v. □

Satz 21.16. *Für jeden Baum $G = (V,E)$ gilt $|E| = |V| - 1$.*

Beweis. Der Fall $|V| = 1$ ist klar. Sei G ein Baum mit $|V| > 1$ Knoten. Nach Satz 21.15 existiert in G ein Knoten v vom Grad 1. Durch Streichen von v entsteht ein Teilgraph $G' = (V', E')$ von G, der wiederum ein Baum ist. Mit der Induktionsannahme ergibt sich $1 = |V'| - |E'| = (|V| - 1) - (|E| - 1) = |V| - |E|$. □

Sei $G = (V,E)$ ein Graph. Ein *Spannbaum* oder *Gerüst* von G ist ein Teilgraph von G, der Baum ist und jeden Knoten von G enthält (Abb. 21.9).

Satz 21.17. *Jeder zusammenhängende Graph hat einen Spannbaum.*

Beweis. Sei $G = (V,E)$ ein zusammenhängender Graph. Im Falle $|E| = 1$ ist die Aussage richtig. Sei $|E| > 1$. Ist G ein Baum, dann ist G sein eigener Spannbaum. Andernfalls gibt es einen einfachen Kreis in G. In diesem Kreis wird eine beliebige Kante gestrichen. Der dadurch entstehende Teilgraphen G' von G hat $|E| - 1$ Kanten und ist nach Satz 21.13 zusammenhängend. Also hat G' nach Induktionsannahme einen Spannbaum. Dieser Spannbaum ist auch ein Spannbaum von G. □

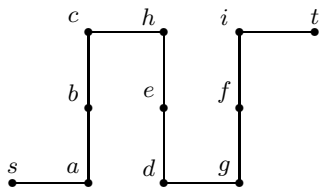


Abb. 21.9. Ein Spannbaum für den Graphen in Abb. 21.6.

Satz 21.18. *Ein zusammenhängender Graph $G = (V, E)$ ist ein Baum genau dann, wenn $|E| = |V| - 1$.*

Beweis. Sei $|E| = |V| - 1$. Angenommen, G wäre kein Baum. Dann gibt es einen einfachen Kreis in G. In diesem Kreis wird eine beliebige Kante herausgenommen. Der daraus resultierende Teilgraph $G' = (V, E')$ von G ist nach Satz 21.13 zusammenhängend. Für die Kantenmenge von G' gilt $|E'| < |V| - 1$. Andererseits enthält G' nach den Sätzen 21.16 und 21.17 einen Spannbaum mit $|V| - 1$ Kanten, dessen Kanten widersprüchlicherweise in E' liegen. Die Umkehrung ist nach Satz 21.16 bereits bewiesen. □

Eine Kante e eines Graphen G heißt eine *Brücke* oder ein *Isthmus* von G, wenn G durch Streichen von e in zwei Komponenten zerfällt. Beispielsweise besitzt der Graph in Abb. 21.1 als einzige Brücke die Kante v_1v_3.

Bipartite Graphen

Ein Graph $G = (V, E)$ heißt *bipartit*, wenn es eine 2-Partition von V in Teilmengen V_1 und V_2 gibt, so dass jede Kante in G einen Endknoten in V_1 und einen Endknoten in V_2 hat.

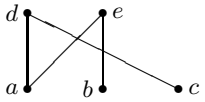


Abb. 21.10. Ein bipartiter Graph mit der 2-Partition $\{\{a, b, c\}, \{d, e\}\}$.

Satz 21.19. *Ein zusammenhängender Graph G ist bipartit genau dann, wenn G keinen Kreis von ungerader Länge enthält.*

Beweis. Sei $G = (V, E)$ bipartit mit der 2-Partition $\{V_1, V_2\}$. Sei $K = (v_0, v_1, \ldots, v_k)$ ein Kreis in G. O.B.d.A. sei $v_0 \in V_1$. Dann folgt $v_1 \in V_2$, $v_2 \in V_1$, $v_3 \in V_2$ usw. Also ist $v_k = v_0 \in V_1$. Somit hat K gerade Länge.

Umgekehrt enthalte G keinen Kreis von ungerader Länge. Sei $v \in V$. Wir definieren

$$V_1 = \{u \in V \mid d_G(v,u) \equiv 1 \bmod 2\}$$

und

$$V_2 = \{u \in V \mid d_G(v,u) \equiv 0 \bmod 2\}.$$

Es ist klar, dass $\{V_1, V_2\}$ eine 2-Partition von V ist. Angenommen, es gäbe eine Kante uw in G mit $u, v \in V_1$. Dann gibt es einen Kreis, bestehend aus der Kante uw, einem Weg der Länge $d_G(w,v)$ von w nach v und einem Weg der Länge $d_G(v,u)$ von v nach u. Dieser Kreis hat die Länge $1 + d_G(w,v) + d_G(v,u)$, die nach Voraussetzung widersprüchlicherweise ungerade ist. Analog wird gezeigt, dass es keine Kante uv in G gibt mit $u, v \in V_2$. □

21.3 Planare Graphen

Ein Graph heißt *planar*, wenn er ein Diagramm besitzt, das kreuzungsfrei in die Ebene gezeichnet werden kann. Ein solches Diagramm wird *eben* genannt. Ein ebenes Diagramm teilt die Zeichenebene in *Flächen* oder *Gebiete*, wobei die gezeichneten Kanten als begrenzende Linien aufgefasst werden.

Beispiel 21.20. Das Hexaeder besitzt ein ebenes Diagramm, das die Zeichenebene in sechs Flächen teilt (Abb. 21.11). Das außerhalb des ebenen Diagramms liegende Gebiet zählt ebenfalls hinzu.

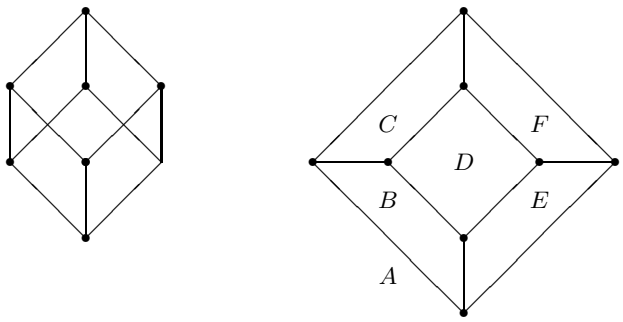


Abb. 21.11. Ein Hexaeder und ein ebenes Diagramm des Hexaeders (mit den Flächen $A, \ldots, F$).

Satz 21.21. (Eulersche Polyederformel) *Für jedes ebene Diagramm eines zusammenhängenden Graphen G mit n Knoten, m Kanten und f Flächen gilt*

$$n - m + f = 2. \tag{21.3}$$

Beweis. Im Falle $f = 1$ besitzt G ein ebenes Diagramm mit einer Fläche. Also ist G kreisfrei, mithin ein Baum. Nach Satz 21.16 gilt $m = n - 1$, woraus die Behauptung folgt.

Sei $f \geq 2$. Dann enthält G mindestens einen einfachen Kreis. Wird eine auf diesem Kreis liegende Kante gestrichen, so hat der entstehende Graph G' nur mehr $m - 1$ Kanten und $f - 1$ Flächen. Für den Teilgraphen G' gilt nach Induktionsannahme $n - (m - 1) + (f - 1) = 2$, mithin $n - m + f = 2$. □

Satz 21.22. *Ein planarer zusammenhängender Graph G mit $n \geq 3$ Knoten hat höchstens $3n - 6$ Kanten.*

Beweis. Sei D ein ebenes Diagramm von $G = (V, E)$ und F die Menge aller Flächen in D. Wir betrachten eine Relation R zwischen Kanten und Fläche. Es sei eRf, wenn in D die Kante e die Fläche f berandet. Jede Kante berandet höchstens zwei Flächen und jede Fläche wird von mindestens drei Kanten berandet. Mit dem Prinzip der doppelten Abzählung folgt $3|F| \leq 2|E|$. Daraus erhellt sich anhand der eulerschen Polyederformel $|E| \leq 3n - 6$. □

Ein Graph G heißt *vollständig*, wenn je zwei Knoten in G adjazent sind. Ein vollständiger Graph mit n Knoten hat $\binom{n}{2}$ Kanten und wird mit K_n bezeichnet (Abb. 21.12). Der kleinste nichtplanare vollständige Graph ist der K_5. Denn dieser Graph hat 10 Kanten, während jeder planare Graph mit fünf Knoten nach Satz 21.22 höchstens $3 \cdot 5 - 6 = 9$ Kanten besitzt. Der *Umfang*

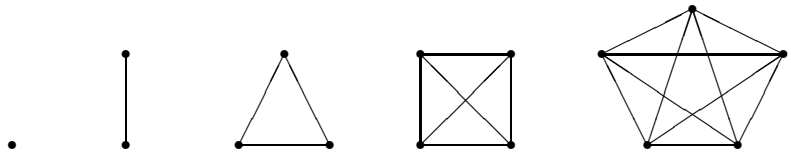


Abb. 21.12. Die ersten fünf vollständigen Graphen $K_1, \ldots, K_5$.

eines Graphen G ist die Länge eines kürzesten einfachen Kreises in G. Der Umfang eines kreisfreien Graphen wird als ∞ definiert.

Satz 21.23. *Ein planarer zusammenhängender Graph G mit $n \geq 3$ Knoten und Umfang $g \geq 3$ hat höchstens* $\max\{g(n-2)/(g-2), n-1\}$ *Kanten.*

Beweis. Ist G kreisfrei, dann ist $g > n$ und das Maximum ist $|E| \leq n - 1$, was nach Satz 21.18 richtig ist. Andernfalls ist $g \leq n$. Wir unterscheiden zwei Fälle. Erstens besitze G eine Brücke. Dann zerfällt G durch Streichen der Brücke in Komponenten G_1 und G_2. Die Komponente G_i besitze n_i Knoten und m_i Kanten. Mit der Induktionsannahme folgt

$$\begin{aligned} m &= m_1 + m_2 + 1 \\ &\leq \max\{g(n_1-2)/(g-2), n_1 - 1\} + \max\{g(n_2-2)/(g-2), n_2 - 1\} \\ &\leq \max\{g(n-2)/(g-2), n-1\}. \end{aligned}$$

Zweitens sei G brückenfrei. Bezeichnet f_i die Anzahl der Flächen in G, die von i Kanten berandet werden, dann gilt

$$2m = \sum_i i f_i = \sum_{i \geq g} i f_i \geq g \sum_{i \geq g} f_i = gf.$$

Mit der eulerschen Polyederformel folgt

$$m + 2 = n + f \leq n + \frac{2}{g} m,$$

woraus sich $m \leq g(n-2)/(g-2)$ ergibt. □

Ein bipartiter Graph heißt *vollständig*, wenn für die 2-Partition $\{V_1, V_2\}$ seiner Knotenmenge gilt, dass jeder Knoten in V_1 mit jedem Knoten in V_2 verbunden ist. Ein vollständiger bipartiter Graph mit $|V_1| = m$ und $|V_2| = n$ wird mit $K_{m,n}$ bezeichnet. Der vollständige bipartite Graph $K_{3,3}$ ist nicht planar (Abb. 21.13). Denn dieser Graph hat 9 Kanten, während jeder planare Graph mit sechs Knoten nach Satz 21.23 höchstens $\max\{4(6-2)/(4-2), 6-2\} = 8$ Kanten besitzt.

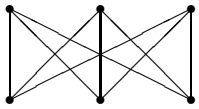


Abb. 21.13. Der bipartite Graph $K_{3,3}$.

Wir stellen ein notwendiges und hinreichendes Kriterium für die Planarität von Graphen auf. Eine *Unterteilung* eines Graphen G ist ein Graph, der aus G durch sukzessives Anwenden folgender Operation entsteht: Wähle eine Kante uv in G und ersetze diese durch einen Weg $(u, w_1, \ldots, w_k, v)$, wobei die Knoten w_i, $1 \leq i \leq k$, nicht in G vorkommen dürfen. Zwei Graphen und heißen *homöomorph*, wenn sie Unterteilungen desselben Graphen sind (Abb. 21.14).

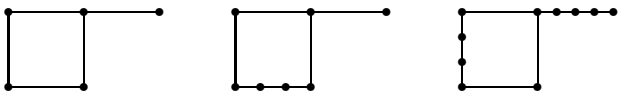


Abb. 21.14. Ein Graph und zwei Unterteilungen dieses Graphen.

Satz 21.24. (Kurt Kuratowski, 1896-1980) *Ein Graph ist planar genau dann, wenn er keinen zu K_5 oder $K_{3,3}$ homöomorphen Teilgraphen enthält.*

Beispiel 21.25. Der Graph in Abb. 21.15 ist nicht planar, weil er einen zu $K_{3,3}$ homöomorphen Teilgraphen enthält.

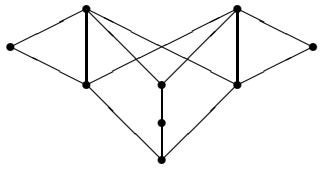


Abb. 21.15. Ein nichtplanarer Graph, der eine Unterteilung von $K_{3,3}$ enthält.

21.4 Datenstrukturen und Algorithmen

Sei $G = (V, E)$ ein Graph mit $V = \{v_1, \ldots, v_n\}$. Die *Adjazenzmatrix* von G ist eine $n \times n$-Matrix $A(G) = (a_{ij})$ mit

$$a_{ij} = \begin{cases} 1 \text{ falls } v_i v_j \in E, \\ 0 \text{ sonst.} \end{cases} \tag{21.4}$$

Beispielsweise hat der Graph in Abb. 21.1 die Adjazenzmatrix

$$A = \begin{pmatrix} 0\,0\,1\,0 \\ 0\,0\,1\,1 \\ 1\,1\,0\,1 \\ 0\,1\,1\,0 \end{pmatrix}.$$

Satz 21.26. *Sei A die Adjazenzmatrix eines Graphen G. Der (i, j)-te Eintrag der k-ten Potenz von A liefert die Anzahl der Wege in G der Länge k von v_i nach v_j.*

Beweis. Wir setzen $A^k = (a_{ij}^{(k)})$. Für $k = 1$ ist die Aussage klar. Für $k \geq 1$ gilt definitionsgemäß

$$a_{ij}^{(k+1)} = \sum_{l=1}^{n} a_{il}^{(k)} a_{lj},$$

wobei $a_{ij}^{(k)}$ nach Induktionsannahme die Anzahl der Wege in G der Länge k von v_i nach v_j ist. Der Term $\sum_{l=1}^{n} a_{il}^{(k)} a_{lj}$ beschreibt die Anzahl der Wege in G der Länge $k+1$ von v_i nach v_j, die sich aus einem Weg der Länge k von v_i nach v_l und einer Kante $v_k v_j$ zusammensetzen, wobei über alle entsprechenden Zwischenknoten summiert wird. Weitere Wege der Länge k von v_i nach v_j gibt es nicht. Damit ist die Aussage bewiesen. □

Sei $G = (V, E)$ ein Graph mit $V = \{v_1, \ldots, v_n\}$ und $E = \{e_1, \ldots, e_m\}$. Die *Inzidenzmatrix* von G ist eine $n \times m$-Matrix $B(G) = (b_{ij})$ mit

$$b_{ij} = \begin{cases} 1 \text{ falls } v_i \text{ mit } e_j \text{ inzidiert}, \\ 0 \text{ sonst.} \end{cases} \tag{21.5}$$

Beispielsweise hat der Graph in Abb. 21.1 die Inzidenzmatrix ($e_1 = v_1v_3$, $e_2 = v_2v_3$, $e_3 = v_2v_4$, $e_4 = v_3v_4$)

$$B = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,1\,0 \\ 1\,1\,0\,1 \\ 0\,0\,1\,1 \end{pmatrix}.$$

Eine Inzidenzmatrix enthält in jeder Spalte zwei Einsen und in jeder Zeile soviele Einsen, wie der Grad des entsprechenden Knotens angibt.

Eine Vorstufe der Matrixdarstellung von Graphen ist die Darstellung durch Listen. Eine *Adjazenzliste* eines Graphen $G = (V, E)$ mit der Knotenmenge $V = \{v_1, \ldots, v_n\}$ besteht aus Listen $L_1, \ldots, L_n$, wobei die Liste L_i die mit v_i adjazenten Knoten enthält. Der Graph in Abb. 21.1 wird durch folgende Adjazenzliste dargestellt: $L_1 : v_3$, $L_2 : v_3, v_4$, $L_3 : v_1, v_2, v_4$ und $L_4 : v_2, v_4$.

Der *Speicherbedarf* (Anzahl von Speicherzellen) für einen Graphen $G = (V, E)$ hängt von der Datenstruktur ab, in der er gespeichert wird. Der Speicherbedarf für eine Adjazenzmatrix ist $O(|V|^2)$, für eine Inzidenzmatrix $O(|V||E|)$ und für eine Adjazenzliste $O(|V| + |E|)$. Ein Algorithmus für einen Graphen G muss wenigstens die Knoten und Kanten von G einlesen. Hierfür beträgt der *Zeitbedarf* (Anzahl von Rechenoperationen) $O(|V| + |E|)$. Also muss ein Graphalgorithmus mit *linearem Zeitbedarf* $O(|V| + |E|)$ den Graphen durch eine Adjazenzliste (oder eine ähnlich speicherplatzsparende Datenstruktur) repräsentieren.

Wir stellen zwei Algorithmen vor, die einen Spannbaum eines Graphen konstruieren. Sei $G = (V, E)$ ein zusammenhängender Graph, v^+ die Menge der zu $v \in V$ adjazenten Knoten und $v_0 \in V$ fest gewählt. Der Algorithmus 21.1 benutzt eine Liste L, in der Elemente hinten anfügt und vorne entfernt werden. Die Knoten des Graphen werden in der Breite durchlaufen, d. h., nach jedem Knoten werden zuerst alle seine adjazenten Knoten bearbeitet. Dafür verantwortlich ist die als Warteschlange ("first-in, first-out") organisierte Liste. Dieses Suchprinzip wird *Breitensuche* genannt.

Der Algorithmus 21.2 verwendet eine Liste L, die nur an einer Seite zugänglich ist. Die Knoten des Graphen werden in der Tiefe durchlaufen, d. h., nach jedem Knoten werden jeweils die adjazenten Knoten seiner adjazenten Knoten bearbeitet. Dafür sorgt die als Stapelspeicher ("last-in, first-out") organisierte Liste. Dieses Suchprinzip wird als *Tiefensuche* bezeichnet. Wird der Graph durch eine Adjazenzliste dargestellt, dann ist die Laufzeit beider Algorithmen $O(|V| + |E|)$. Denn in beiden Fällen wird die Adjazenzliste je einmal abgearbeitet (Abb. 21.16).

Algorithmus 21.1 BreitenSuche(G)

Eingabe: Graph $G = (V, E)$
Ausgabe: Folge aller Knoten in G.

```
 1: wähle Startknoten v_0 ∈ V
 2: L := {v_0}
 3: V' := {v_0}
 4: E' := ∅
 5: while L ≠ ∅ do
 6:    entferne den ersten Knoten v aus L
 7:    print v
 8:    for all w ∈ v^+ und w war noch nicht in L do
 9:       if w ∉ V' then
10:          V' := V' ∪ {w}
11:          E' := E' ∪ {vw}
12:          füge w am Ende von L ein
13:       end if
14:    end for
15: end while
```

Algorithmus 21.2 TiefenSuche(G)

Eingabe: Graph $G = (V, E)$
Ausgabe: Folge aller Knoten in G.

```
 1: wähle Startknoten v_0 ∈ V
 2: L := {v_0}
 3: V' := {v_0}
 4: E' := ∅
 5: while L ≠ ∅ do
 6:    v ∈ L sei oberstes Listenelement
 7:    if es gibt w ∈ v^+ und w war noch nicht in L then
 8:       füge w in L ein
 9:       V' := V' ∪ {w}
10:       E' := E' ∪ {vw}
11:    else
12:       entferne v aus L
13:       print v
14:    end if
15: end while
```

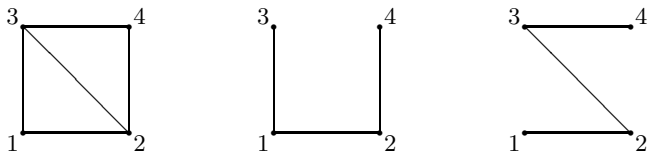


Abb. 21.16. Ein Graph und zwei Spannbäume des Graphen, der erste wird durch Breitensuche, der zweite durch Tiefensuche erhalten, wobei jeweils der kleinste Knoten zuerst ausgewählt wird.

Selbsttestaufgaben

21.1. Beweise das Handschlagslemma mithilfe des Prinzips der doppelten Abzählung.

21.2. Gibt es einen Graphen mit der Gradfolge $(3, 2, 2, 2)$?

21.3. Welche Gestalt hat ein zusammenhängender 2-regulärer Graph?

21.4. Zeichne einen Graphen, dessen Knoten die 2-Teilmengen von $\underline{5}$ sind und zwei Knoten durch eine Kante verbunden werden, wenn die zugehörigen 2-Teilmengen disjunkt sind. Dieser Graph wird *Petersen-Graph* genannt. Ist der Petersen-Graph planar?

21.5. Ein Baum habe drei Knoten vom Grad 3 und vier Knoten vom Grad 2. Die übrigen Knoten seien vom Grad 1. Wie viele Knoten vom Grad 1 gibt es?

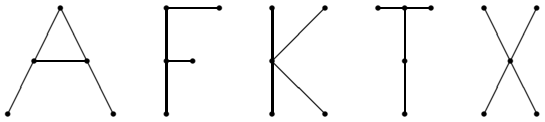
21.6. Schneide aus einem Schachbrett die beiden diagonal gegenüberliegenden Felder heraus. Kann dieses Brett (aus 62 Feldern) mit Dominosteinen, je zwei Felder bedecken, vollständig überdecken?

21.7. Beweise den Satz 21.7.

21.8. Beweise das Lemma 21.11.

21.9. Gib einen unendlichen Graphen an, der zu einem seiner echten Teilgraphen isomorph ist.

21.10. Welche der folgenden Graphen sind isomorph zueinander?



21.11. Zeige, dass die Graphen in Abb. 21.4 nicht isomorph sind.

21.12. Zeichne Diagramme von allen Graphen mit vier Knoten

21.13. Sei $G = (V, E)$ ein Graph. Das *Komplement* von G ist ein Graph $\overline{G} = (V, \overline{E})$ mit der Kantenmenge $\overline{E} = \binom{V}{2} \setminus E$. Betrachte alle Graphen mit vier Knoten. Welche dieser Graphen sind *selbstkomplementär*, d. h., isomorph zu ihrem eigenen Komplement?

21.14. Ermittle alle Spannbäume von K_4.

21.15. Sei $n \geq 2$ eine natürliche Zahl. Sei $W_n = (\{0, 1\}^n, E)$ der Graph des n-dimensionalen Würfels mit $uv \in E$ genau dann, wenn es genau ein i, $1 \leq i \leq n$, gibt mit $u_i \neq v_i$. Zeige, dass W_n einen hamiltonschen Kreis enthält.

21.16. Ein einfacher Kreis in einem Graphen G heißt *eulersch*, wenn der Weg jede Kante von G enthält. Ein Graph G heißt *eulersch*, wenn G einen eulerschen Kreis besitzt. Zeige, dass ein Graph G eulersch ist genau dann, wenn jeder Knoten in G geraden Grad aufweist.

21.17. Welche der Graphen K_n und $K_{m,n}$ sind eulersch?

21.18. Spezifiziere einen Algorithmus, der einen Graphen daraufhin testet, ob er ein Baum ist.

22

Netzwerke

In diesem Kapitel werden grundlegende Algorithmen für Netzwerke präsentiert. Hierbei geht es um die Konstruktion kürzester Wege, minimaler Spannbäume, maximaler Flüsse, minimaler trennender Knoten- und Kantenmengen, minimaler Knotenüberdeckungen und maximaler oder vollständiger Paarungen.

22.1 Kürzeste Wege

In diesem Abschnitt werden Algorithmen vorgestellt, die in einem Wegenetz kürzeste Wege zwischen je zwei Knoten bzw. von einem Knoten zu allen anderen liefern.

Ein *(gerichtetes) Wegenetz* ist ein Paar (D, ω), bestehend aus einem Digraphen $D = (V, E)$ und einer Kostenfunktion $\omega : E \to \mathbb{R}$ auf den Kanten von D. Die *Länge* eines gerichteten Weges $W = (v_0, \ldots, v_k)$ in D ist die Summe der Kosten seiner Kanten

$$\omega(W) = \sum_{i=0}^{k-1} \omega(v_i v_{i+1}). \tag{22.1}$$

Seien $u, v \in V$. Ein *kürzester Weg* in D von u nach v ist ein gerichteter Weg in D von u nach v mit minimaler Länge. Für den *Abstand* zwischen den Knoten eines Wegenetzes gilt

$$d_D(u, v) = \begin{cases} 0 & \text{wenn } u = v, \\ \infty & \text{wenn in } D \text{ kein Weg von } u \text{ nach } v \text{ existiert,} \\ l & \text{wenn } l \text{ Länge eines kürzesten Weges in } D \text{ von } u \text{ nach } v. \end{cases} \tag{22.2}$$

Satz 22.1. *Ist (D, ω) ein Wegenetz, dann definiert der Abstand d_D eine Metrik.*

Beispiel 22.2. Im Wegenetz von Abb. 22.1 ist (v_1, v_2, v_3, v_4) ein kürzester Weg von v_1 nach v_4 der Länge 6.

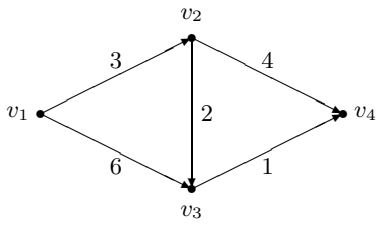


Abb. 22.1. Ein Wegenetz.

Kürzeste Wege zwischen je zwei Knoten

Zuerst behandeln wir einen Algorithmus von R. Floyd und S. Warshall, der in einem Wegenetz die Abstände zwischen je zwei Knoten ermittelt. Dabei sind negative Kantenbewertungen zugelassen, aber keine Kreise negativer Länge. Ein Kreis negativer Länge kann nämlich beliebig oft durchlaufen werden und hat somit beliebig kleine Länge. Sei (D, ω) ein Wegenetz ohne Kreise negativer Länge. Die Knotenmenge des Wegenetzes sei mit $V = \{v_1, \ldots, v_n\}$ bezeichnet.

Algorithmus 22.1 FLOYD-WARSHALL(D, ω)

Eingabe: Wegenetz (D, ω) mit $D = (V, E)$, $V = \{v_1, \ldots, v_n\}$
Ausgabe: Kürzeste Wege zwischen allen Knoten des Wegenetzes

for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    if $v_i v_j \in E$ **then**
      $d^{(0)}(v_i, v_j) := \omega(v_i v_j)$
    else if $v_i = v_j$ **then**
      $d^{(0)}(v_i, v_j) := 0$
    else
      $d^{(0)}(v_i, v_j) := \infty$
    end if
  end for
end for
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      $d^{(k)}(v_i, v_j) := \min\{d^{(k-1)}(v_i, v_j), d^{(k-1)}(v_i, v_k) + d^{(k-1)}(v_k, v_j)\}$
    end for
  end for
end for

Satz 22.3. (Floyd-Warshall) *Der Algorithmus 22.1 berechnet den Abstand* $d_D(v_i, v_j) = d^{(n)}(v_i, v_j)$ *für alle Knoten* v_i, v_j *in einem Wegenetz* (D, ω)*. Der Zeitbedarf des Algorithmus' ist* $O(|V|^3)$*.*

Beweis. In jedem Durchlauf der dreifach geschachtelten Schleife wird eine Abstandsmatrix $D_k = (d^{(k)}(v_i, v_j))$ berechnet. Wir zeigen, dass $d^{(k)}(v_i, v_j)$ die Länge eines kürzesten Weges von v_i nach v_j ist, der nur $v_1, \ldots, v_k$ benutzt (von v_i und v_j abgesehen). Für $k = 0$ ist die Aussage klar. Sei $k \geq 1$. Wir betrachten einen kürzesten Weg in D von v_i nach v_j, der höchstens $v_1, \ldots, v_k$ benutzt. Wird v_k nicht verwendet, dann hat dieser Weg nach Induktionsannahme die Länge $d^{(k-1)}(v_i, v_j)$. Andernfalls setzt sich dieser Weg zusammen aus einem Weg W_1 von v_i nach v_k, einem Weg W_2 von v_k nach v_k und einem Weg W_3 von v_k nach v_j, wobei W_1 und W_3 nur $v_1, \ldots, v_{k-1}$ benutzen. Da D nach Voraussetzung keine Kreise negativer Länge enthält, ist die Länge dieses Weges aus der Summe der Längen von W_1 und W_3 gegeben. Somit gilt nach Induktionsannahme $d^{(k)}(v_i, v_j) = d^{(k-1)}(v_i, v_k) + d^{(k-1)}(v_k, v_j)$. Mit $k = n$ folgt die erste Behauptung. Für den Zeitbedarf ist die dreifach geschachtelte Schleife maßgebend. □

Beispiel 22.4. Für das Wegenetz in Abb. 22.1 liefert Floyd-Warshall folgende erste und letzte Abstandsmatrizen:

$$D_0 = \begin{pmatrix} 0 & 3 & 6 & \infty \\ \infty & 0 & 2 & 4 \\ \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & 0 \end{pmatrix}, \quad D_4 = \begin{pmatrix} 0 & 3 & 5 & 6 \\ \infty & 0 & 2 & 3 \\ \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & 0 \end{pmatrix}.$$

Kürzeste Wege von einem Knoten zu allen anderen

Abschließend wird ein Algorithmus von E.W. Dijkstra vorgestellt, der in einem Wegenetz die Abstände zwischen einem festen Knoten und allen übrigen Knoten berechnet. Sei (D, ω) ein Wegenetz, v_0 ein Knoten in D und für jede Kante e gelte $\omega(e) \geq 0$.

Satz 22.5. (Dijkstra) *Der Alg. 22.2 berechnet den Abstand* $d_D(v_0, v) = d(v)$ *von* v_0 *zu allen übrigen Knoten* v *in einem Wegenetz* (D, ω)*. Der Zeitbedarf des Algorithmus' ist* $O(|V|^2)$*.*

Beweis. Es gilt die folgende Schleifeninvariante:

- Für jeden Knoten $v \in S$ ist $d(v) = d_D(v_0, v)$.
- Für jeden Knoten $v \in V \setminus S$ ist $d(v)$ die Länge eines kürzesten Weges von v_0 nach v, der nur Knoten in S benutzt (bis auf den Endknoten v).

Nach Terminierung ist $S = V$ und somit die erste Aussage klar. Der Zeitbedarf bestimmt sich aus der `while`-Schleife, die $|V|$-mal durchlaufen wird und in jedem Durchlauf höchstens $|V|$ Schritte erfordert. □

Algorithmus 22.2 DIJKSTRA(D, ω, v_0)

Eingabe: Wegenetz (D, ω) mit $D = (V, E)$ und Startknoten $v_0 \in V$
Ausgabe: Kürzeste Wege von v_0 zu allen anderen Knoten

```
 1: d(v_0) := 0
 2: S := ∅
 3: for all v ∈ V \ {v_0} do
 4:    d(v) := ∞
 5: end for
 6: while S ≠ V do
 7:    wähle v ∈ V \ S mit d(v) = min{d(u) | u ∈ V \ S}
 8:    S := S ∪ {v}
 9:    for all u ∈ V \ S do
10:       d(u) := min{d(u), d(v) + ω(vu)}
11:    end for
12: end while
```

Beispiel 22.6. Für das Wegenetz in Abb. 22.1 mit dem Startknoten v_1 liefert DIJKSTRA zunächst $S = \emptyset$, $d(v_1) = 0$ und $d(v_2) = d(v_3) = d(v_4) = \infty$. Im ersten Schritt ergibt sich $S = \{v_1\}$, $d(v_1) = 0$, $d(v_2) = 3$, $d(v_3) = 6$ und $d(v_4) = \infty$. Im zweiten Schritt erhalten wir $S = \{v_1, v_2\}$, $d(v_1) = 0$, $d(v_2) = 3$, $d(v_3) = 5$ und $d(v_4) = 7$. Im dritten Schitt berechnet sich $S = \{v_1, v_2, v_3\}$, $d(v_1) = 0$, $d(v_2) = 3$, $d(v_3) = 5$, $d(v_4) = 6$ und im letzten Schritt haben wir $S = \{v_1, v_2, v_3, v_4\}$, $d(v_1) = 0$, $d(v_2) = 3$, $d(v_3) = 5$, $d(v_4) = 6$.

22.2 Minimale Spannbäume

In diesem Abschnitt wird ein Algorithmus von J. Kruskal (1956) vorgestellt, der für einen bewerteten Graphen einen Spannbaum mit minimalen Kosten konstruiert.

Sei (G, ω) ein *Wegenetz*, bestehend aus einem Graphen $G = (V, E)$ und einer Kostenfunktion $\omega : E \to \mathbb{R}$ auf den Kanten von G. Die *Kosten* eines Teilgraphen $G' = (V', E')$ von G sind die Summe der Kosten seiner Kanten

$$\omega(G') = \sum_{e \in E'} \omega(e). \tag{22.3}$$

Ein *minimaler Spannbaum* eines Wegenetzes (G, ω) ist ein Spannbaum von G, der unter allen Spannbäumen von G minimale Kosten besitzt (Abb. 22.2).

Die Anzahl der Spannbäume eines Graphen ist so groß, dass es sich lohnt, nach guten Algorithmen zur Bestimmung minimaler Spannbäume zu suchen.

Satz 22.7. (Arthur Cayley, 1821-1895) *Die Anzahl der Spannbäume des vollständigen Graphen K_n ist n^{n-2}.*

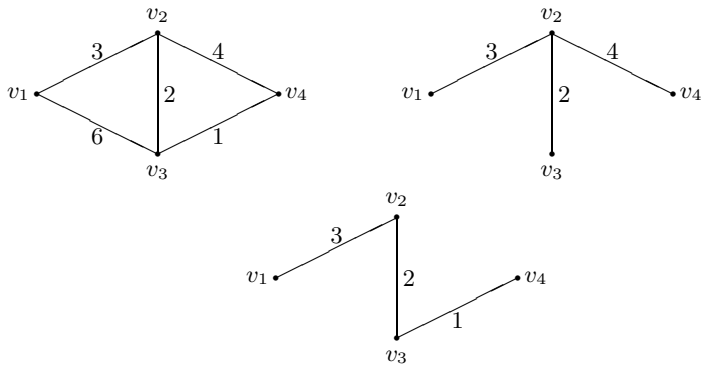


Abb. 22.2. Ein Wegenetz mit zwei Spannbäumen, der erste Spannbaum hat Kosten 9 und der zweite Spannbaum hat minimale Kosten 6.

Beweis. Sei $\{v_1, \ldots, v_n\}$ die Knotenmenge von K_n. Sei A eine k-Teilmenge von $\{v_1, \ldots, v_n\}$. Sei $T_{n,k}$ die Anzahl der Wälder mit der Knotenmenge $\{v_1, \ldots, v_n\}$, die jeweils aus k Bäumen bestehen, sodass die Knoten von A in jeweils unterschiedlichen Bäumen liegen.

Sei $A = \{v_1, \ldots, v_k\}$. Sei G ein Wald und sei der Knoten v_1 adjazent zu i Knoten. Wird der Knoten v_1 gestrichen, dann entstehen $T_{n-1,k-1+i}$ Wälder mit $A = \{v_2, \ldots, v_k\}$. Es folgt

$$T_{n,k} = \sum_{i=0}^{n-k} \binom{n-k}{i} T_{n-1,k-1+i}. \tag{22.4}$$

Setzen wir $T_{0,0} = 1$ und $T_{n,0} = 0$ für $n > 0$, dann ergibt sich mittels vollständiger Induktion

$$T_{n,k} = kn^{n-k-1}. \tag{22.5}$$

Insbesondere ist $T_{n,1} = n^{n-2}$ die Anzahl der Spannbäume von K_n. □

Algorithmus von Kruskal

Satz 22.8. (Kruskal) *Sei (G, w) ein Wegenetz auf einem zusammenhängenden Graphen $G = (V, E)$. Der Algorithmus 22.3 berechnet einen minimalen Spannbaum für das Wegenetz (G, ω).*

Um den Satz zu beweisen, wird ein Kriterium für die Minimalität von Spannbäumen benötigt. MinSpannbaum hat die Laufzeit $O(|E| \log |E|) = O(|E| \log |V|)$, wenn die Kanten des Graphen gemäß ihrer Kosten vorsortiert werden.

Beispiel 22.9. Für das Wegenetz in Abb. 22.2 liefert MinSpannbaum der Reihe nach die Kanten v_3v_4, v_2v_3 und v_1v_2. Der hiervon aufgespannte Spannbaum ist minimal mit Kosten 6.

Algorithmus 22.3 MinSpannbaum(G, ω)

Eingabe: Wegenetz (G, ω), $G = (V, E)$ zusammenhängend
Ausgabe: Kanten eines minimalen Spannbaums

```
1: E' := ∅
2: while E ≠ ∅ do
3:    wähle e ∈ E mit ω(e) = min{ω(e') | e' ∈ E}
4:    E := E \ {e}
5:    if aufgespannter Graph von E' ∪ {e} ist kreisfrei then
6:       E' := E' ∪ {e}
7:    end if
8:    return E'
9: end while
```

Ein Kriterium für minimale Spannbäume

Sei (G, ω) ein Wegenetz und $G = (V, E)$ zusammenhängend, sei $G' = (V, E')$ ein Spannbaum von G. Fügen wir in den Spannbaum eine Kante $e = uv \in E \setminus E'$ ein, so enthält der entstehende Teilgraph einen Kreis. Denn in G' gibt es zwischen u und v einen Weg, der zusammen mit der neuen Kante e einen Kreis bildet. Die Kante e heißt eine *Sehne* von G' und der mit e gebildete Kreis $K_{G'}(e)$ *Fundamentalkreis* der Sehne e (Abb. 22.3).

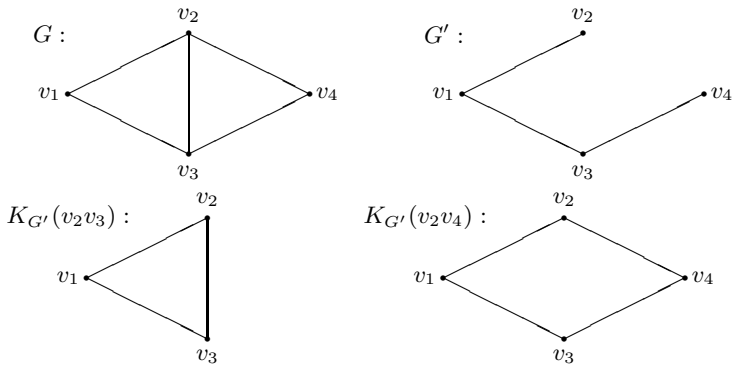


Abb. 22.3. Ein Graph G mit einem Spannbaum G' und Fundamentalkreisen $K_{G'}(v_2v_3)$ und $K_{G'}(v_2v_4)$.

Satz 22.10. *Sei (G, ω) ein Wegenetz und G zusammenhängend. Ein Spannbaum G' von G ist minimal genau dann, wenn für jede Sehne e von G' gilt $\omega(e) \geq \omega(e')$ für alle Kanten e' in $K_{G'}(e)$.*

Beweis. Sei G' minimal. Angenommen, es gäbe eine Sehne e von G' und eine Kante $e' \in K_{G'}(e)$ mit $\omega(e) < \omega(e')$. Wir streichen e' aus G' und fügen e

hinzu. Dadurch entsteht ein Spannbaum von G mit geringeren Kosten, was der Voraussetzung widerspricht.

Umgekehrt sei $\omega(e) \geq \omega(e')$ für jede Sehne e von G' und alle $e' \in K_{G'}(e)$. Sei G_0 ein minimaler Spannbaum von G. Wir zeigen, dass G' und G_0 die gleichen Kosten haben, also auch G' minimal ist. Gehört jede Kante von G_0 auch zu G', dann ist $G' = G_0$. Andernfalls gibt es eine Kante e in G_0, die nicht zu G' gehört und somit eine Sehne von G' ist. Durch Streichen von e zerfällt G_0 in zwei Komponenten und durch Hinzunehmen einer Kante e' des Fundamentalkreises $K_{G'}(e)$ entsteht ein Spannbaum G_1. Da G_0 minimal ist, gilt $\omega(e) \leq \omega(e')$. Andererseits ist e eine Sehne von G' und somit $\omega(e) \geq \omega(e')$. Also ist $\omega(e) = \omega(e')$ und deshalb auch G_1 ein minimaler Spannbaum. Allerdings hat G_1 mit G' eine Kante mehr gemeinsam als G_0. Mit vollständiger Induktion folgt, dass auch G' minimal ist. □

Abschließend wird der Satz 22.8 gezeigt.

Beweis. Der in MinSpannbaum berechnete Teilgraph G' enthält alle Knoten von G, ist kreisfrei und zusammenhängend, mithin ein Spannbaum von G. Ferner werden im Algorithmus die Kanten so gewählt, dass die Bedingung für die Sehnen in Satz 22.10 erfüllt ist. Also ist der berechnete Spannbaum minimal. □

22.3 Maximale Flüsse

In diesem Abschnitt wird ein Algorithmus von L.R. Ford und D.R. Fulkerson (1956) vorgestellt, der für einen bewerteten Graphen einen maximalen Fluss von einer Quelle zu einer Senke konstruiert.

Ein *Flussnetz* ist ein Quadrupel $N = (D, \kappa, q, s)$, bestehend aus einem Digraphen $D = (V, E)$, einer Kostenfunktion $\kappa : E \to \mathbb{R}_0^+$, genannt *Kapazität*, und Knoten q und s in D, sodass es einen gerichteten Weg in D von q von s gibt. Die von Quelle und Senke verschiedenen Knoten in D heißen *innere Knoten* (Abb. 22.4).

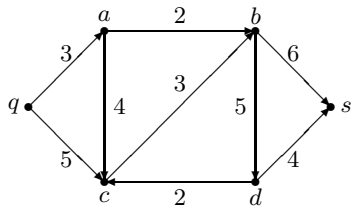


Abb. 22.4. Ein Flussnetz.

Flüsse in Flussnetzen

Sei $N = (D, \kappa, q, s)$ ein Flussnetz. Ein *Fluss* auf N ist eine Kostenfunktion $f : E \to \mathbb{R}_0^+$ mit den folgenden Eigenschaften:

- $f \leq \kappa$, d. h., $f(e) \leq \kappa(e)$ für alle Kanten $e \in E$.
- Für jeden inneren Knoten $v \in V$ gilt

$$\sum_{w \in v^+} f(vw) = \sum_{u \in v^-} f(uv). \tag{22.6}$$

Die erste Bedingung besagt, dass der Fluss die Kapazität nicht überschreiten darf. Die zweite Bedingung bedeutet, dass in jeden inneren Knoten so viel herausfließt wie hineinfließt. Diese Bedingung wird in Anlehnung an ein elektrotechnisches Analogon *kirchhoffsche Bedingung* genannt (Abb. 22.5).

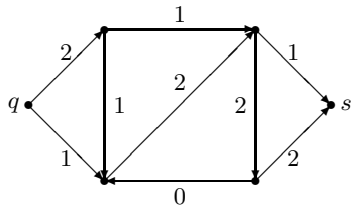


Abb. 22.5. Ein Fluss auf dem Flussnetz von Abb. 22.4.

Lemma 22.11. *Für jeden Fluss f auf einem Flussnetz $N = (D, \kappa, q, s)$ gilt*

$$\sum_{w \in q^+} f(qw) - \sum_{u \in q^-} f(uq) = \sum_{u \in s^-} f(us) - \sum_{w \in s^+} f(sw). \tag{22.7}$$

Beweis. Es gilt

$$\sum_{v \in V} \sum_{w \in v^+} f(vw) = \sum_{v \in V} \sum_{u \in v^-} f(uv),$$

weil auf beiden Seiten über alle Kanten summiert wird. Daraus folgt mit dem kirchhoffschen Gesetz

$$\begin{aligned} 0 &= \sum_{v \in V} \left(\sum_{w \in v^+} f(vw) - \sum_{u \in v^-} f(uv) \right) \\ &= \sum_{v \in \{q,s\}} \left(\sum_{w \in v^+} f(vw) - \sum_{u \in v^-} f(uv) \right). \end{aligned}$$

□

Die Zahl (22.7) wird *Wert des Flusses* f genannt und mit $\Phi(f)$ bezeichnet. Sie charakterisiert den Gesamtfluss durch das Flussnetz. Ein Fluss f heißt *maximal* auf N, wenn für jeden Fluss f' auf N gilt $\Phi(f') \leq \Phi(f)$.

Schnitte in Flussnetzen

Sei $N = (D, \kappa, q, s)$ ein Flussnetz. Eine Teilmenge S von V heißt ein *Schnitt* in N, wenn S die Quelle q, aber nicht die Senke s enthält. Die *Kapazität* eines Schnitts S in N ist die Zahl

$$\kappa(S) = \sum_{\substack{e^- \in S \\ e^+ \in \overline{S}}} \kappa(e). \tag{22.8}$$

Die Kapazität eines Schnitts S ist die Summe der Kapazitäten von Kanten, deren Startknoten in S und deren Endknoten in $\overline{S} = V \setminus S$ liegen. Ein Schnitt S in N heißt *minimal*, wenn für jeden Schnitt S' in N gilt $\kappa(S) \leq \kappa(S')$. Ein minimaler Schnitt existiert stets, da es nur endlich viele Schnitte in einem Flussnetz gibt.

Beispiel 22.12. Für das Flussnetz in Abb. 22.4 sind einige Schnitte in folgender Tabelle angegeben:

S	$\{q\}$	$\{q,a\}$	$\{q,a,c\}$	$\{q,a,c,b\}$
$\kappa(S)$	8	11	5	11

Lemma 22.13. *Sei $N = (D, \kappa, q, s)$ ein Flussnetz. Für jeden Fluss f auf N und jeden Schnitt S in N gilt*

$$\Phi(f) \leq \kappa(S). \tag{22.9}$$

Beweis. Es gilt

$$\begin{aligned}
\Phi(f) &= \sum_{w \in q^+} f(qw) - \sum_{u \in q^-} f(uq) \\
&= \sum_{v \in S} \left(\sum_{w \in v^+} f(vw) - \sum_{u \in v^-} f(uv) \right) \\
&= \sum_{\substack{vw \in E \\ v, w \in S}} f(vw) + \sum_{\substack{vw \in E \\ v \in S, w \in \overline{S}}} f(vw) - \sum_{\substack{uv \in E \\ u, v \in S}} f(uv) - \sum_{\substack{uv \in E \\ v \in S, u \in \overline{S}}} f(uv) \\
&= \sum_{\substack{vw \in E \\ v \in S, w \in \overline{S}}} f(vw) - \sum_{\substack{uv \in E \\ v \in S, u \in \overline{S}}} f(uv) \\
&\leq \sum_{\substack{vw \in E \\ v \in S, w \in \overline{S}}} f(vw) \leq \kappa(S).
\end{aligned}$$

□

Lemma 22.14. *In jedem Flussnetz N existiert ein maximaler Fluss.*

Beweis. Nach Lemma 22.13 gilt für jeden Fluss f auf N

$$\Phi(f) \leq \sum_{vw \in E} \kappa(vw).$$

Also ist der maximale Flusswert $\Phi = \sup\{\Phi(f) \mid f \text{ Fluss auf } N\}$ endlich. Ist (f_i) eine Folge von Flüssen mit dem Grenzwert $\lim_i \Phi(f_i) = \Phi$, dann ist für jede (fest gewählte) Kante $vw \in E$ auch die Unterfolge $(f_i(vw))$ konvergent mit einem Grenzwert $f(vw)$. Somit ist f ein Fluss mit maximalem Wert Φ. □

Der folgende Satz besagt, dass der maximale Fluss dem minimalen Schnitt entspricht. Im Englischen wird vom "MaxFlow MinCut Theorem" gesprochen.

Satz 22.15. (Ford-Fulkerson) *In jedem Netzwerk N ist der Wert eines maximalen Flusses gleich der Kapazität eines minimalen Schnitts*

$$\max\{\Phi(f) \mid f \text{ Fluss auf } N\} = \min\{\kappa(S) \mid S \text{ Schnitt in } N\}. \quad (22.10)$$

Beweis. Sei f ein maximaler Fluss auf N. Wir definieren induktiv ein Schnitt S in N:

- $S := \{q\}$.
- $S := S \cup \{w\}$ für ein $w \in \overline{S}$, falls es ein $v \in S$ gibt derart, dass $\kappa(vw) > f(vw)$, falls $vw \in E$, oder $f(wv) > 0$, falls $wv \in E$.

Zuerst wird gezeigt, dass S ein Schnitt ist. Angenommen, S wäre kein Schnitt, also $s \in S$. Dann gibt es eine Folge $X = (x_0, \ldots, x_m)$ von Knoten in N mit $x_0 = q$, $x_m = s$ und für jedes i, $1 \leq i \leq m$, ist eine der beiden folgenden Bedingungen erfüllt:

- $\kappa(x_{i-1}x_i) > f(x_{i-1}x_i)$, falls $x_{i-1}x_i \in E$,
- $f(x_ix_{i-1}) > 0$, falls $x_ix_{i-1} \in E$.

Die Kante $x_{i-1}x_i$ heißt *Vorwärtskante* und die Kante x_ix_{i-1} *Rückwärtskante* von X. Die Folge X wird *flussvergrößernder Weg von q nach s* (bzgl. f) genannt.

Sei ϵ das Minimum aller Werte $\kappa(e) - f(e)$ über alle Vorwärtskanten e und aller Werte $f(e)$ über alle Rückwärtskanten e von X. Definitionsgemäß ist $\epsilon > 0$ und ein Fluss $f^* = f(X, \epsilon)$ auf N wird definiert durch

$$f^*(e) = \begin{cases} f(e) + \epsilon & \text{falls } e \text{ Vorwärtskante von } X, \\ f(e) - \epsilon & \text{falls } e \text{ Rückwärtskante von } X, \\ f(e) & \text{sonst.} \end{cases} \quad (22.11)$$

Für den Wert des Flusses f^* gilt widersprücherweise $\Phi(f^*) = \Phi(f) + \epsilon$.

Es ist noch die Kapazität des Schnitts S zu berechnen. Für den Wert des Flusses f gilt nach dem Beweis des Lemmas 22.13

$$\Phi(f) = \sum_{\substack{vw \in E \\ v \in S, w \in \overline{S}}} f(vw) - \sum_{\substack{uv \in E \\ v \in S, u \in \overline{S}}} f(uv). \tag{22.12}$$

Für den Schnitt S gilt aber

- $\kappa(vw) = f(vw)$, falls $vw \in E$ mit $v \in S$ und $w \in \overline{S}$,
- $f(uv) = 0$, falls $uv \in E$ mit $v \in S$ und $u \in \overline{S}$.

Also folgt mit (22.12) sofort $\Phi(f) = \kappa(S)$. □

Der Beweis liefert einen iterativen Algorithmus zur Berechnung eines maximalen Flusses auf einem Flussnetz. Beginnend mit dem Nullfluss wird in jedem Schritt ein flussvergrößernder Weg von der Quelle zur Senke konstruiert, mit dessen Hilfe der momentane Fluss erhöht wird. Der Algorithmus wird durch zwei Routinen implementiert, die sich jeweils gegenseitig aufrufen. In der ersten Routine wird ein flussvergrößernder Weg konstruiert, der dann in der zweiten Routine dazu verwendet wird, den Fluss zu vergrößern. Die Knoten v des Flussnetzes werden jeweils markiert mit einem ihrer Vorgängerknoten $\text{pred}(v)$, der Richtung $R(v)$ der Kante, die mit $\text{pred}(v)$ und v inzidiert, und der möglichen Flussverbesserung $\epsilon(v)$. Die Menge M enthält die markierten Knoten, deren Nachbarn noch nicht allesamt markiert sind. Wenn die Markierungsroutine die Senke nicht mehr erreicht, liegt ein maximaler Fluss vor und S ist ein minimaler Schnitt.

Sei ein Flussnetz vorausgesetzt, bei dem alle Kapazitätswerte ganzzahlig sind. Dann terminiert der Algorithmus, weil der momentane Fluss in jedem Schritt um eine positive ganze Zahl vergrößert wird und somit ein Fluss erreicht wird, für den es keinen flussvergrößernden Weg mehr gibt. Dieser Fluss ist dann maximal. Der Algorithmus ist auch auf Flussnetze anwendbar, bei denen alle Kapazitätswerte rational sind. Denn durch Multiplizieren mit dem Hauptnenner der Kapazitätswerte ergibt sich ein äquivalentes Problem, bei dem alle Kapazitätswerte ganzzahlig sind.

Bei Flussnetzen mit irrationalen Kapazitätswerten kann es vorkommen, dass die Verbesserungen $\epsilon > 0$ infinitesimal klein sind und deshalb der Algorithmus nicht terminiert. Der Algorithmus wurde von Edmonds und Karp (1972) so abgeändert, dass der flussvergrößernde Weg über Breitensuche gefunden wird. Dieser modifizierte Algorithmus hat auch bei irrationalen Kapazitätswerten den Zeitbedarf $O(|V| \cdot |E|^2)$.

Beispiel 22.16. Wir wenden den Algorithmus auf das Flussnetz in Abb. 22.4 an. Beginnend mit dem Nullfluss liefert FlussMarkieren das markierte Flussnetz in Abb. 22.6. FlussVergrössern verbessert den Nullfluss entlang des flussvergrößernden Weges $X = (q, c, b, s)$ zu einem Fluss mit dem Wert 3 und FlussMarkieren wird ein zweites Mal aufgerufen (Abb. 22.7). Durch FlussVergrössern wird der Fluss entlang des flussvergrößernden Weges

Algorithmus 22.4 FlussMarkieren(D, κ, q, sf)

Eingabe: Flussnetz (D, κ, q, s) und Fluss f auf Flussnetz
Ausgabe: markierte Knoten, falls Senke erreicht wird, sonst wird minimaler Schnitt S ausgegeben

```
 1: S := {q}
 2: M := {q}
 3: ε(q) := ∞
 4: repeat
 5:    wähle Knoten v ∈ M
 6:    M := M \ {v}
 7:    for all w ∈ V \ S mit vw ∈ E do
 8:       if f(vw) < κ(vw) then
 9:          pred(w) := v
10:          R(w) := ′ →′
11:          ε(w) := min{κ(vw) − f(vw), ε(v)}
12:          S := S ∪ {w}
13:          M := M ∪ {w}
14:       end if
15:    end for
16:    for all u ∈ V \ S mit uv ∈ E do
17:       if f(uv) > 0 then
18:          pred(u) := v
19:          R(u) := ′ ←′
20:          ε(u) := min{f(uv), ε(v)}
21:          S := S ∪ {u}
22:          M := M ∪ {u}
23:       end if
24:    end for
25: until M = ∅ or s ∈ S
26: if M = ∅ then
27:    return S
28: end if
29: if s ∈ S then
30:    FlussVergrössern(D, κ, q, sf)
31: end if
```

$X = (q, a, b, s)$ um den Wert 2 erhöht. Anschließendes FlussMarkieren erreicht die Senke nicht mehr (Abb. 22.8). Der Fluss ist also maximal mit dem Wert 5 und der zugehörige minimale Schnitt ist $S = \{q, a, c\}$.

Satz 22.17. (Ganzzahligkeit) *In einem Netzwerk mit ganzzahligen Kapazitätenwerten gibt es einen ganzzahligen maximalen Fluss.*

Beweis. Im Algorithmus wird, beginnend mit dem Nullfluss, der Fluss in jedem Schritt um eine ganze Zahl $\epsilon > 0$ vergrößert. Also hat der jeweilig vergrößerte Fluss nur ganzzahlige Kantenwerte. Dies gilt insbesondere für den maximalen Fluss. □

Algorithmus 22.5 FlussVergrössern(D, κ, q, s, f)

Eingabe: Flussnetz (D, κ, q, s) und Fluss f auf Flussnetz
Ausgabe: Fluss f vergrößert um ϵ

```
 1: ε := ε(s)
 2: v := s
 3: while v ≠ q do
 4:    if R(v) = ' →' then
 5:       f(pred(v), v) := f(pred(v), v) + ε
 6:    end if
 7:    if R(v) = ' ←' then
 8:       f(v, pred(v)) := f(v, pred(v)) − ε
 9:    end if
10:    v := pred(v)
11: end while
12: FlussMarkieren(D, s, t, κ, f)
```

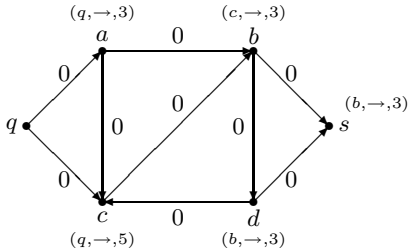


Abb. 22.6. Erstes FlussMarkieren.

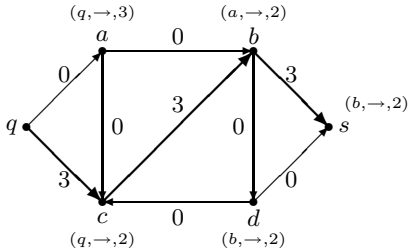


Abb. 22.7. Erstes FlussVergrössern und zweites FlussMarkieren.

22.4 Die Sätze von Hall, König-Egerváry und Menger

In diesem Abschnitt werden eine Reihe kombinatorischer Anwendungen des Satzes von Ford und Fulkerson behandelt.

Der Satz von Menger

Der Satz von Karl Menger (1902-1985) bestimmt die Anzahl der kanten- und knotendisjunkten Wege in einem Digraphen. Sei $D = (V, E)$ ein Digraph und

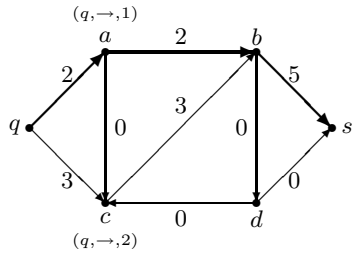


Abb. 22.8. Zweites FlussVergrössern und drittes FlussMarkieren. Die Senke wird nicht mehr erreicht und der Algorithmus terminiert.

seien q und s Knoten in D. Eine Menge E' von Kanten in D heißt eine *q und s trennende Kantenmenge in D*, wenn jeder gerichtete Weg von q nach s mindestens eine Kante aus E' enthält. Eine q und s trennende Kantenmenge E_0 in D heißt *minimal*, wenn für jede q und s trennende Kantenmenge E' in D gilt $|E_0| \leq |E'|$.

Satz 22.18. (Menger, 1927) *Sei $D = (V, E)$ ein Digraph und seien q und s verschiedene Knoten in D. Die Maximalzahl kantendisjunkter gerichteter Wege in D von q nach s ist gleich der Mächtigkeit einer minimalen q und s trennenden Kantenmenge in D.*

Beweis. Wir betrachten ein Flussnetz N auf D, in dem jede Kante die Kapazität 1 besitzt. Nach dem Ganzzahligkeitssatz gibt es einen maximalen Fluss auf N, in dem jede Kante mit 0 oder 1 bewertet ist. Ein solcher Fluss heißt *0-1-Fluss*. Dabei liefern k kantendisjunkte Wege von q nach s in D einen 0-1-Fluss auf N mit dem Wert k, und umgekehrt.

Jeder Schnitt S in N liefert eine q und s trennende Kantenmenge $E' = \{e \in E \mid e^- \in S, e^+ \in \overline{S}\}$ mit der Kapazität $|E'| = \kappa(S)$. Umgekehrt sei E' eine q und s trennende Kantenmenge in D. Sei $S(E')$ die Menge aller Knoten in D, die von q aus auf einem gerichteten Weg erreichbar sind, ohne die Kanten in E' zu benutzen. Die Menge $S(E')$ bildet einen Schnitt in N mit der Kapazität $|E'| = \kappa(S(E'))$. Mit dem Satz von Ford und Fulkerson folgt die Behauptung. □

Beispiel 22.19. Der Digraph in Abb. 22.9 enthält höchstens drei kantendisjunkte Wege von q und s, weil q zu drei Knoten adjazent ist. Andererseits ist jede q und s trennende Kantenmenge mindestens 3-elementig, etwa $\{ad, be, ce\}$ und $\{df, dg, be, ce\}$.

Sei $D = (V, E)$ ein Digraph und seien q und s Knoten in D. Eine Menge V' von Knoten in D heißt eine *q und s trennende Knotenmenge in D*, wenn jeder gerichtete Weg von q nach s in D mindestens einen Knoten aus V' enthält. Eine q und s trennende Knotenmenge V_0 in D heißt *minimal*, wenn für jede q und s trennende Knotenmenge V' in D gilt $|V_0| \leq |V'|$.

Satz 22.20. (Menger) *Sei $D = (V, E)$ ein Digraph und seien q und s nichtadjazente Knoten in D. Die Maximalzahl knotendisjunkter gerichteter Wege in D von q nach s ist gleich der Mächtigkeit einer minimalen q und s trennenden Knotenmenge in D.*

Beweis. Wir ordnen dem Digraphen D einen Digraphen D' zu, in dem jeder Knoten v (außer q und s) durch zwei Knoten v' und v'' und eine Kante $v'v''$ ersetzt wird. Dabei beginnen die Kanten uv in D im neuen Digraphen bei u'' und enden in v' (Abb. 22.10).

Definitionsgemäß entsprechen die knotendisjunkten Wege in D den kantendisjunkten Wegen in D'. Also ist nach Satz 22.18 die Maximalzahl knotendisjunkter gerichteter Wege von q nach s in D gleich der Mächtigkeit einer minimalen q und s trennenden Kantenmenge in D'. In einer q und s trennenden Kantenmenge in D' brauchen nur Kanten der Form $v'v''$ berücksichtigt zu werden, weil eine Kante der Form $u''v'$ stets durch $u'u''$ ersetzt werden kann. Derartige q und s trennenden Kantenmengen in D' entsprechen den q und s trennenden Knotenmengen in D. □

Beispiel 22.21. Der Digraph in Abb. 22.9 enthält höchstens zwei knotendisjunkte Wege von q und s, weil jeder solche Weg entweder durch d oder e verläuft, etwa (q, a, d, f, s) und (q, b, e, h, s). Andererseits ist jede q und s trennende Knotenmenge mindestens 2-elementig, etwa $\{a, b, c\}$, $\{d, e\}$ und $\{f, g, h, i\}$.

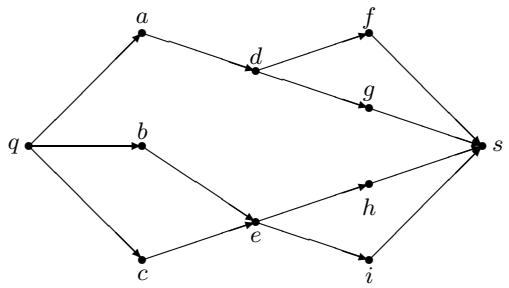


Abb. 22.9. Ein Digraph mit zwei knotendisjunkten und drei kantendisjunkten Wegen von q nach s.

Der Satz von König und Egerváry

Der Satz von König und Egerváry gibt Auskunft über die Mächtigkeit einer maximalen Paarung in einem bipartiten Graphen. Sei G ein Graph. Eine

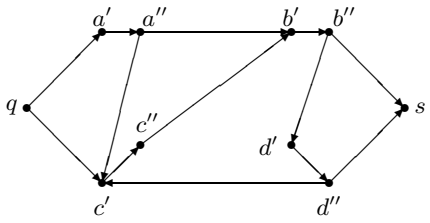


Abb. 22.10. Der dem Digraphen D in Abb. 22.4 zugeordnete Digraph D'.

Menge von Kanten P heißt eine *Paarung* in G, wenn die Kanten in P keinen gemeinsamen Endpunkt haben (Abb. 22.11). Eine Paarung P in G heißt *maximal*, wenn für jede Paarung P' in G gilt $|P'| \leq |P|$.

Eine Menge U von Knoten in G heißt eine *Knotenüberdeckung* von G, wenn für jede Kante uv in G gilt $u \in U$ oder $v \in U$. Eine Knotenüberdeckung U von G heißt *minimal*, wenn für jede Knotenüberdeckung U' von G gilt $|U| \leq |U'|$.

Satz 22.22. (König-Egerváry, 1931) *Sei G ein bipartiter Graph. Die Mächtigkeit einer maximalen Paarung in G ist gleich der Mächtigkeit einer minimalen Knotenüberdeckung von G.*

Beweis. Sei G ein bipartiter Graph mit der 2-Partition $\{V_1, V_2\}$. Wir ordnen dem Graphen G einen Digraphen D zu, in dem zwei Knoten q und s sowie gerichtete Kanten qv, $v \in V_1$, und vs, $v \in V_2$, hinzugefügt werden. Ferner wird jede Kante zwischen V_1 und V_2 von V_1 nach V_2 gerichtet (Abb. 22.12). Definitionsgemäß korrespondiert eine aus k Kanten bestehende Paarung in G zu k knotendisjunkten Wegen in D von q nach s. Ferner entspricht einer Knotenüberdeckung von G eine q und s trennenden Knotenmenge in D. Also folgt mit Satz 22.20 die Behauptung. □

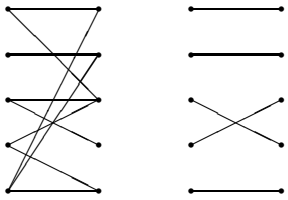


Abb. 22.11. Ein bipartiter Graph mit einer maximalen und vollständigen Paarung.

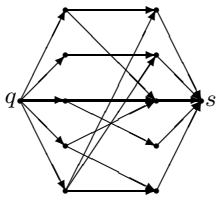


Abb. 22.12. Der zum Digraphen aus Abb. 22.11 gehörende Digraph.

Der Satz von Hall

Der Satz von Philip Hall (1904-1982) ermittelt die Mächtigkeit einer vollständigen Paarung in einem bipartiten Graphen. Sei $G = (V, E)$ ein bipartiter Graph mit 2-Partition $\{V_1, V_2\}$. Eine Paarung P in G heißt *vollständig*, wenn jeder Knoten in V_1 mit einer Kante in P inzidiert (Abb. 22.11).

Für jede Teilmenge U von V_1 bestehe U^+ aus allen Knoten von V_1, die mit irgendeinem Knoten aus V_2 adjazent sind, also

$$U^+ = \{v \in V_2 \mid \exists u \in U\ [uv \in E]\}, \tag{22.13}$$

Satz 22.23. (Hall, 1935) *Sei G ein bipartiter Graph mit einer 2-Partition $\{V_1, V_2\}$, so dass $|V_1| = |V_2|$. Eine vollständige Paarung in G existiert genau dann, wenn für jede Teilmenge U von V_1 gilt $|U^+| \geq |U|$.*

Beweis. Sei P eine vollständige Paarung in G und U eine Teilmenge von V_1. Dann inzidiert jeder Knoten in U mit einer Kante in P und der jeweils andere Endpunkt einer solchen Kante liegt in U^+. Da P eine Paarung ist, gilt $|U| \leq |U^+|$.

Umgekehrt sei $|U_1^+| \geq |U_1|$ für jede Teilmenge U_1 von V_1. Sei U eine Knotenüberdeckung von G mit $U_1 = U \cap V_1$ und $U_2 = U \cap V_2$. Es gibt keine Kante uv mit der Eigenschaft $u \in V_1 \setminus U_1$ und $v \in V_2 \setminus U_2$, weil U sonst keine Knotenüberdeckung von G ist. Also ist $(V_1 \setminus U_1)^+ \subseteq U_2$, mithin aufgrund der Annahme $|U_2| \geq |(V_1 \setminus U_1)^+| \geq |V_1 \setminus U_1|$. Daraus ergibt sich $|U| = |U_1| + |U_2| \geq |U_1| + |V_1 \setminus U_2| = |V_1|$. Somit hat jede Knotenüberdeckung von G mindestens $|V_1|$ Elemente. Jede minimale Knotenüberdeckung von G hat also nach Voraussetzung die Mächtigkeit $|V_1|$. Nach dem Satz von König und Egerváry ist $|V_1|$ die Mächtigkeit einer maximalen Paarung in G. Eine solche Paarung in G ist definitionsgemäß vollständig. □

Dieser Satz wird *Heiratssatz* genannt, weil ein offensichtlicher Zusammenhang mit einem Heiratsproblem besteht.

Selbsttestaufgaben

22.1. Zeige den Satz 22.1.

22.2. Ergänze FLOYD-WARSHALL so, dass nicht nur Abstände, sondern auch kürzeste Wege berechnet werden.

22.3. Beweise die Aussage über die Schleifeninvariante in Satz 22.5.

22.4. Sei (G, ω) ein Wegenetz, in dem je zwei Kanten verschiedene Bewertungen haben. Zeige, dass das Netz genau ein minimales Gerüst besitzt.

22.5. In einem Datennetz werden den Leitungen (Kanten) Wahrscheinlichkeiten für ihr Ausfallen zugewiesen. Gesucht sind zwischen zwei Stationen (Knoten) die Wege, die mit der größten Wahrscheinlichkeit nicht ausfallen. Wie kann dieses Problem auf die Bestimmung kürzester Wege zurückgeführt werden?

22.6. Modifiziere FLOYD-WARSHALL so, dass damit die transitive Hülle einer homogenen Relation berechnet werden kann.

22.7. Beweise die Gl. (22.5).

22.8. Zeige, dass MINSPANNBAUM die Laufzeit $O(|E| \log |E|)$ besitzt.

22.9. Verallgemeinere den Algorithmus von Ford und Fulkerson auf Netze mit mehreren Quellen und Senken.

22.10. Die *Zusammenhangszahl* eines Graphen $G = (V, E)$ ist erklärt durch

$$\kappa(G) = \min\{|F| \mid F \subseteq E, G \setminus F \text{ ist nicht zshgd}\}.$$

Ein Graph G heißt *p-fach zusammenhängend*, wenn $\kappa(G) \geq p$. Zeige, dass ein Graph G genau dann p-fach zusammenhängend ist, wenn je zwei Knoten in G duch mindestens p kantendisjunkte Wege verbunden sind.

22.11. Seien S_1 und S_2 minimale Schnitte in einem Flussnetz N. Zeige, daß auch $S_1 \cap S_2$ ein minimaler Schnitt in N ist.

22.12. In einem Datennetz erfolgt der Datentransport zwischen zwei Stationen q und s. Wie kann festgestellt werden, wie viele Leitungen (Kanten) maximal ausfallen dürfen, damit immer noch eine Datenleitung zwischen q und s funktioniert?

22.13. Fünf Jungen, Alfred, Bert, Claus und Detlef, und fünf Mädchen, Rita, Susi, Thea, Ute und Vera, belegen einen Tanzkurs. Alfred ist befreundet mit Rita, Susi und Ute, Bert mit Susi, Claus mit Susi, Ute sowie Thea, Detlef mit Ute und Vera und schließlich Egon mit Rita, Susi und Vera. Kann jeder Junge ein mit ihm befreundetes Mädchen zum Abschlussball einladen, ohne dass ein Mädchen zwischen zwei Jungen zu wählen braucht?

22.14. Sei A eine endliche nichtleere Menge und $M = (A_1, \ldots, A_n)$ eine Folge von nichtleeren Teilmengen von A. Gesucht wird ein Vertreter a_i aus jeder Teilmenge A_i, so dass verschiedene Teilmengen durch verschiedene Elemente repräsentiert werden. Ein solches Tupel wird *Vertretersystem* von M genannt.

Zeige, dass $M = (A_1, \ldots, A_n)$ ein Vertretersystem genau dann besitzt, wenn gilt für jede Teilmenge I von $\underline{n}$ gilt

$$\left| \bigcup_{i \in I} A_i \right| \geq |I|.$$

23

Kombinatorische Optimierung

Viele Optimierungsaufgaben mit großer wirtschaftlicher Bedeutung sind so komplex, dass eine exakte Lösung trotz der enorm gestiegenen Rechenleistung von Computern nicht möglich ist. Diese Optimierungsaufgaben gehören zu den NP-harten Problemen und sind dadurch gekennzeichnet, dass ihre Lösung exponentiellen Aufwand erfordert. In diesem Kapitel werden Methoden vorgestellt, um derartige Optimierungsprobleme zu lösen. Dazu gehören Backtracking-Verfahren, die den gesamten Lösungsraum durchmustern, und heuristische Verfahren, die Näherungslösungen generieren. Weiterhin wird die Optimalität spezieller Heuristiken, so genannter Greedy-Algorithmen, mithilfe von Matroiden charakterisiert. Schließlich wird das Problem der Knotenfärbung von Graphen untersucht.

23.1 Komplexitätsklassen

Die Laufzeit von Algorithmen wird in Abhängigkeit von der jeweiligen Probleminstanz n gemessen. Um praktische Probleme zu lösen, werden Algorithmen mit *polynomialer Laufzeit* $O(n^d)$, $d \geq 0$, benötigt. Demgegenüber sind Algorithmen mit *exponentieller Laufzeit* $O(c^n)$, $c > 1$, meist unpraktikabel.

Komplexität von Entscheidungsproblemen

Ein *Entscheidungsproblem* erfordert eine Antwort "ja" oder "nein". Entscheidungsprobleme werden in zwei Klassen eingeteilt. Die Klasse P besteht aus allen Entscheidungsproblemen, die in polynomialer Zeit lösbar sind. Die Klasse NP umschließt alle Entscheidungsprobleme mit der Eigenschaft, dass es zu jeder affirmativ beanworteten Probleminstanz einen Beweis für die affirmative Antwort gibt, der in polynomialer Zeit verifizierbar ist. Offensichtlich ist $\mathrm{P} \subseteq \mathrm{NP}$. Es ist wahrscheinlich, dass die Klasse NP größer ist als die Klasse P. Einen Beweis hierfür gibt es nicht.

Beispiel 23.1. Sei $G = (V, E)$ ein Graph. Eine Teilmenge U von V heißt *unabhängig* in G, wenn $uv \notin E$ für alle $u, v \in U$. Wir betrachten das Entscheidungsproblem, zu einem Graphen G und einer natürlichen Zahl K eine unabhängige Menge U in G mit $|U| \geq K$ zu finden. Dieses Problem liegt in NP. Denn eine Teilmenge U von V besitzt $\binom{|U|}{2}$ zweielementige Teilmengen uv mit $u, v \in U$ und mit $n = |V|$ ist $\binom{|U|}{2} = O(n^2)$. Der Test, ob uv in E liegt, kann in konstanter Zeit erfolgen. Also kann in $O(n^2)$ Schritten getestet werden, ob U unabhängig in G ist.

Seien D und D' Entscheidungsprobleme. Eine *polynomiale Transformation* von D auf D' ist ein Algorithmus mit polynomialer Laufzeit, der jede Instanz I des Problems D in eine Instanz I' des Problems D' so transformiert, dass die beiden Antworten von I und I' übereinstimmen. Wenn es eine solche Transformation gibt, schreiben wir $D \propto D'$.

Beispiel 23.2. Sei $G = (V, E)$ ein Graph. Eine Teilmenge $U \subseteq V$ heißt eine *Clique* in G, wenn $uv \in E$ für alle $u, v \in U$. Eine Clique ist also ein vollständiger Teilgraph von G. Wir betrachten das Entscheidungsproblem, zu einem Graphen G und einer natürlichen Zahl K eine Clique U in G mit $|U| \geq K$ zu finden. Dieses Problem gehört zu NP.

Sei $G' = (V, E')$ der zu G *komplementäre Graph*, d. h., $uv \in E'$ genau dann, wenn $uv \notin E$. Die unabhängigen Mengen in G sind genau die Cliquen in G', und umgekehrt. Der Graph G' kann aus G in $O(|V|^2)$ Schritten konstruiert werden. Also ist das Entscheidungsproblem, eine unabhängige Teilmenge mit wenigstens K Elementen in einem Graphen zu finden, polynomial transformierbar auf das Entscheidungsproblem, in einem Graphen eine Clique mit mindestens K Elementen zu finden.

Ein Entscheidungsproblem D heißt *NP-vollständig*, wenn $D \in \mathrm{NP}$ und für jedes Problem $D' \in \mathrm{NP}$ gilt $D' \propto D$.

Satz 23.3. *Sei D ein NP-vollständiges Entscheidungsproblem. Liegt D in P, dann ist $P = NP$.*

Beweis. Sei $D \in \mathrm{P}$ ein NP-vollständiges Problem. Sei A ein polynomialer Algorithmus, der D löst. Sei $D' \in \mathrm{NP}$. Nach Voraussetzung ist $D' \propto D$. Somit existiert ein polynomialer Algorithmus B, der jede Instanz von D' in eine Instanz von D übersetzt. Also lösen die hintereinander ausgeführten Algorithmen A und B in polynomialer Zeit jede Instanz von D'. Folglich ist $D' \in \mathrm{P}$. Daraus folgt die Behauptung. □

Beispiel 23.4. Sei n eine natürliche Zahl. Das *Erfüllbarkeitsproblem* lautet, zu einer n-stelligen Schaltfunktion f eine Belegung $b_1, \ldots, b_n \in \{0, 1\}$ der Variablen von f zu finden, so dass $f(b_1, \ldots, b_n) = 1$. Dieses Problem ist NP-vollständig. Bewiesen hat dies zuerst Stephen A. Cook (1971).

Derzeit sind mehr als eintausend NP-vollständige Probleme bekannt, darunter viele graphentheoretische Probleme.

Komplexität von Optimierungsproblemen

Kombinatorische Probleme treten meist in der Gestalt von Optimierungsproblemen auf. Wir klassifizieren derartige Probleme mit Hilfe der Turing-Reduktion.

Eine *Turing-Reduktion* ordnet einem Problem D ein Problem D' so zu, dass der Lösungsalgorithmus A für D als "Subalgorithmus" im Lösungsalgorithmus B für D' verwendet wird. Dabei soll A genau dann polynomiale Laufzeit haben, wenn B polynomiale Laufzeit besitzt. Eine solche Reduktion wird mit $D' \propto_T D$ bezeichnet. Eine polynomiale Transformation ist eine spezielle Turing-Reduktion, d. h., $D' \propto D$ impliziert $D' \propto_T D$.

Beispiel 23.5. Das zum Entscheidungsproblem, eine Clique mit wenigstens K Elementen in einem Graphen zu finden, gehörende Optimierungsproblem lautet, in einem Graphen eine Clique mit maximaler Mächtigkeit zu finden.

Sei A ein Algorithmus, der obiges Entscheidungsproblem löst. Dieser Algorithmus wird in eine Laufschleife eingebettet, die in jedem Schritt $K = n, \ldots, 1$ eine Instanz $A(G, K)$ des Entscheidungsproblems bearbeitet. Sobald eine Instanz $A(G, K)$ "ja" liefert, ist eine maximale Clique gefunden und somit das zugehörige Optimierungsproblem gelöst.

Das Analogon von NP-vollständigen Entscheidungsproblemen sind NP-harte Optimierungsprobleme. Ein Problem D heißt *NP-hart*, wenn es ein NP-vollständiges Problem D' gibt, so dass $D' \propto_T D$. Nach 23.5 ist das Optimierungsproblem, eine maximale Clique in einem Graphen zu finden, NP-hart. Im Allgemeinen sind die Optimierungsvarianten von NP-vollständigen Entscheidungsproblemen NP-hart.

23.2 Backtracking-Algorithmen

Kombinatorische Optimierung

Ein kombinatorischen Optimierungsproblem basiert auf einer endlichen Menge. Eine solche Menge wird im Praktikerjargon *Universum* genant und die Elemente eines Universums heißen *Lösungen*. Die Menge aller Lösungen ist in zulässige und nichtzulässige Lösungen unterteilt. Auf einem Universum X ist ferner eine ganzzahlige *Zielfunktion* $f : X \to \mathbb{Z}$ definiert. Gefragt wird nach einer zulässigen Lösung $x^* \in X$ mit maximalem Wert, d. h., $f(x^*) \geq f(x)$ für alle zulässigen Lösungen $x \in X$. Eine solche Lösung x^* heißt *optimal*. Ein *kombinatorisches Optimierungsproblem* hat also folgende Gestalt

$$\begin{array}{ll} & \max f(x). \\ s.d. & x \in X \\ & x \text{ ist zulässig} \end{array} \tag{23.1}$$

Maximierungs- und Minimierungsprobleme sind gleichwertig, weil das Maximieren von f dem Minimieren von $-f$ entspricht, genauer

$$\max\{f(x) \mid x \in X\} = -\min\{-f(x) \mid x \in X\}. \tag{23.2}$$

Beispiel 23.6. Sei $G = \{1, \ldots, n\}$ eine Menge von Gegenständen. Jeder Gegenstand i besitze einen ganzzahligen Wert $f_i \geq 1$ und eine ganzzahlige Größe $g_i \geq 1$. Ferner sei K_0 eine natürliche Zahl, genannt *Kapazität*.

Ein *Rucksack* ist eine Teilmenge G' von G. Der Wert eines Rucksacks G' ist die Summe der Werte aller gepackten Gegenstände $\sum_{i \in G'} f_i$ und die Größe von G' ist die Summe der Größen aller gepackten Gegenstände $\sum_{i \in G'} g_i$. Das *Rucksack-Problem* lautet, einen Rucksack mit maximalem Wert zu finden, so dass seine Größe die Kapazität nicht überschreitet. Das Rucksack-Problem ist NP-hart.

Wir ordnen jedem Rucksack G' seinen charakteristischen Vektor $x \in \{0,1\}^n$ zu, d. h., $x_i = 1$, falls $i \in G'$, und $x_i = 0$, falls $i \notin G'$. Dann besitzt ein Rucksack x den Wert $f(x) = \sum_i f_i x_i$ und die Größe $g(x) = \sum_i g_i x_i$. Das Rucksack-Problem lautet also

$$\begin{array}{c} \max f(x). \\ s.d.\ x \in \{0,1\}^n \\ g(x) \leq K_0 \end{array} \tag{23.3}$$

Backtracking-Verfahren

Ein *Backtracking-Algorithmus* ist eine rekursive Prozedur, die alle Lösungen eines kombinatorischen Optimierungsproblems schrittweise erzeugt. Backtracking-Algorithmen gehören zu den exhaustiven Suchmethoden.

RUCKSACK-BACKTRACK1 ist ein Backtracking-Algorithmus für das Rucksack-Problem. Der Algorithmus startet mit dem so genannten leeren Tupel $x = ( )$ und erzeugt alle binären n-Tupel in lexikographischer Reihenfolge. Dabei ist $aktf$ der Wert des aktuellen n-Tupels x, $optx$ die momentan beste, zulässige Lösung und $optf$ der Wert von $optx$. Die rekursiven Aufrufe der Routine RUCKSACK-BACKTRACK1 werden anhand eines in der Tiefe durchlaufenen *Aufrufbaums* veranschaulicht.

Beispiel 23.7. Wir betrachten ein Rucksack-Problem mit fünf Gegenständen und Kapazität $K_0 = 35$. Die Gegenstände besitzen die Größen 14, 15, 11, 10 und 12 sowie die Werte 26, 27, 18, 16 und 19. Den Aufrufbaum von RUCKSACK-BACKTRACK1 zeigt die Abb. 23.1. Die Blätter dieses Baumes entsprechen den Rucksäcken. An den Blättern ist das Gewicht des jeweiligen Rucksacks angegeben. Eine optimale Lösung ist der Rucksack $x = (1, 0, 1, 1, 0)$ mit $f(x) = 60$ und $g(x) = 35$.

Algorithmus 23.1 RUCKSACK-BACKTRACK1$(x_1, \ldots, x_k, k)$

Eingabe: $x_1, \ldots, x_k \in \{0, 1\}$, $k \geq 0$

```
 1: global optf, optx
 2: if k = n then
 3:    if g_1x_1 + ... + g_nx_n ≤ K_0 then
 4:       aktf := f_1x_1 + ... + f_nx_n
 5:       if aktf > optf then
 6:          optf := aktf
 7:          optx := (x_1, ..., x_n)
 8:       end if
 9:    end if
10: else
11:    x_{k+1} := 0
12:    RUCKSACK-BACKTRACK1(x_1, ..., x_k, x_{k+1}, k + 1)
13:    x_{k+1} := 1
14:    RUCKSACK-BACKTRACK1(x_1, ..., x_k, x_{k+1}, k + 1)
15: end if
```

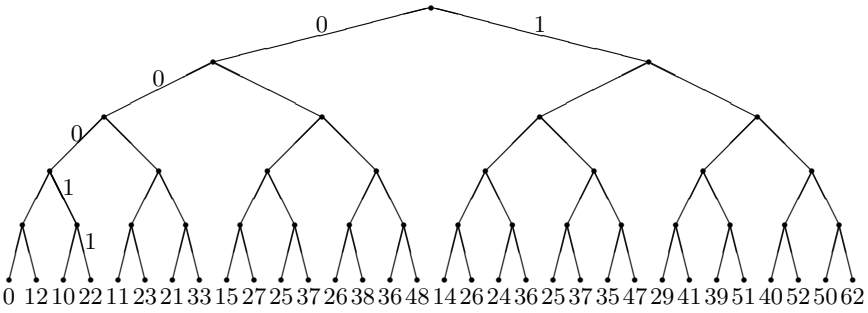


Abb. 23.1. Ein Aufruf-Baum von RUCKSACK-BACKTRACK1 für das Rucksack-Problem in 23.7. Die Blätter sind mit den Größen der Rucksäcke bewertet.

Abschneiden von Teilbäumen

Ein Aufrufbaum kann durch gezieltes Abschneiden von Teilbäumen verkleinert werden. Die inneren Knoten eines Aufrufbaums werden *Teillösungen* genannt. Den Teillösungen entsprechen im Rucksack-Problem teilweise gepackte Rucksäcke $x = (x_1, \ldots, x_k)$, $0 \leq k \leq n$. Jede Teillösung x hat ebenfalls einen Wert $f(x) = \sum_i f_i x_i$ und eine Größe $g(x) = \sum_i g_i x_i$. Eine Teillösung x heißt *zulässig*, wenn $g(x) \leq K_0$. Im Aufrufbaum kann ein Teilbaum weggeschnitten werden, wenn seine Wurzel nicht zulässig ist. Dann sind nämlich alle Teillösungen in diesem Teilbaum ebenfalls nicht zulässig. Aus RUCKSACK-BACKTRACK1 erhalten wir auf diese Weise RUCKSACK-BACKTRACK2.

Algorithmus 23.2 RUCKSACK-BACKTRACK2$(x_1, \ldots, x_k, k)$

Eingabe: $x_1, \ldots, x_k \in \{0, 1\}$, $k \geq 0$

```
 1: global optf, optx
 2: if k = n then
 3:    if g_1x_1 + ... + g_nx_n ≤ K_0 then
 4:       aktf := f_1x_1 + ... + f_nx_n
 5:       if aktf > optf then
 6:          optf := aktf
 7:          optx := (x_1, ..., x_n)
 8:       end if
 9:    end if
10: else
11:    x_{k+1} := 0
12:    RUCKSACK-BACKTRACK2(x_1, ..., x_k, x_{k+1}, k + 1)
13:    x_{k+1} := 1
14:    if g_1x_1 ... + g_kx_k + g_{k+1} ≤ K_0 then
15:       RUCKSACK-BACKTRACK2(x_1, ..., x_k, x_{k+1}, k + 1) {Abschneiden}
16:    end if
17: end if
```

Bounding-Funktionen

Sei $x = (x_1, \ldots, x_k)$ eine Teillösung eines Aufrufbaums und $P(x)$ der maximale Wert aller zulässigen Lösungen, die in dem Teilbaum mit der Wurzel x liegen. Eine *Bounding-Funktion* ist eine reellwertige Abbildung B auf den Knoten eines Aufrufbaums derart, dass für jede zulässige Teillösung x gilt $B(x) \geq P(x)$. Ein Aufrufbaum kann mithilfe einer Bounding-Funktion verkleinert werden. Sei x eine zulässige Teillösung und $optf$ der momentan optimale Wert. Im Falle $B(x) \leq optf$ gilt für jede zulässige Lösung y im Teilbaum mit der Wurzel x

$$f(y) \leq B(x) \leq optf. \tag{23.4}$$

Der Teilbaum mit der Wurzel x kann also die momentan optimale Lösung nicht verbessern. Er kann deshalb weggeschnitten werden.

Eine Bounding-Funktion für das Rucksack-Problem erhalten wir anhand des rationalen Rucksack-Problems. Ein Rucksack $x = (x_1, \ldots, x_n)$ heißt *rational*, wenn er rationale Komponenten x_i mit $0 \leq x_i \leq 1$ besitzt. Ein rationaler Rucksack x hat den Wert $f(x) = \sum_i f_i x_i$ und die Größe $g(x) = \sum_i g_i x_i$. Das *rationale Rucksack-Problem* lautet

$$\begin{array}{c} \max f(x). \\ s.d.\ x \text{ ist rational} \\ g(x) \leq K_0 \end{array} \tag{23.5}$$

Seien die Gegenstände so nummeriert, dass $f_1/g_1 \geq f_2/g_2 \geq \ldots \geq f_n/g_n$. Ein optimaler rationaler Rucksack x^* lässt sich dann direkt angeben. Im Falle

$g_1 + \ldots + g_n \leq K_0$ setze $x^* = (1, \ldots, 1)$. Andernfalls gibt es einen Index l mit $g_1 + \ldots + g_l \leq K_0$ und $g_1 + \ldots + g_l + g_{l+1} > K_0$. Dann setze $x_1^* = \ldots = x_l^* = 1$, $x_{l+1}^* = \frac{K_0 - (g_1 + \ldots + g_l)}{g_{l+1}}$ und $x_{l+2}^* = \ldots = x_n^* = 0$. Der optimale Rucksack x^* wird in $O(n)$ Schritten berechnet.

Beispiel 23.8. Der optimale rationale Rucksack des Rucksack-Problems in 23.7 ist $x = (1, 1, 0.545455, 0, 0)$ mit $g(x) = 35$ und $f(x) = 62.818182$.

Das (ganzzahlige) Rucksack-Problem ist ein Spezialfall des rationalen Rucksack-Problems. Also ist der Wert jedes zulässigen Rucksacks in einem Teilbaum mit der Wurzel $x = (x_1, \ldots, x_k)$ nach oben beschränkt durch

$$B(x) = f(x) + \textsc{Rucksack-Rat}(f_{k+1}, \ldots, f_n, g_{k+1}, \ldots, g_n, K_0 - g(x)),$$

wobei der zweite Term den Wert eines optimalen rationalen Rucksacks für die letzten $n - k$ Gegenstände angibt. Somit ist $B(x)$ eine Bounding-Funktion für das Rucksack-Problem, mit der sich aus Rucksack-Backtrack2 der Algorithmus Rucksack-Backtrack3 ergibt.

Algorithmus 23.3 Rucksack-Backtrack3$(x_1, \ldots, x_k, k)$

Eingabe: $x_1, \ldots, x_k \in \{0, 1\}$, $k \geq 0$

```
 1: global optf, optx
 2: if k = n then
 3:    if g_1 x_1 + ... + g_n x_n ≤ K_0 then
 4:       aktf := f_1 x_1 + ... + f_n x_n
 5:       if aktf > optf then
 6:          optf := aktf
 7:          optx := (x_1, ..., x_n)
 8:       end if
 9:    end if
10: else
11:    B := (f_1 x_1 + ... + f_k x_k) + Rucksack-Rat(f_{k+1}, ..., f_n, g_{k+1}, ..., g_n, K_0 −
       Σ_{i=1}^{k} g_i x_i) {Bounding-Funktion}
12:    if B ≤ optf then
13:       return
14:    end if
15:    x_{k+1} := 0
16:    Rucksack-Backtrack3(x_1, ..., x_k, x_{k+1}, k + 1)
17:    x_{k+1} := 1
18:    if g_1 x_1 ... + g_k x_k + g_{k+1} ≤ K_0 then
19:       Rucksack-Backtrack3(x_1, ..., x_k, x_{k+1}, k + 1) {Abschneiden}
20:    end if
21: end if
```

Das Rundreise-Problem

Sei $K_n = (V, E)$ der vollständige Graph mit n Knoten und $f : E \to \mathbb{N}$ eine Kostenfunktion auf den Kanten von K_n. Eine *Rundreise* oder ein *hamiltonscher Kreis* in K_n ist ein einfacher Kreis in K_n, der alle Knoten in K_n enthält. Die *Länge einer Rundreise* $x = (x_1, \ldots, x_n, x_1)$ in K_n ist

$$f(x) = \sum_{i=1}^{n} f(x_i x_{i+1}) + f(x_n x_1). \tag{23.6}$$

Das *Rundreise-Problem* für den bewerteten Graphen K_n besteht darin, eine Rundreise in K_n mit minimaler Länge zu finden. Das Rundreise-Problem ist NP-hart und kann als Problem eines Handlungsreisenden oder Bestückungsproblem für Leiterplatten interpretiert werden.

Eine Rundreise $(x_1, \ldots, x_n, x_1)$ kann als eine Permutation $x = (x_1, \ldots, x_n)$ der Knotenmenge von K_n dargestellt werden. Jede zyklische Verschiebung einer solchen Permutation stellt dieselbe Rundreise dar. RUNDREISE-BACKTRACK ist ein Backtracking-Algorithmus für das Rundreise-Problem. Dieser Algo-

Algorithmus 23.4 RUNDREISE-BACKTRACK$(x_1, \ldots, x_k, k)$

Eingabe: Graph $G = (V, E)$, $x_1, \ldots, x_k \in V$, $k \geq 0$

```
 1: global optf, optx
 2: if k = n then
 3:    aktf := f(x_1 x_2) + ... + f(x_{n-1} x_n) + f(x_n x_1)
 4:    if aktf < optf then
 5:       optf := aktf
 6:       optx := (x_1, ..., x_n)
 7:    end if
 8: else
 9:    for all x_{k+1} ∉ {x_1, ..., x_k} do
10:       RUNDREISE-BACKTRACK(x_1, ..., x_k, x_{k+1}, k + 1)
11:    end for
12: end if
```

rithmus wird mit dem leeren Tupel $x = (\,)$ gestartet und erzeugt alle Permutationen der Knotenmenge von K_n.

Das Rundreise-Problem ist ein Minimierungsproblem. Also muss eine Bounding-Funktion $B(x)$ für das Rundreise-Problem eine untere Schranke für alle zulässigen Lösungen liefern, die eine zulässige Teillösung x enthalten. Sei $x = (x_1, \ldots, x_k)$ eine zulässige Teillösung. Wir betrachten eine zulässige Lösung $y = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_n, x_1)$ des Rundreise-Problems, die x fortsetzt. Dann ist $(x_k, y_{k+1}, \ldots, y_n, x_1)$ ein Weg in demjenigen Teilgraphen $G = G(x)$ von K_n, der von $V \setminus \{x_2, \ldots, x_{k-1}\}$ aufgespannt wird. Ein Weg, der alle Knoten eines Graphen enthält, ist aber ein spezieller Spannbaum.

Damit erhalten wir folgende Bounding-Funktion für das Rundreise-Problem

$$B(x) = \sum_{i=1}^{k-1} f(x_i x_{i+1}) + \textsc{MinSpannbaum}(G(x), f).$$

23.3 Heuristische Algorithmen

Backtracking-Algorithmen liefern alle optimalen Lösungen eines kombinatorischen Optimierungsproblems. Hat das Optimierungsproblems jedoch einen sehr großen Suchraum, dann sind Backtracking-Algorithmen zu aufwendig. An ihre Stelle treten heuristische Algorithmen, die ein kombinatorisches Optimierungsproblem näherungsweise lösen. Die Güte eines heuristischen Algorithmus', d. h., wie weit die Näherungslösung von der optimalen Lösung entfernt ist, lässt sich im Allgemeinen nicht abschätzen.

Ein *heuristischer Algorithmus* für ein kombinatorisches Optimierungsproblem wird durch eine Heuristik festgelegt, die wiederum auf einer Nachbarschaft basiert. Eine *Nachbarschaft* für ein Universum X ist eine Abbildung $N : X \to P(X)$, die jeder Lösung $x \in X$ eine Menge benachbarter Lösungen $N(x)$ zuordnet. Benachbarte Lösungen sollen auf eine vom jeweiligen Problem abhängige Weise ähnlich sein.

Beispiele 23.9. Im Rucksack-Problem werden zwei Rucksäcke als *benachbart* angesehen, wenn sie sich um einen Gegenstand unterscheiden. D. h., die Nachbarschaft eines Rucksacks x wird durch den Hamming-Abstand auf $X = \{0,1\}^n$ definiert durch $N(x) = \{y \in X \mid d(x,y) = 1\}$.

Im Rundreise-Problem werden zwei Rundreisen als *benachbart* definiert, wenn eine Rundreise in die andere durch einen *Lin-2-Opt-Schritt* transformiert werden kann (Abb. 23.2).

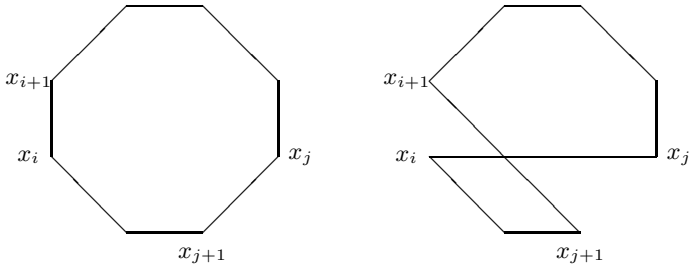


Abb. 23.2. Ein Lin-2-Opt-Schritt.

Eine *Heuristik* für ein kombinatorisches Optimierungsproblem ist ein Algorithmus H_N, der zu jeder zulässigen Lösung x eine benachbarte, zulässige Lösung y ermittelt. Wenn keine benachbarte, zulässige Lösung existiert, dann terminiert der Algorithmus mit der Ausgabe `fail`.

HeuristischeSuche ist ein generischer, heuristischer Algorithmus, der typischerweise als iterative Prozedur implementiert ist. Der Algorithmus ge-

Algorithmus 23.5 HeuristischeSuche(X, f, H_N, I)

Eingabe: Universum X, Zielfunktion f, Heuristik H_N, natürliche Zahl I
Ausgabe: lokal opimale Lösung x^*

```
 1: i := 1
 2: wähle zulässige Lösung x ∈ X{Startpunkt}
 3: x* := x
 4: while i ≤ I do
 5:    y := H_N(x)
 6:    if y ≠ fail then
 7:       x := y
 8:       if f(x) > f(x*) then
 9:          x* := x
10:       end if
11:    else
12:       return x* {vorzeitiges Ende}
13:    end if
14:    i := i + 1
15: end while
16: return x*
```

neriert eine Folge zulässiger Lösungen, in der konsekutive Lösungen benachbart sind. Die resultierende *Näherungslösung* x^* hängt meist vom *Startpunkt* ab. Eine zulässige Lösung $x^* \in X$ heißt ein *lokales Maximum* des Optimierungsproblems (23.1), wenn $f(x^*) \geq f(x)$ für alle zulässigen $x \in N(x^*)$. Eine zulässige Lösung $x^* \in X$ heißt ein *globales Maximum*, wenn $f(x^*) \geq f(x)$ für alle zulässigen $x \in X$. Kombinatorische Optimierungsprobleme besitzen in der Regel sehr viele lokale Optima.

Die Bergauf-Methode

Die *Bergauf-Methode* ist ein heuristisches Verfahren, das vergleichbar ist mit dem Vorgehen eines Bergsteigers, der dem Gipfel zustrebt, ohne zwischenzeitlich abzusteigen. In der Heuristik der Bergauf-Methode wird zu einer zulässigen Lösung x eine zulässige Lösung $y \in N(x)$ mit höherem Wert $f(y) > f(x)$ konstruiert. Die Bergauf-Methode wird *Methode des steilsten Aufstiegs* genannt, wenn in jeder Nachbarschaft stets eine zulässige Lösung mit maximalem Wert gewählt wird.

Algorithmus 23.6 HillClimbing(X, f, H_N)

Eingabe: Universum X, Zielfunktion f, Heuristik H_N
Ausgabe: lokal opimale Lösung x^*

```
 1: wähle zulässige Lösung x ∈ X{Startpunkt}
 2: x* := x
 3: searching := true
 4: while searching do
 5:    y := H_N(x)
 6:    if y ≠ fail then
 7:       x := y
 8:       if f(x) > f(x*) then
 9:          x* := x
10:       end if
11:    else
12:       searching := false
13:    end if
14: end while
15: return x*
```

HillClimbing ist ein generischer Bergauf-Algorithmus. Ein Bergauf-Algorithmus liefert typischerweise ein vom Startwert abhängiges, lokales Maximum. Die Bergauf-Methode gehört deshalb zu den lokalen Optimierungsverfahren. Derartige Verfahren eignen sich besonders bei Problemen, die nur ein Optimum besitzen. Konvexe Optimierungsprobleme haben diese Eigenschaft.

Beispiel 23.10. Wir entwickeln ein Bergauf-Verfahren für das Rucksack-Problem. Die Nachbarschaft sei wie in 23.9 festgelegt. Die Heuristik H_N wird wie folgt definiert: Sei x ein zulässiger Rucksack. Sei $y \in N(x)$ ein zufälliger Rucksack, der anhand einer Zufallszahl $j \in \{1, \ldots, n\}$ erzeugt wird

$$y_i = \begin{cases} x_i & \text{falls } i \neq j \\ 1 - x_i & \text{falls } i = j. \end{cases}$$

Die beiden Rucksäcke x und y unterscheiden sich nur im i-ten Gegenstand und sind deshalb benachbart. Für die Größe von y gilt

$$g(y) = \begin{cases} g(x) + g_j & \text{falls } x_j = 0 \\ g(x) - g_j & \text{falls } x_j = 1. \end{cases}$$

Der Rucksack y ist zulässig, wenn entweder $x_j = 1$ oder $x_j = 0$ und $g(x)+g_j \leq K_0$. Die Heuristik gibt `fail` aus, wenn $x_j = 0$ und $g(x) + g_j > K_0$. Um eine vorzeitige `fail`-Ausgabe zu vermeiden, wird die Heuristik $H_N(x)$ so erweitert, dass sie zu einem gegebenen Rucksack x mehrere zufällige Rucksäcke probiert, bis sich ein zulässiger Rucksack ergibt. Der Algorithmus wird mit dem leeren Rucksack $(0, \ldots, 0)$ gestartet.

Simuliertes Ausglühen

Unter *Ausglühen* wird ein industrieller Prozess verstanden, bei dem ein Festkörper erhitzt und abgekühlt wird. Bei schlagartigem Abkühlen sind die Atome sehr unregelmäßig angeordnet, so dass in dieser Struktur noch eine relativ hohe potentielle Energie steckt. Hingegen entsteht bei langsamem Abkühlen eine gleichmäßige Gitterstruktur, deren potentielle Energie sehr niedrig ist. *Simuliertes Ausglühen* (engl. Simulated Annealing) ist ein heuristisches Verfahren, das auf dem Vorgang des Ausglühens beruht und von Kirkpatrick *et al.* (1983) eingeführt wurde. Simuliertes Ausglühen basiert auf folgender Heuristik: Sei x eine zulässige Lösung. Wähle eine zulässige Lösung $y \in N(x)$. Ist $f(y) > f(x)$, dann wird x durch y ersetzt. Andernfalls wird eine Zufallszahl $r \in [0,1]$ mit dem Wert $e^{(f(y)-f(x))/T}$ verglichen. Im Falle $r < e^{(f(y)-f(x))/T}$ wird x durch y substituiert. Ansonsten wird entweder `fail` ausgegeben oder ein neues $y \in N(x)$ gesucht.

Simuliertes Ausglühen gestattet neben einer *Aufwärtsbewegung* wie in der Bergauf-Methode eine *Abwärtsbewegung*, die von der momentanen *Temperatur* T abhängt. Bei hoher Temperatur ist der Faktor $e^{(f(y)-f(x))/T}$ nahe bei 1 und somit eine Abwärtsbewegung sehr wahrscheinlich, während bei niedriger Temperatur $e^{(f(y)-f(x))/T}$ nahe bei 0 liegt und deshalb eine Abwärtsbewegung fast unmöglich ist. Mithilfe einer Abwärtsbewegung kann ein lokales Optimum wieder verlassen werden, was bei der Bergauf-Methode nicht möglich ist. Simuliertes Ausglühen zählt deshalb zu den globalen Optimierungsverfahren.

Die Temperatur wird nach einem *Abkühlungsplan* gesenkt. Die *Anfangstemperatur* T_0 wird so hoch gewählt, dass Abwärtsbewegungen mit hoher Wahrscheinlichkeit möglich sind. Die Temperatur wird in jeder Iteration um einen prozentualen Anteil gesenkt, bis eine Endtemperatur T_f erreicht wird. Unsere Überlegungen sind in einem generischer Algorithmus für simuliertes Ausglühen, SimulatedAnnealing, zusammengefasst.

Beispiel 23.11. Sei $G = (V, E)$ ein Graph mit einer Kostenfunktion $f : E \to \mathbb{N}_0$. Eine *uniforme Partition* von G ist eine 2-Partition $\{V_1, V_2\}$ von V mit $|V_1| = |V_2|$. Die *Kosten* einer uniformen Partition $\{V_1, V_2\}$ von G sind

$$f(V_1, V_2) = \sum_{\substack{uv \in E \\ u \in V_1, v \in V_2}} f(u, v).$$

Das Optimierungsproblem für uniforme Graph-Partitionen lautet: Finde zu einem Graphen $G = (V, E)$ mit gerader Knotenanzahl und Kostenfunktion $f : E \to \mathbb{N}_0$ eine uniforme Partition mit minimalen Kosten. Dieses Problem ist NP-hart.

Um dieses Optimierungsproblem durch simuliertes Ausglühen zu beschreiben, verwenden wir als Universum X die Menge aller 2-Partitionen $\{V_1, V_2\}$ von V mit $|V_1| = |V_2|$. Die Nachbarschaft von $\{V_1, V_2\}$ sollen alle Partitionen bilden, die aus $\{V_1, V_2\}$ durch Vertauschen zweier Knoten $v_1 \in V_1$ und $v_2 \in V_2$

Algorithmus 23.7 SIMULATEDANNEALING$(X, f, H_N, T_0, T_f, \alpha)$

Eingabe: Universum X, Zielfunktion f, Heuristik H_N, Anfangstemperatur T_0, Endtemperatur T_f, Dekrement α
Ausgabe: lokal optimale Lösung x^*

```
 1: T := T_0
 2: wähle zulässige Lösung x ∈ X
 3: x* := x
 4: while T ≥ T_f do
 5:    y := H_N(x)
 6:    if y ≠ fail then
 7:       if f(y) > f(x) then
 8:          x := y {Aufwärtsbewegung}
 9:          if f(x) > f(x*) then
10:             x* := x;
11:          end if
12:       else
13:          r := random(0,1) {Abwärtsbewegung}
14:          if r < e^((f(y)-f(x))/T) then
15:             x := y
16:          end if
17:       end if
18:    else
19:       return x* {vorzeitiges Ende}
20:    end if
21:    T := αT {α = 0.99}
22: end while
23: return x*
```

entstehen. Der *Gewinn*, der durch das Vertauschen von v_1 mit v_2 entsteht, ist definiert durch

$$F(v_1, v_2, V_1, V_2) = f(V_1, V_2) - f([V_1 \setminus \{v_1\}] \cup \{v_2\}, [V_2 \setminus \{v_2\}] \cup \{v_1\}).$$

In der Heuristik soll gemäß steilstem Aufstieg zu einer Partition eine benachbarte Partition mit maximalem Gewinn geliefert werden.

Der Graph in Abb. 23.3 besitzt die Partition $\{\{a, b, c\}, \{d, e, f\}\}$ mit den Kosten $1 + 1 + 3 + 2 + 0 = 7$. Ihre benachbarten Partitionen sind

$$\begin{array}{l}\{\{a, b, d\}, \{c, e, f\}\}, \{\{a, b, e\}, \{c, d, f\}\},\\ \{\{a, b, f\}, \{c, d, e\}\}, \{\{a, d, c\}, \{b, e, f\}\},\\ \{\{a, c, e\}, \{b, d, f\}\}, \{\{a, c, f\}, \{b, d, e\}\},\\ \{\{b, c, d\}, \{a, e, f\}\}, \{\{b, c, e\}, \{a, d, f\}\},\\ \{\{b, c, f\}, \{a, d, e\}\}.\end{array}$$

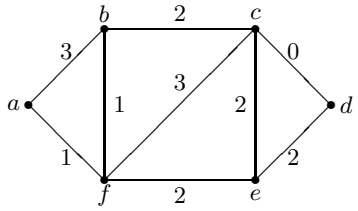


Abb. 23.3. Ein kantenbewerteter Graph.

Genetische Algorithmen

Genetische Algorithmen arbeiten nach dem Vorbild der biologischen Evolution und gehören zu den globalen Optimierungsmethoden. Die Grundmuster und Begriffswelt sind der Biologie entnommen. Ein *genetischer Algorithmus* verwaltet eine Menge von Lösungen fester Mächtigkeit, genannt *Population.* Eine Population kann aus zulässigen und nichtzulässigen Lösungen (höhere und niedere Individuen) bestehen. Ein genetischer Algorithmus arbeitet auf folgende Weise:

- Initialisierung: Wähle eine Population P aus N Individuen.
- Iteration:
  - Selektion: Wähle N Individuen aus der Population P entsprechend ihrer Fitness.
  - Mutation: Ersetze jedes Individuum der Population P durch ein benachbartes Individuum.
  - Kreuzung: Paare die Individuen der Population P und kreuze jedes Paar, sodass neue Individuen entstehen.

Beispiel 23.12. Wir entwickeln einen genetischen Algorithmus für das Rundreise-Problem. Die Nachbarschaft sei wie in 23.9 festgelegt.

- *Selektion:* Jeder Rundreise x in K_n wird ein so genanntes *Boltzmann-Gewicht* $\exp(-\beta f(x))$, $\beta > 0$, und damit eine *Boltzmann-Wahrscheinlichkeit* zugeordnet

$$p(x) = \frac{\exp(-\beta f(x))}{\sum_{y \in P} \exp(-\beta f(y))}.$$

  Je kürzer eine Rundreise ist, desto höher ist ihre Boltzmann-Wahrscheinlichkeit. Aus der Population P werden N Individuen gemäß ihrer Boltzmann-Wahrscheinlichkeit ausgewählt.
- *Mutation* oder *Heuristik:* Eine Rundreise x wird durch eine zufällige Rundreise $x^{(ij)} \in N(x)$ ersetzt, die aus x durch einen Lin-2-Opt-Schritt anhand zweier Zufallszahlen $i, j \in \{1, \ldots, n\}$ erzeugt wird.

- *Kreuzung:* Zwei Rundreisen $x = (x_1, \ldots, x_n)$ und $y = (y_1, \ldots, y_n)$ werden zufällig gekreuzt, indem eine Zufallszahl $j \in \{1, \ldots, n\}$ generiert wird und damit zwei Nachkommen gebildet werden

$$u = (x_1, \ldots, x_j, y_{j+1}, \ldots, y_n) \quad \text{und} \quad v = (y_1, \ldots, y_j, x_{j+1}, \ldots, x_n).$$

  Die Zahl j ist so zu wählen, dass u und v wiederum Rundreisen sind.

23.4 Greedy-Algorithmen und Matroide

Bergauf-Algorithmen und der Algorithmus von Kruskal sind so genannte Greedy-Algorithmen. Derartige Algorithmen wählen in jedem Schritt eine optimale Verbesserung und landen ob ihrer Kurzsichtigkeit oft in einem lokalen Optimum. Wir wollen der Frage nachgehen, unter welchen Umständen Greedy-Algorithmen ein Optimierungsproblem lösen.

Greedy-Algorithmen

Sei A eine endliche nichtleere Menge, sei M ein aus Teilmengen von A bestehendes Mengensystem mit $\emptyset \in M$ und sei $\omega : A \to \mathbb{R}_0^+$ eine Kostenfunktion auf A. Gesucht wird ein Element T des Mengensystems M mit maximalen Kosten

$$\omega(T) = \max\{\omega(T') \mid T' \in M\}. \tag{23.7}$$

Dabei sind die *Kosten* einer Teilmenge T von A gleich die Summe der Kosten ihrer Elemente

$$\omega(T) = \sum_{t \in T} \omega(t). \tag{23.8}$$

Greedy ist ein Greedy-Algorithmus für dieses Problem. Algorithmus 21.1

Greedy$(A, M, \cdot)$ heißt *optimal*, wenn die für *jede* Kostenfunktion $\omega : A \to \mathbb{R}_0^+$ berechnete Lösung $T = T(\omega)$ optimal ist, d. h.

$$\omega(T) = \max\{\omega(T') \mid T' \in M\}. \tag{23.9}$$

Wir zeigen, dass optimale Greedy-Algorithmen durch Matroide charakterisierbar sind.

Algorithmus 23.8 GREEDY(A, M, ω)

Eingabe: Menge A, Mengensystem M auf A, $\omega : A \to \mathbb{R}_0^+$.
Ausgabe: $T \in M$ mit maximalen Kosten

```
1: T := ∅
2: while A ≠ ∅ do
3:    wähle a ∈ A mit ω(a) = max{ω(a') | a' ∈ A}
4:    A := A \ {a};
5:    if T ∪ {a} ∈ M then
6:       T := T ∪ {a};
7:    end if
8: end while
9: return T
```

Matroide

Sei A eine nichtleere Menge und M ein aus Teilmengen von A bestehendes Mengensystem. Das Mengensystem M heißt ein *Filter* auf A, wenn M nach unten bzgl. Inklusion abgeschlossen ist, d. h., aus $S \in M$ und $T \subseteq S$ stets $T \in M$ folgt. Ein Filter M auf A heißt ein *Matroid*, wenn M die *Austauscheigenschaft* besitzt, d. h., für alle $S, T \in M$ gilt

$$|S| = |T| + 1 \quad \Longrightarrow \quad \exists s \in S \setminus T \; [T \cup \{s\} \in M]. \tag{23.10}$$

Die Elemente eines Matroids M heißen *unabhängige Mengen*. Die unabhängigen Mengen maximaler Mächtigkeit werden *Basen* von M genannt. Matroide wurden von Hassler Whitney (907-1989) eingeführt.

Beispiele 23.13.
- Sei A ein endlich-dimensionaler Vektorraum. Die Menge M aller linear unabhängigen Teilmengen von A bildet ein Matroid, das *lineare Matroid* auf A. Die Austauscheigenschaft entspricht dem aus der Linearen Algebra bekannten Austauschsatz von Steinitz.
- Sei A eine endliche Menge und k eine natürliche Zahl. Die Menge M aller Teilmengen T von A mit $|T| \leq k$ bildet ein Matroid, das *uniforme Matroid* auf A bzgl. k.
- Sei $G = (V, E)$ ein Graph. Die Menge M aller Teilmengen von E, die einen Wald aufspannen, bildet ein Matroid, das *graphische Matroid* auf G. Die Filtereigenschaft von M ist erfüllt, weil ein Teilgraph eines Waldes wiederum ein Wald ist. Wir beweisen die Austauscheigenschaft. Seien $S, T \in M$ mit $|S| = |T| + 1$. Dann gibt es eine Kante $uv \in S$, die nicht zu T gehört. Diese Kante kann nach Satz 21.16 so gewählt werden, dass einer ihrer Endpunkte in dem von S aufgespannten Wald liegt, aber nicht in dem von T aufgespannten Wald. Der von $T \cup \{uv\}$ aufgespannte Teilgraph von G ist dann ebenfalls ein Wald, also $T \cup \{uv\} \in M$.

Alle Basen eines endlich-dimensionalen Vektorraums sind gleichmächtig. Dies gilt allgemeiner für die Basen eines endlichen Matroids.

Lemma 23.14. *Ist M ein Matroid auf einer endlichen Menge A, dann sind alle Basen von M gleichmächtig.*

Beweis. Seien S und T Basen von M. Angenommen, es wäre $|S| \neq |T|$. O.B.d.A. sei $|S| > |T|$. Wir verringern die Basis S um $|S| - |T| - 1$ Elemente. Die resultierende Menge S' liegt ebenfalls in M, weil M ein Filter ist. Es gilt $|S'| = |T| + 1$. Aufgrund der Austauscheigenschaft gibt es ein Element $s \in S' \setminus T$ mit $T \cup \{s\} \in M$. Also ist T widersprüchlicherweise keine Basis von M. □

Satz 23.15. *Sei M ein Filter auf einer endlichen Menge A. Der Algorithmus* Greedy$(A, M, \cdot)$ *ist optimal genau dann, wenn M ein Matroid auf A ist.*

Beweis. Sei M ein Filter auf A. Angenommen, M würde die Austauscheigenschaft nicht erfüllen. Dann gibt es Elemente $S, T \in M$ mit $|S| = |T| + 1$, sodass für jedes Element $s \in S \setminus T$ gilt $T \cup \{s\} \notin M$. Wir setzen $k = |T|$ und definieren eine Kostenfunktion $\omega : A \to \mathbb{R}_0^+$ vermöge

$$\omega(a) = \begin{cases} k+2, \text{ falls } a \in T, \\ k+1, \text{ falls } a \in S \setminus T, \\ 0, \qquad \text{sonst.} \end{cases}$$

Der Algorithmus Greedy(A, M, ω) wählt definitionsgemäß zuerst alle Elemente der Menge T, weil sie allesamt maximale Kosten haben. Die Kosten von T sind

$$\omega(T) = |T| \cdot (k+2) = k(k+2).$$

Gemäß obiger Annahme kann der Algorithmus kein weiteres Element auswählen, um das Kosten zu erhöhen. Es gilt aber

$$\omega(S) \geq |S| \cdot (k+1) = (k+1)^2 > \omega(T).$$

Also ist Greedy(A, M, ω) nicht optimal.

Umgekehrt sei M ein Matroid auf A und $\omega : A \to \mathbb{R}_0^+$ eine Abbildung. Sei T das von Greedy(A, M, ω) gelieferte Element aus M. Angenommen, T wäre keine Basis von M. Dann gibt es wie im Beweis von Lemma 23.14 ein Element s, sodass $T \cup \{s\}$ zum Matroid M gehört. Ein solches Element s fügt Greedy(A, M, ω) aber widersprüchlicherweise der Menge T hinzu.

Wir zeigen, dass für jede weitere Basis S von M die Ungleichung $\omega(S) \leq \omega(T)$ besteht. Seien $T = \{t_1, \ldots, t_k\}$ mit $\omega(t_1) \geq \ldots \geq \omega(t_k)$ und $S = \{s_1, \ldots, s_k\}$ mit $\omega(s_1) \geq \ldots \geq \omega(s_k)$. Nach Definition von T gilt $\omega(t_1) \geq \omega(s_1)$. Seien $T_{l-1} = \{t_1, \ldots, t_{l-1}\}$ und $S_{l-1} = \{s_1, \ldots, s_{l-1}\}$ mit $\omega(T_{l-1}) \geq \omega(S_{l-1})$. Im Falle $\omega(t_l) \geq \omega(s_l)$ haben wir für $T_l = \{t_1, \ldots, t_l\}$ und $S_l = \{s_1, \ldots, s_l\}$ sofort $\omega(T_l) \geq \omega(S_l)$. Andernfalls wählt der Algorithmus ein Element $s_i \in S_l \setminus T_l$, $1 \leq i \leq l$, oder gar ein Element einer anderen Basis von M mit Kosten $\geq \omega(s_i)$, sodass $T_{l-1} \cup \{s_i\} \in M$. Wegen $\omega(s_i) \geq \omega(s_l) > \omega(t_l)$ hat der Algorithmus also vor t_l schon s_i ausgewählt, weshalb dieser Fall nicht auftreten kann. Also ist Greedy$(A, M, \cdot)$ optimal. □

Dass dieser Satz nicht für Filter gilt, zeigt das folgende

Beispiel 23.16. Sei $A = \{a, b, c\}$. Die Menge $M = \{\emptyset, \{a\}, \{b\}, \{c\}, \{b, c\}\}$ ist ein Filter, aber kein Matroid auf A. Mit den Kosten $\omega(a) = 3$ und $\omega(b) = \omega(c) = 2$ liefert $\textsc{Greedy}(A, M, \omega)$ die Menge $T = \{a\}$ mit $\omega(T) = 3$, obgleich $S = \{b, c\}$ mit $\omega(S) = 4$ optimal ist.

Wird obiger Satz auf das graphische Matroid angewendet, so ergibt sich ein wichtiges

Korollar 23.17. *Sei G ein endlicher Graph.* `MinSpannbaum`*$(G, \cdot)$ ist ein optimaler Greedy-Algorithmus, der zu jeder Kostenfunktion ω auf den Kanten von G einen minimalen Spannbaum von G liefert.*

23.5 Knotenfärbungen

Abschließend wird das Problem der Knotenfärbung von Graphen untersucht. Sei $G = (V, E)$ ein Graph und k eine natürliche Zahl. Eine *Färbung* von G mit k Farben ist eine Abbildung $f : V \to \underline{k}$, die je zwei adjazenten Knoten von G unterschiedliche Farben zuordnet. Existiert eine solche Abbildung, dann heißt der Graph *k-färbbar*. Die *chromatische Zahl* von G ist die minimale Anzahl von Farben, die zur Färbung von G notwendig sind, sie wird mit $\chi(G)$ bezeichnet.

Satz 23.18. *Ein Graph G ist 2-färbbar genau dann, wenn G bipartit ist.*

Beweis. Ein bipartiter Graph G mit der 2-Partition $\{V_1, V_2\}$ seiner Knotenmenge wird 2-gefärbt, indem alle Knoten in V_1 rot und alle Knoten in V_2 blau gefärbt werden. Umgekehrt enthält ein 2-färbbarer Graph keine Kreise von ungerader Länge und ist somit nach Satz 21.19 bipartit. □

Das Entscheidungsproblem, zu einem Graphen G und einer natürlichen Zahl K eine Färbung von G mit höchstens K Farben zu finden, ist NP-vollständig. Das zugehörige Optimierungsproblem, die chromatische Zahl eines Graphen zu ermitteln, ist NP-hart.

Ein bekanntes Greedy-Verfahren zur Knotenfärbung eines Graphen G stammt von Welch und Powell. Hierbei werden die Knoten von G in einer beliebigen Reihenfolge geordnet. Der erste Knoten wird mit der Farbe 1 gefärbt. Ist der i-te Knoten schon gefärbt, dann wird der $(i + 1)$-te Knoten mit der kleinsten Farbe gefärbt, die nicht für seine Nachbarn verwendet wird. Die resultierende Färbung hängt von der jeweiligen Knotenreihenfolge ab.

Algorithmus 23.9 WELCH-POWELL(G)

Eingabe: Graph G mit Knotenfolge $(v_1, \ldots, v_n)$
Ausgabe: Knotenfärbung f von G

```
 1: f(v_1) := 1;
 2: for i := 2 to n do
 3:    F := ∅
 4:    for j := 1 to i − 1 do
 5:       if v_j ist adjazent zu v_i then
 6:          F := F ∪ {f(v_j)}
 7:       end if
 8:    end for
 9:    l := 1
10:    while l ∈ F do
11:       l := l + 1
12:    end while
13:    f(v_i) := l
14: end for
```

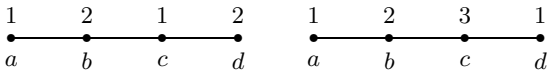


Abb. 23.4. Zwei Knotenfärbungen eines Graphen.

Beispiel 23.19. Wir betrachen den Graphen G in Abb. 23.4. Der Algorithmus WELCH-POWELL(G) liefert für die Knotenfolgen (a, b, c, d) und (a, d, b, c) folgende Färbungen

v	a	b	c	d
$f(v)$	1	2	1	1
F		$\{a\}$	$\{b\}$	$\{c\}$

v	a	d	b	c
$f(v)$	1	1	2	3
F		$\emptyset$	$\{a\}$	$\{b, d\}$

Die erste Färbung zeigt, dass der Graph 2-färbbar ist.

Satz 23.20. *Sei G ein Graph. Bezeichnet $\Delta(G)$ den maximalen Grad von G und $\omega(G)$ die Knotenanzahl einer maximalen Clique in G, dann gilt*

$$\omega(G) \leq \chi(G) \leq \Delta(G) + 1. \tag{23.11}$$

Ist der Graph G nichtregulär und zusammenhängend, so gilt

$$\chi(G) \leq \Delta(G). \tag{23.12}$$

Beweis. Um eine Clique mit k Knoten zu färben, sind mindestens k Farben notwendig. Also folgt $\omega(G) \leq \chi(G)$. WELCH-POWELL(G) liefert eine Knotenfärbung, die mit maximal $\Delta(G) + 1$ Farben auskommt, weil jeder Knoten höchstens $\Delta(G)$ Nachbarn besitzt. Damit ist die erste Aussage bewiesen.

Um die zweite Aussage zu beweisen, werden die Knoten von G in einer Liste angeordnet. Sei $k = \Delta(G)$. Der letzte Knoten v_n in der Liste wird so gewählt, dass sein Grad kleiner als k ist. Ein solcher Knoten existiert, weil G nicht regulär ist. Die zu v_n adjazenten Knoten werden in die Liste von hinten her eingefügt. Sei v_{i+1} der letzte Knoten in der Liste, dessen adjazente Knoten schon eingetragen wurden. Dann werden im nächsten Schritt alle zu v_i adjazenten Knoten in die Liste von hinten her eingefügt. Da G zusammenhängend ist, werden auf diese Weise alle Knoten in die Liste aufgenommen. In jedem Schritt ist die Anzahl der eingefügten Knoten höchstens $k-1$. Für diese Liste liefert WELCH-POWELL(G) also eine k-Färbung von G. □

Sei $p_G(k)$ die Anzahl der k-Färbungen eines Graphen G. Die chromatische Zahl $\chi(G)$ ist per definitionem die kleinste natürliche Zahl k mit $p_G(k) \geq 1$.

Jede k-Färbung des vollständigen Graphen K_n ist eine n-Permutation von k, mithin gilt

$$p_{K_n}(k) = k(k-1)\cdots(k-n+1). \tag{23.13}$$

Sei G ein Graph mit zwei nichtadjazenten Knoten u und v. Sei G_{uv} derjenige Graph, der aus G durch Hinzufügen der Kante uv entsteht, und sei $G_{u=v}$ derjenige Graph, der aus G durch Identifizieren der beiden Knoten hervorgeht (Abb. 23.5). Die k-Färbungen von G zerfallen in zwei disjunkte Teilmengen,

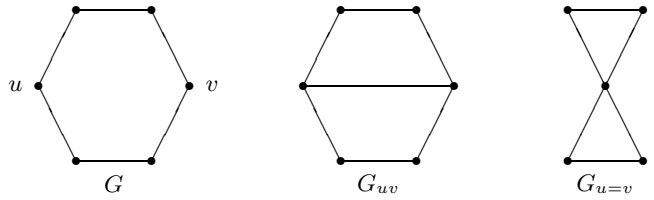


Abb. 23.5. Ein Graph G und die abgeleiteten Graphen G_{uv} und $G_{u=v}$.

je nachdem, ob die Knoten u und v gleich oder verschieden gefärbt sind. Die k-Färbungen von G, in denen die Knoten u und v verschieden gefärbt sind, entsprechen den k-Färbungen von G_{uv}, während die k-Färbungen von G, in denen die Knoten u und v gleich gefärbt sind, zu den k-Färbungen von $G_{u=v}$ korrespondieren. Damit haben wir bewiesen

Satz 23.21. *Sei G ein Graph. Sind u und v nichtadjazente Knoten in G, dann gilt*

$$p_G(k) = p_{G_{uv}}(k) + p_{G_{u=v}}(k) \tag{23.14}$$

und

$$\chi(G) = \min\{\chi(G_{uv}), \chi(G_{u=v})\}. \tag{23.15}$$

Die abgeleiteten Graphen G_{uv} und $G_{u=v}$ besitzen mindestens ein nichtadjazentes Paar von Knoten weniger als G. Wenn wir dieses Verfahren sukzessive auf G_{uv} und $G_{u=v}$ fortsetzen, so resultieren nach endlich vielen Schritten vollständige Graphen, für die die Anzahl der Färbungen gemäß (23.13) bekannt ist (Abb. 23.6).

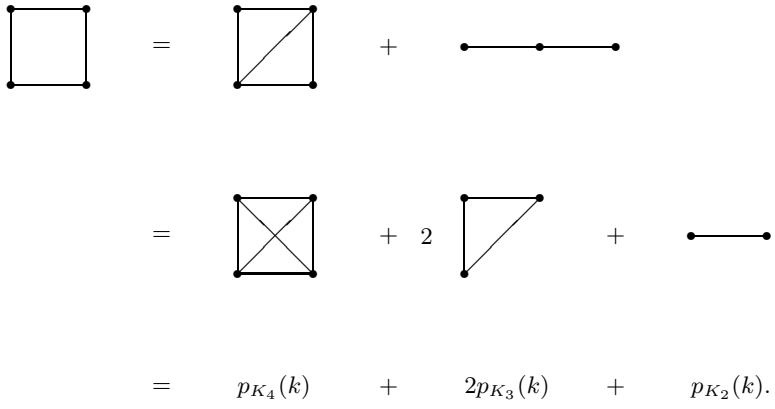


Abb. 23.6. Iterative Berechnung des chromatischen Polynoms eines Quadrats.

Der Ausdruck $p_G(k)$ ist eine ganzzahlige Linearkombination von Ausdrücken $p_{K_n}(k)$ für vollständige Graphen K_n. Der Ausdruck $p_{K_n}(k)$ ist aber eine Polynomfunktion in k mit ganzzahligen Koeffizienten. Damit haben bewiesen

Satz 23.22. *Sei G ein Graph. Der Ausdruck $p_k(G)$ ist eine Polynomfunktion in k mit ganzzahligen Koeffizienten.*

Die Polynomfunktion $p_G(k)$ heißt das *chromatisches Polynom* von G.

Selbsttestaufgaben

23.1. Zeige, dass das Cliquenproblem in NP liegt.

23.2. Entwickle einen Backtracking-Algorithmus für das Knotenüberdeckungsproblem und gib eine geeignete Bounding-Funktion an.

23.3. Sei $(M_1, \ldots, M_n)$ eine Familie von Mengen. Gesucht ist eine Teilfamilie $(M_{i_1}, \ldots, M_{i_m})$, so dass

$$\bigcup_{j=1}^{n} M_j = \bigcup_{j=1}^{m} M_{i_j}.$$

Löse dieses *Mengenüberdeckungsproblem* anhand eines Backtracking-Algorithmus'.

23.4. Berechne alle k-Färbungen eines endlichen Graphen anhand eines Backtracking-Algorithmus.

23.5. Entwickle einen Bergauf-Algorithmen für das obige Mengenüberdeckungsproblem.

23.6. In welcher Beziehung stehen die Entwurfsparameter von heuristischen Algorithmen?

23.7. Wir betrachten ein Rucksack-Problem mit 15 Gegenständen und Kapazität $K_0 = 1019$. Die Gegenstände haben folgenden Wert und Größe:

Werte	535	440	418	120	512	386	48	574	376	146	143	106	55	124	266
Größe	244	202	192	56	240	182	23	277	183	73	72	54	29	67	144

- Berechne einen optimalen Rucksack anhand eines Backtracking-Algorithmus.
- Berechne Näherungslösungen durch die Bergaufmethode, simuliertes Ausglühen und einen genetischen Algorithmus.

23.8. Berechne das chromatische Polynom eines Fünfecks.

23.9. Sei M ein Matroid auf einer endlichen Menge A. Der *Rang* einer Teilmenge T von A ist definiert durch

$$\mathrm{rg}(T) = \max\{|X| \mid X \subseteq T, X \in M\}.$$

Zeige, dass für alle $S, T \subseteq A$ und $a, b \in A$ gilt

- $0 \leq \mathrm{rg}(T) \leq |T|$.
- $T \in M \iff \mathrm{rg}(T) = |T|$.
- Monotonie: $T \subseteq S \implies \mathrm{rg}(T) \leq \mathrm{rg}(S)$.
- $\mathrm{rg}(T) \leq \mathrm{rg}(T \cup \{a\}) \leq \mathrm{rg}(T) + 1$.
- Submodularität: $\mathrm{rg}(T) + \mathrm{rg}(S) \geq \mathrm{rg}(T \cup S) + \mathrm{rg}(T \cap S)$.
- $\mathrm{rg}(T \cup \{a\}) = \mathrm{rg}(T \cup \{b\}) = \mathrm{rg}(T) \implies \mathrm{rg}(T \cup \{a, b\}) = \mathrm{rg}(T)$.

24

Lineare Optimierung

Viele Optimierungsaufgaben lassen sich durch lineare Programme darstellen. Dies gilt insbesondere für Produktions-, Transport-, Zuordnungs- und Netzwerkprobleme. Wir entwickeln die Grundlagen der linearen Optimierung und zeigen, dass die Sätze von Ford-Fulkerson und König-Egerváry allesamt Spezialfälle des Dualitätssatzes der linearen Optimierung sind.

24.1 Beispiele

Wir spezifizieren drei weiter vorne schon behandelte Optimierungsaufgaben als lineare Optimierungsprobleme.

Rucksackproblem

Das rationale Rucksackproblem (23.5) wird durch folgende Aufgabe beschrieben

$$\begin{array}{ll} & \max \sum_i f_i x_i. \\ \text{s.d.} & \sum_i g_i x_i \leq K_0 \\ & x_i \geq 0 \quad \forall i \in \{1, \ldots, n\} \\ & x_i \leq 1 \quad \forall i \in \{1, \ldots, n\} \end{array} \tag{24.1}$$

Paarungsproblem

Sei $G = (V, E)$ ein bipartiter Graph. Wir spezifizieren jede Teilmenge P von E durch einen Vektor $x = (x_e)$ mit

$$x_e = \begin{cases} 1 \text{ falls } e \in P, \\ 0 \text{ sonst.} \end{cases} \tag{24.2}$$

Das Problem, eine maximale Paarung in G zu finden, ist durch folgende Aufgabe gegeben

$$\begin{aligned} &\max \textstyle\sum_{e\in E} x_e. \\ \text{s.d. } &\textstyle\sum_{v\in e} x_e \leq 1 \quad \forall v \in V \\ &x_e \in \{0,1\} \quad \forall e \in E \end{aligned} \tag{24.3}$$

Netzwerkproblem

Sei $N = (D, \kappa, q, s)$ ein Flussnetz. Sei f_{vw} der Wert eines Flusses f auf der Kante vw. Das Problem, einen maximalen Fluss auf N zu finden, ist durch folgende Aufgabe spezifiziert

$$\begin{aligned} &\max \textstyle\sum_{v\in q^+} f_{qv} - \sum_{v\in q^-} f_{vq} \\ \text{s.d. } &f_{vw} \leq \kappa(vw) \quad \forall vw \in E \\ &\textstyle\sum_{u\in v^-} f_{uv} = \sum_{w\in v^+} f_{vw} \quad \forall v \in V \setminus \{q, s\} \\ &f_{vw} \geq 0 \quad \forall vw \in E \end{aligned} \tag{24.4}$$

24.2 Lineare Programme und Dualität

In diesem Abschnitt werden die Grundlagen der linearen Programmierung erörtert. Alle Vektoren sind Spaltenvektoren.

Lineare Programme

Seien $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$. Unter einem *linearen Programm* (LP) wird die Aufgabe verstanden, einen Vektor $x \in \mathbb{R}^n$ zu finden, sodass

$$\begin{aligned} &\max c^T x. \\ \text{s.d. } &Ax \leq b \\ &x \geq 0 \end{aligned} \tag{24.5}$$

Ein Vektor $x \in \mathbb{R}^n$ heißt eine *zulässige Lösung* von (24.5), wenn er die Nebenbedingungen erfüllt, d. h., $Ax \leq b$ und $x \geq 0$. Die Menge aller zulässigen Lösungen von (24.5) heißt *zulässiger Bereich*. Eine zulässige Lösung x^* heißt *optimal*, wenn für jede zulässige Lösung x gilt $c^T x^* \geq c^T x$. Ein LP heißt *lösbar*, wenn es eine optimale Lösung besitzt. Ein LP kann in kanonische oder Standardform transformiert werden, wie in der ersten Selbsttestaufgabe angegeben.

Ein LP in der Form (24.5) wird *primales LP* genannt. Das zugehörige *duale LP* lautet

$$\begin{aligned} &\min b^T y. \\ \text{s.d. } &A^T y \geq c \\ &y \geq 0 \end{aligned} \tag{24.6}$$

Lemma 24.1. *Ist x eine zulässige Lösung des primalen LP (24.5) und y eine zulässige Lösung des dualen LP (24.6), dann gilt*

$$c^T x \leq b^T y. \tag{24.7}$$

Beweis. Da x zulässig ist und $y \geq 0$, gilt

$$b^T y = \sum_{i=1}^{m} b_i y_i \geq \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i. \tag{24.8}$$

Da y zulässig ist und $x \geq 0$, erhellt sich

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i \right) x_j \geq \sum_{j=1}^{n} c_j x_j = c^T x. \tag{24.9}$$

Damit ist die Aussage gezeigt. □

Beispiel 24.2. Wir betrachten das LP

$$\begin{aligned} & \max x_1 + 3x_2. \\ \text{s.d. } & -x_1 + 3x_2 \leq 6 \\ & 3x_1 + x_2 \leq 12 \\ & x_i \geq 0 \quad \forall i \end{aligned} \tag{24.10}$$

Der zulässige Bereich dieses LP ist in Abb. 24.1 illustriert. Die Zielfunktion $f(x) = x_1 + 3x_2$ beschreibt eine Parallelenschar, die senkrecht zum Vektor $c = (1, 3)^T$ steht. Die optimale Lösung des LP ist $x^* = (3, 3)^T$ mit dem Zielfunktionswert $f(x^*) = 12$.

Das zugehörige duale LP lautet

$$\begin{aligned} & \min 6y_1 + 12y_2. \\ \text{s.d. } & -y_1 + 3y_2 \leq 1 \\ & 3y_1 + y_2 \leq 3 \\ & y_i \geq 0 \quad \forall i \end{aligned} \tag{24.11}$$

Die zulässigen Lösungen des primalen LP und dualen LP verhalten sich gemäß Lemma 24.1 wie in Abb. 24.2 illustriert.

Korollar 24.3. *Sei x eine zulässige Lösung des primalen LP (24.5) und y eine zulässige Lösung des dualen LP (24.6). Aus $c^T x = b^T y$ folgt, dass x eine optimale Lösung von (24.5) und y eine optimale Lösung von (24.6) ist.*

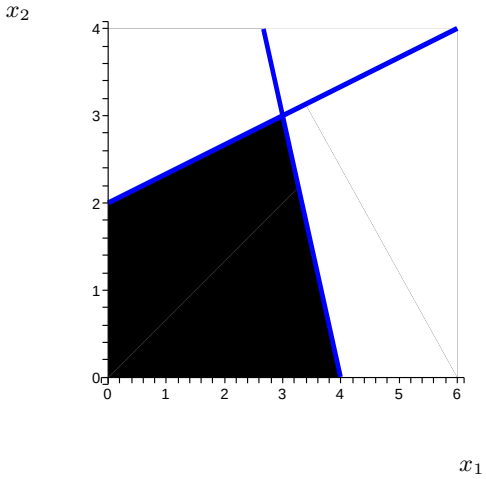


Abb. 24.1. Zulässiger Bereich (dunkel) des LP (24.10).

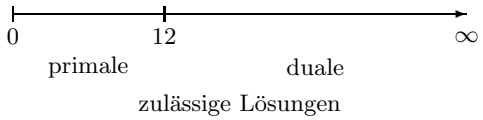


Abb. 24.2. Die Zielfunktionswerte des primalen LP (24.10) und des dualen LP (24.11).

Satz 24.4. (Farkas, 1902) *Die Menge* $P = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$ *ist nicht leer genau dann, wenn für jedes* $y \in \mathbb{R}^m$ *mit* $A^T y \geq 0$ *auch* $b^T y \geq 0$ *gilt.*

Beweis. Sei $\mathbf{x} \in P$. Dann gilt für jedes $y \in \mathbb{R}^m$ mit $A^T y \geq 0$ auch $b^T y = (Ax)^T y = x^T (A^T y) \geq 0$.

Umgekehrt sei P leer. Sei $a^{(i)}$ die i-te Spalte von A. Angenommen, $Ax = b$ wäre nicht lösbar. Dann ist b nicht darstellbar als Linearkombination der Spalten von A. Also hat die Matrix $(A \mid b)$ den Rang $r(A \mid b) = r(A) + 1$ und die Matrix $A' = \left(\begin{array}{c|c} A & b \\ \hline 0 & 1 \end{array}\right)$ den Rang $r(A') = r(A) + 1 = r(A \mid b)$. Somit ist die letzte Zeile $(0 \mid 1)$ linear abhängig von den ersten m Zeilen von A'. Also existieren Zahlen $y_1, \ldots, y_m$ mit $\sum_i {a^{(i)}}^T y_i = 0$ und $\sum_i b_i y_i = -1$, d. h., $y = (y_1, \ldots, y_m)^T$ erfüllt $A^T y = 0$ und $b^T y = -1$.

Angenommen, $\mathbf{Ax} = \mathbf{b}$ wäre lösbar, aber unter den Lösungen befänden sich keine nichtnegativen. Im Falle $n = 1$ sei $x_1 a^{(1)} = b$ und $x_1 < 0$. Mit $y = -b$ folgt ${a^{(1)}}^T y = -\frac{1}{x_1} b^T b \geq 0$ und $b^T y = -b^T b < 0$. Sei die Aussage für

alle r-dimensionalen LP mit $r \leq n-1$ richtig. Nach Induktionsannahme gibt es ein $v \in \mathbb{R}^m$ mit ${a^{(i)}}^T v \geq 0$, $1 \leq i \leq n-1$, und $b^T v < 0$. Im Falle ${a^{(n)}}^T v \geq 0$ sind wir fertig. Andernfalls setzen wir

$$\begin{aligned} \hat{a}^{(i)} &= ({a^{(i)}}^T v)a^{(n)} - ({a^{(n)}}^T v)a^{(i)}, \quad 1 \leq i \leq n-1, \\ \hat{b} &= (b^T v)a^{(n)} - ({a^{(n)}}^T v)b. \end{aligned}$$

Angenommen, es gäbe $x_i \geq 0$, $1 \leq i \leq n-1$, so dass $\sum_{i=1}^{n-1} x_i \hat{a}^{(i)} = \hat{b}$. Dann folgt

$$-\frac{1}{{a^{(n)}}^T v}\left(\sum_{i=1}^{n-1} x_i({a^{(i)}}^T v) - b^T v\right) a^{(n)} + \sum_{i=1}^{n-1} x_i a^{(i)} = b. \tag{24.12}$$

Die Koeffizienten $a^{(i)}$, $1 \leq i \leq n$, sind nach Annahme nichtnegativ. Dies widerspricht der Annahme, dass P leer ist. Also ist $\sum_{i=1}^{n-1} x_i \hat{a}^{(i)} = \hat{b}$, $x_i \geq 0$, $1 \leq i \leq n-1$, nicht lösbar. Nach Induktionsannahme gibt es ein $w \in \mathbb{R}^m$ mit ${\hat{a}^{(i)}}^T w \geq 0$, $1 \leq i \leq n-1$, und $\hat{b}^T w < 0$. Für den Vektor $y = ({a^{(n)}}^T w)v - ({a^{(n)}}^T v)w$ gilt dann ${a^{(i)}}^T y = {\hat{a}^{(i)}}^T w \geq 0$, $1 \leq i \leq n-1$, ${a^{(n)}}^T y = 0$ und $b^T y = \hat{b}^T w < 0$. □

Satz 24.5. *Die Menge $P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ ist nicht leer genau dann, wenn für jedes $y \in \mathbb{R}^m$ mit $A^T y \geq 0$ und $y \geq 0$ auch $b^T y \geq 0$ gilt.*

Beweis. Ist P nicht leer, dann wird wie im obigen Satz verfahren. Sei P leer. Dann ist auch $P' = \{x \in \mathbb{R}^n \mid Ax + z = b, x \geq 0, z \geq 0\}$ leer. Sei $B = (A \mid I_m)$, wobei I_m die m-reihige Einheitsmatrix ist. Dann ist P' leer genau dann, wenn $P'' = \{x \in \mathbb{R}^{n+m} \mid Bv = b, v \geq 0\}$ leer ist. Nach dem obigen Satz gibt es ein $y \in \mathbb{R}^m$ mit $A^T y \geq 0$, $y = I_m y \geq 0$ und $b^T y < 0$. □

Satz 24.6. (Dualitätssatz) *Seien ein primales und das zugehörige duale Programm gegeben*

$$\begin{array}{lll} & \max c^T x & \qquad \min b^T y. \\ s.d. & Ax \leq b & \qquad s.d.\ A^T y \geq c \\ & x \geq 0 & \qquad y \geq 0 \end{array} \tag{24.13}$$

- *Wenn beide Programme jeweils mindestens eine zulässige Lösung haben, dann sind beide Programme lösbar. Für die optimalen Lösungen x^* und y^* gilt $c^T x^* = b^T y^*$.*
- *Wenn eines der beiden Programme keine zulässige Lösung besitzt, dann hat keines der beiden Programme eine optimale Lösung.*

Beweis. Wir nehmen an, dass beide LP zulässige Lösungen besitzen. Sei $P = \{(x, y) \mid Ax \leq b, x \geq 0, A^T y \geq c, y \geq 0, c^T x - b^T y \geq 0\}$. Wir setzen

$$A' = \begin{pmatrix} A & 0 \\ 0 & -A^T \\ -c^T & b^T \end{pmatrix} \quad \text{und} \quad d = \begin{pmatrix} b \\ -c \\ 0 \end{pmatrix}.$$

Dann folgt $P = \{z \mid A'z \leq d, z \geq 0\}$. Angenommen, P wäre leer. Dann gibt es nach Satz 24.5 ein $y \in \mathbb{R}^{m+n+1}$ mit $A'^T y \geq 0$, $y \geq 0$ und $d^T y < 0$. D. h., es gibt $u \in \mathbb{R}^m$, $v \in \mathbb{R}^n$ und $\kappa \in \mathbb{R}$ mit $u \geq 0$, $v \geq 0$ und $\kappa \geq 0$, so dass

$$A^T u \geq \kappa c, \quad Av \leq \kappa b, \quad b^T u < c^T v. \tag{24.14}$$

Angenommen, es wäre $\kappa = 0$. Sei x eine zulässige Lösung des primalen LP und y eine zulässige Lösung des dualen LP. Dann folgt widersprüchlicherweise

$$0 \leq x^T (A^T u) = (Ax)^T u \leq b^T u < c^T v \leq (A^T y)^T v = y^T (Av) \leq 0.$$

Also muss $\kappa > 0$ sein. Setzen wir $x = \frac{1}{\kappa} v$ und $y = \frac{1}{\kappa} u$, dann erhellt sich $Ax \leq b$, $A^T y \geq c$, $x \geq 0$ und $y \geq 0$. Mit Lemma 24.1 ergibt sich $c^T v = \kappa(c^T x) \leq \kappa(b^T y) = b^T u$, was (24.14) widerspricht. Also ist P nicht leer und mit Korollar 24.3 folgt die erste Aussage.

Sei das primale LP nicht lösbar. Dann gibt es nach Satz 24.5 ein $y \in \mathbb{R}^m$ mit $A^T y \geq 0$, $y \geq 0$ und $b^T y < 0$. Sei y' eine zulässige Lösung des dualen LP. Dann ist auch $y' + \kappa y$ eine zulässige Lösung des dualen LP für jedes $\kappa \geq 0$. Es gilt $b^T(y' + \kappa y) = b^T y' + \kappa(b^T y)$. Dieser Ausdruck kann wegen $b^T y < 0$ beliebig klein werden. Also ist das duale LP nicht lösbar. Die Umkehrung wird analog bewiesen. □

Beispiel 24.7. Sei $G = (V, E)$ ein Graph mit der Inzidenzmatrix $A = (a_{v,e})$. Das Problem, eine maximale Paarung in G finden, lautet nach (24.3)

$$\begin{aligned} &\max 1^T x. \\ \text{s.d. } &Ax \leq 1 \\ &x_e \in \{0, 1\} \quad \forall e \in E \end{aligned} \tag{24.15}$$

Wir relaxieren dieses Problem, indem wir reellwertige Lösungen zulassen, und erhalten

$$\begin{aligned} &\max 1^T x. \\ \text{s.d. } &Ax \leq 1 \\ &x \geq 0 \end{aligned} \tag{24.16}$$

Das duale Problem lautet

$$\begin{aligned} &\min 1^T y. \\ \text{s.d. } &A^T y \geq 1 \\ &y \geq 0 \end{aligned} \tag{24.17}$$

Wir restringieren die Variablen und erhalten das ganzzahlige LP

$$\begin{aligned} &\min 1^T y. \\ \text{s.d. } &A^T y \geq 1 \\ &y_v \in \{0,1\} \quad \forall v \in V \end{aligned} \tag{24.18}$$

Jede Kante $e = uv$ in G liefert eine Nebenbedingung $y_u + y_v \geq 1$ in $A^T y \geq 1$. Eine optimale Lösung $y = (y_v)$ von (24.18) beschreibt eine minimale Knotenüberdeckung von G.

24.3 Der zulässige Bereich und die Rolle der Ecken

Eine Teilmenge A von $\mathbb{R}^n$ heißt *konvex*, wenn mit $x, y \in A$ auch $\lambda x + (1-\lambda)y \in A$ für alle $\lambda \in [0,1]$. Eine konvexe Menge enthält also mit je zwei Punkten auch die Strecke zwischen diesen Punkten.

Lemma 24.8. *Sei $A \subseteq \mathbb{R}^n$ eine konvexe Menge und r eine natürliche Zahl. Sind $x^{(1)}, \ldots, x^{(r)} \in A$, dann ist $\sum_i \lambda_i x^{(i)} \in A$ für alle $\lambda_1, \ldots, \lambda_r \geq 0$ mit $\sum_{i=1}^r \lambda_i = 1$.*

Der Beweis wird durch vollständige Induktion nach r geführt.

Die *konvexe Hülle* einer Menge $A \subseteq \mathbb{R}^n$ ist definiert durch

$$\operatorname{conv}(A) = \{\sum_{i=1}^{r} \lambda_i x^{(i)} \mid \lambda_i \geq 0, x^{(i)} \in A, \sum_{i=1}^{r} \lambda_i = 1, r \geq 1\}. \tag{24.19}$$

Die konvexe Hülle von A ist die kleinste konvexe Menge, die A enthält. Ist A endlich, dann wird $\operatorname{conv}(A)$ ein *konvexes Polyeder* genannt.

Sei P ein konvexes Polyeder. Ein Punkt $x \in P$ heißt eine *Ecke* von P, wenn für jede Darstellung

$$x = \sum_{i=1}^{r} \lambda_i x^{(i)}, \quad \text{mit } x^{(i)} \in P,\ \lambda_i \geq 0, \text{ und } \textstyle\sum_{i=1}^r \lambda_i = 1, \tag{24.20}$$

gilt $x = x^{(i)}$ für alle i mit $\lambda_i > 0$. Eine Ecke x lässt sich also nicht als eine *konvexe Linearkombination* (d. h., $\lambda_i \geq 0$ und $\sum_{i=1}^r \lambda_i = 1$) von anderen Punkten in P schreiben. Insbesondere liegt eine Ecke von P nicht zwischen zwei anderen Punkten in P.

Beispiel 24.9. Die Ecken des LP (24.10) sind $(0,0)^T$, $(4,0)^T$, $(3,3)^T$ und $(0,2)^T$.

Satz 24.10. *Ein konvexes Polyeder ist die konvexe Hülle seiner Ecken.*

Beweis. Sei $A = \{x^{(1)}, \ldots, x^{(r)}\}$ und $P = \text{conv}(A)$. Aus A können alle Punkte entfernt werden, die sich als konvexe Linearkombination der übrigen Punkte darstellen lassen. Denn sei etwa

$$x^{(1)} = \sum_{i=2}^{r} \lambda_i x^{(i)} \quad \text{mit } \lambda_i \geq 0, \text{ und } \textstyle\sum_{i=2}^{r} \lambda_i = 1.$$

Für jedes $y \in P$ gilt

$$y = \sum_{i=1}^{r} \mu_i x^{(i)} \quad \text{mit } \mu_i \geq 0, \text{ und } \textstyle\sum_{i=1}^{r} \mu_i = 1.$$

Es folgt $y = \sum_{i=2}^{r} (\mu_1 \lambda_i + \mu_i) x^{(i)}$ und $\sum_{i=2}^{r} (\mu_1 \lambda_i + \mu_i) = 1$. Somit ist $P = \text{conv}(\{x^{(2)}, \ldots, x^{(r)}\})$. Es kann angenommen werden, dass kein Punkt in A eine konvexe Linearkombination der übrigen Punkte in A ist. Denn sei

$$x^{(1)} = \sum_{j=1}^{t} \mu_j y^{(j)} \quad \text{mit } y^{(j)} \in P, \, \mu_j \geq 0, \text{ und } \textstyle\sum_{j=1}^{t} \mu_j = 1,$$

und

$$y^{(j)} = \sum_{i=1}^{r} \lambda_{ji} x^{(i)} \quad \text{mit } \lambda_{ji} \geq 0, \text{ und } \textstyle\sum_{i=1}^{r} \lambda_{ji} = 1.$$

Dann ist $x^{(1)} = \sum_{i=1}^{r} \sum_{j=1}^{t} \mu_j \lambda_{ji} x^{(i)}$, also

$$(1 - \sum_{j=1}^{t} \mu_j \lambda_{j1}) x^{(1)} = \sum_{i=2}^{r} \sum_{j=1}^{t} \mu_j \lambda_{ji} x^{(i)}.$$

Da $x^{(1)}$ keine konvexe Linearkombination von $x^{(2)}, \ldots, x^{(r)}$ ist, erhellt sich $\sum_{j=1}^{t} \mu_j \lambda_{j1} = 1$ und somit $\lambda_{j1} = 1$ für jedes j mit $\mu_j > 0$. Also ist $x^{(1)} = y^{(j)}$ für jedes j mit $\mu_j > 0$. Folglich ist $x^{(1)}$ eine Ecke von P. □

Wir untersuchen im Folgenden das LP

$$\begin{aligned} &\max c^T x. \\ \text{s.d. } &Ax = b \\ &x \geq 0 \end{aligned} \tag{24.21}$$

Wir können $m \leq n$ annehmen und dass A den Rang $r(A) = m$ besitzt. Im Falle $m > n$ wären $m - r(A)$ Gleichungen überflüssig. Sei $P = \{x \mid Ax = b, x \geq 0\}$ der zulässige Bereich von (24.21) und bezeichne $a^{(i)}$ die i-te Spalte von A. Sei $x \in P$ und $I = \{i \mid x_i > 0\}$ die Menge aller Indizes i mit $x_i > 0$. Die Menge

$\{a^{(i)} \mid i \in I\}$ wird die zu x *gehörende Basis* von A genannt. Für die zu x gehörende Basis von A gilt

$$\sum_{i \in I} x_i a^{(i)} = b. \tag{24.22}$$

Satz 24.11. *Ein Punkt $x \in P$ ist eine Ecke von P genau dann, wenn die zu x gehörende Basis von A linear unabhängig ist.*

Beweis. Sei x ist eine Ecke von P und $I = \{i \mid x_i > 0\}$. Angenommen, $\{a^{(i)} \mid i \in I\}$ wäre linear abhängig. Dann gibt es Zahlen d_i, $i \in I$, die nicht alle gleich Null sind, so dass $\sum_{i \in I} d_i a^{(i)} = 0$. Setzen wir $d_j = 0$ für alle $j \in \{1, \ldots, n\} \setminus I$, dann ist $Ad = 0$ für $d = (d_1, \ldots, d_n)^T$. Wir zeigen, dass $y = x - \theta d$ und $z = x + \theta d$ in P liegen, wobei

$$\theta = \min \left\{ \frac{x_i}{|d_i|} \mid i \in I, d_i \neq 0 \right\} > 0.$$

Wegen $Ad = 0$ ist $Ay = b = Az$. Sei $1 \leq i \leq n$. Ist $i \notin I$, dann ist $d_i = 0$ und somit $y_i = 0 = z_i$. Ist $i \in I$ und $d_i = 0$, so ist $y_i = x_i = z_i$. Ist $i \in I$ und $d_i \neq 0$, dann ist $\theta |d_i| \leq \frac{x_i}{|d_i|} \cdot |d_i| = x_i$ und deshalb $y_i = x_i - \theta d_i \geq 0$ und $z_i = x_i + \theta d_i \geq 0$. Also liegen y und z in P. Die Punkte y und z sind wegen $\theta > 0$ von x verschieden und es gilt $x = \frac{1}{2} y + \frac{1}{2} z$. Also ist x widersprüchlicherweise keine Ecke von P.

Umgekehrt sei $\{a^{(i)} \mid i \in I\}$ linear unabhängig. Angenommen, es wäre $x = \lambda y + (1 - \lambda) z$, wobei $0 < \lambda < 1$ und $y, z \in P$. Nach Voraussetzung ist x die eindeutige Lösung des linearen Gleichungssystems $Ax = b$ mit $x_i = 0$ für alle $i \notin I$. Da y der Gleichung $Ay = b$ genügt, gibt es einen Index $i \notin I$ mit $y_i > 0$. Wegen $z \geq 0$ ist dann die i-te Komponente von $\lambda y + (1 - \lambda) z$ widersprüchlicherweise positiv. □

Jede Ecke von P hat also höchstens m positive Koordinaten. Die Anzahl der Ecken von P ist somit endlich. Eine Ecke von P heißt *nichtentartet*, wenn sie m positive Koordinaten besitzt, andernfalls heißt sie *entartet*.

Satz 24.12. *Ist das LP (24.21) lösbar und sein zulässiger Bereich ein konvexes Polyeder, dann gibt es unter den optimalen Lösungen eine Ecke.*

Beweis. Sei $A = \{x^{(1)}, \ldots, x^{(r)}\}$ die Menge aller Ecken von P und $P = \mathrm{conv}(A)$. Nach Satz 24.10 hat jedes $x \in P$ die Form

$$x = \sum_{i=1}^{r} \lambda_i x^{(i)} \quad \text{mit } \lambda_i \geq 0, \textstyle\sum_{i=1}^{r} \lambda_i = 1.$$

Sei $x^{(k)}$ eine Ecke mit $c^T x^{(k)} = \max\{c^T x^{(i)} \mid 1 \leq i \leq r\}$. Dann folgt

$$c^T x = \sum_{i=1}^{r} \lambda_i c^T x_i^{(i)} \leq \sum_{i=1}^{r} \lambda_i c^T x^{(k)} = c^T x^{(k)}.$$

Also ist $x^{(k)}$ eine optimale Lösung des LP. □

Im obigen Satz wurde angenommen, dass der zulässige Bereich von (24.21) ein konvexes Polyeder bildet. Im Allgemeinen ist der zulässige Bereich eines LP gegeben durch eine Summe

$$B = P + K = \{x + y \mid x \in A, y \in K\}, \tag{24.23}$$

wobei P ein konvexes Polyeder und $K = \{x \mid Ax = 0, x \geq 0\}$ ein endlich erzeugter konvexer Kegel ist. Es kann gezeigt werden, dass der Satz 24.12 in dieser Verallgemeinerung gültig bleibt.

Beispiel 24.13. Wir schreiben das LP (24.10) in Standardform

$$\begin{aligned} & \max x_1 + 3x_2 \\ \text{s.d. } & -x_1 + 3x_2 + x_3 = 6 \\ & 3x_1 + x_2 + x_4 = 12 \\ & x_i \geq 0 \quad \forall i \end{aligned}$$

indem wir jede Ungleichung durch eine *Schlupfvariable* in eine Gleichung transformieren ($x_3 \geq 0$ und $x_4 \geq 0$). Die Ecken dieses LP lassen sich nach Satz 24.11 durch $\binom{4}{2} = 6$ Gleichungssysteme ermitteln

$-x_1 + 3x_2 = 6$ $3x_1 + x_2 = 12$ $\mathbf{x}^{(1)} = (3,3,0,0)^T$	$-x_1 + x_3 = 6$ $3x_1 = 12$ $\mathbf{x}^{(2)} = (4,0,10,0)^T$
$-x_1 = 6$ $3x_1 + x_4 = 12$ $\mathbf{x}^{(3)} = (-6,0,0,30)^T$	$3x_2 + x_3 = 6$ $x_2 = 12$ $\mathbf{x}^{(4)} = (0,12,-30,0)^T$
$3x_2 = 6$ $x_2 + x_4 = 12$ $\mathbf{x}^{(5)} = (0,2,0,10)^T$	$x_3 = 6$ $x_4 = 12$ $\mathbf{x}^{(6)} = (0,0,6,12)^T$

Die optimale Lösung ist $\mathbf{x}^{(1)} = (3,3,0,0)^T$ mit dem Zielfunktionswert 12. Also ist $(3,3)^T$ die optimale Lösung von (24.10).

24.4 Ganzzahlige Programmierung

Ein *ganzzahliges lineares Programm* (ILP) hat die Form

$$\begin{aligned} & \max c^T x. \\ \text{s.d. } & Ax = b \\ & x \geq 0 \\ & x_i \in \mathbb{Z} \quad \forall i \end{aligned} \tag{24.24}$$

Wir relaxieren das Problem, indem wir reellwertige Lösungen zulassen, und erhalten

$$\begin{aligned} &\max c^T x. \\ \text{s.d. } &Ax = b \\ &x \geq 0 \end{aligned} \tag{24.25}$$

Der zulässige Bereich $P = \{x \mid Ax = b, x \geq 0\}$ dieses LP heißt *ganzzahlig*, wenn jede Ecke von P ganzzahlig ist. Wir entwickeln eine hinreichende Bedingung für die Ganzzahligkeit von P.

Eine quadratische Matrix A heißt *modular*, wenn A die Determinante ± 1 hat. Eine ganzzahlige Matrix A heißt *total unimodular*, wenn jede nichtsinguläre quadratische Untermatrix von A modular ist. Eine total unimodulare Matrix besteht also nur aus Einträgen 0, 1 und -1.

Lemma 24.14. *Ist A eine total unimodulare $m \times n$-Matrix, dann sind folgende Matrizen ebenfalls total unimodular: $-A$, A^T, $(A \mid I_m)$ und $(A \mid -A)$. Ebenso ist jede Matrix total unimodular, die durch Multiplikation einer Zeile oder Spalte von A mit -1 entsteht.*

Satz 24.15. *Ist A eine total unimodulare $m \times n$-Matrix und $b \in \mathbb{Z}^m$, dann ist $P = \{x \mid Ax = b, x \geq 0\}$ ganzzahlig.*

Beweis. Sei x eine Ecke von P und $\{a^{(i)} \mid i \in I\}$ die zu x gehörende Basis von A. Nach (24.22) ist $\sum_{i \in I} x_i a^{(i)} = b$. Für die Komponenten dieses Gleichungssystems gilt nach der Cramerschen Regel

$$x_i = \frac{\det(B^{(i)})}{\det(B)}, \quad i \in I,$$

wobei B die aus den Spalten $a^{(i)}$, $i \in I$, gebildete Matrix ist und $B^{(i)}$ aus B entsteht, indem die i-te Spalte durch b ersetzt wird. Da A total unimodular ist, folgt mit Satz 24.11 sofort $\det(B) = \pm 1$. Nach Voraussetzung sind alle Einträge von $B^{(i)}$ ganzzahlig, sodass $\det(B^{(i)})$ ganzzahlig ist. Folglich ist x ganzzahlig. □

Lemma 24.16. *Die Inzidenzmatrix eines Graphen G ist total unimodular genau dann, wenn G bipartit ist.*

Beweis. Sei G nicht bipartit und A die Inzidenzmatrix von G. Nach Satz 21.19 enthält G einen einfachen Kreis ungerader Länge. Die zu diesem Kreis gehörende quadratische Untermatrix von A hat die Determinante ± 2.

Sei $G = (V, E)$ bipartit mit der Knotenzerlegung $V = V_1 \cup V_2$ und A die Inzidenzmatrix von G. Für jede einreihige Untermatrix von A ist die Aussage richtig, weil A nur Einträge 0 und 1 enthält. Sei die Aussage für t-reihige Untermatrizen von A schon bewiesen. Sei B eine $(t+1)$-reihige Untermatrix von A. Wir unterscheiden drei Fälle:

- Die Matrix B enthält eine Nullspalte. Dann ist $\det(B) = 0$.
- Die Matrix B enthält eine Spalte j mit genau einer Eins, also etwa $a_{ij} = 1$. Dann ist $\det(B) = 1 \cdot \det(B')$, wobei B' diejenige Matrix ist, die aus B durch Streichen der i-ten Zeile und j-ten Spalte entsteht. Mit der Induktionsannahme über B' folgt die Behauptung.
- Ansonsten enthält B pro Spalte zwei Einsen, weil A eine Inzidenzmatrix ist. Da G bipartit ist, können die Zeilen von B in zwei Mengen $\{\, b^{(v)} \mid v \in V_1' \}$, $V_1' \subseteq V_1$, und $\{\, b^{(v)} \mid v \in V_2' \}$, $V_2' \subseteq V_2$, zerlegt werden. Dabei gilt

$$\sum_{v \in V_1'} b^{(v)} = \sum_{v \in V_2'} b^{(v)}.$$

  Also ist $\det(B) = 0$. □

Beispiel 24.17. Sei $G = (V, E)$ ein bipartiter Graph mit der Inzidenzmatrix $A = (a_{v,e})$. Nach den Lemmata 24.14 und 24.16 sind A und A^T total unimodular. Also hat die Relaxation (24.16) des Problems, eine maximale Paarung in G zu finden, nach Satz 24.15 eine optimale ganzzahlige Lösung x^*. Ebenso hat die Relaxation (24.17) des Problems, eine minimale Knotenüberdeckung von G zu finden, eine optimale ganzzahlige Lösung y^*. Aufgrund der Nebenbedingungen ist $x^* \in \{0,1\}^{|E|}$ und $y^* \in \{0,1\}^{|V|}$. Also beschreibt x^* eine maximale Paarung in G und y^* eine minimale Knotenüberdeckung von G. Nach dem Dualitätssatz haben die beiden relaxierten LP (24.16) und (24.17) den gleichen Zielfunktionswert, d. h., $1^T x^* = 1^T y^*$. Damit haben wir den Satz von König-Egerváry mithilfe der linearen Optimierung bewiesen.

Sei $D = (V, E)$ ein Digraph. Die *Inzidenzmatrix* von $D = (V, E)$ ist eine Matrix $A = (a_{v,e})$, definiert durch

$$a_{v,e} = \begin{cases} 1 \text{ falls } v = e^-, \\ -1 \text{ falls } v = e^+, \\ 0 \text{ sonst.} \end{cases}$$

Jede Spalte von A enthält genau einen Eintrag 1 und -1, die restlichen Einträge sind 0.

Lemma 24.18. *Die Inzidenzmatrix eines Digraphen ist total unimodular.*

Beweis. Sei $D = (V, E)$ ein Digraph mit der Inzidenzmatrix $A = (a_{v,e})$. Für jede einreihige Untermatrix von A ist die Aussage richtig, da A nur Einträge 0, 1 und -1 enthält. Sei die Aussage für t-reihige Untermatrizen von A schon bewiesen. Sei B eine $(t+1)$-reihige Untermatrix von A. Wir unterscheiden drei Fälle:

- Die Matrix B enthält eine Nullspalte. Dann ist $\det(B) = 0$.
- Die Matrix B enthält eine Spalte j mit genau einer Eins, also etwa $a_{ij} = 1$. Dann ist $\det(B) = 1 \cdot \det(B')$, wobei B' diejenige Matrix ist, die aus B durch Streichen der i-ten Zeile und j-ten Spalte entsteht. Mit der Induktionsannahme über B' folgt die Behauptung.
- Ansosten enthält B pro Spalte eine 1 und eine -1. Dann ist die Summe der Zeilen von B gleich 0 und somit $\det(B) = 0$. □

Beispiel 24.19. Sei $N = (D, \kappa, q, s)$ ein Flussnetz und $A = (a_{v,e})$ die Inzidenzmatrix von D. Sei A' die durch Streichen der Zeilen q und s aus A entstehende Matrix. Dann lauten die zu Kirchhoffs Gesetz gehörenden Nebenbedingungen

$$A'x = 0.$$

Sei w^T die Zeile q von A und beschreibe $c \geq 0$ die Kapazität von N. Dann lautet das Problem, einen maximalen Fluss auf N zu finden,

$$\begin{aligned} &\max w^T x. \\ \text{s.d. } &A'x = 0 \\ &x \leq c \\ &x \geq 0 \end{aligned} \tag{24.26}$$

Die Koeffizientenmatrix A' dieses LP ist total unimodular, weil A nach Lemma 24.18 total unimodular ist. Wegen Satz 24.15 hat dieses LP eine optimale ganzzahlige Lösung x^*. Das duale Problem lautet

$$\begin{aligned} &\min c^T y. \\ \text{s.d. } &A'^T z + y \geq w \\ &y \geq 0 \end{aligned} \tag{24.27}$$

Für die Nebenbedingungen gilt

$$\begin{pmatrix} A'^T & I \\ 0 & I \end{pmatrix} \begin{pmatrix} z \\ y \end{pmatrix} \geq \begin{pmatrix} w \\ 0 \end{pmatrix}.$$

Da A nach Lemma 24.18 total unimodular ist, ist auch diese Koeffizientenmatrix total unimodular. Also hat das duale Problem nach Satz 24.15 eine optimale ganzzahlige Lösung (y^*, z^*). Wir erweitern den Vektor z^* um Komponenten q und s anhand

$$\hat{z}_v^* = \begin{cases} -1 \text{ falls } v = q, \\ 0 \text{ falls } v = s, \\ z_v^* \text{ sonst.} \end{cases}$$

Die Menge $S = \{v \in V \mid \hat{z}_v^* \leq -1\}$ ist definitionsgemäß einen Schnitt in N. Es gilt $A^T \hat{z}^* + y^* = (A'^T z^* - w) + y^* \geq 0$. Also gilt für jede Kante vw die Ungleichung $y_{vw}^* + \hat{z}_v^* - \hat{z}_w^* \geq 0$. Daraus erhellt sich für jede Kante vw mit $v \in S$ und $w \notin S$ sofort $y_{vw}^* \geq 1$, woraus sich aus Optimalitätsgründen

$y^*_{vw} = 1$ ergibt. Weiter gilt für jede Kante vw mit $w \in S$ und $v \notin S$, dass $y^*_{vw} + \hat{z}^*_v - \hat{z}^*_w \geq y^*_{vw} \geq 0$, mithin aus Optimalitätsgründen $y^*_{vw} = 0$. Schließlich ist aus Optimalitätsgründen $y^*_{vw} = 0$ für jede Kante mit $v, w \in S$ oder $v, w \notin S$. Also definiert S einen minimalen Schnitt in N mit der Kapazität $c^T y^*$. Nach dem Dualitätssatz haben die beiden relaxierten LP (24.26) und (24.27) den gleichen Zielfunktionswert, d. h., $w^T x^* = c^T y^*$. Damit haben wir den Satz von Ford-Fulkerson mithilfe der linearen Optimierung gezeigt.

24.5 Das Simplex-Verfahren

Wir betrachten das LP

$$\begin{aligned} &\max c^T x \\ \text{s.d. } &Ax = b \\ &x \geq 0 \end{aligned} \tag{24.28}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$ mit $m \leq n$ und $r(A) = m$.

Sei $x^{(0)}$ eine Ecke von $P = \{x \mid Ax = b, x \geq 0\}$ mit der Basis $\{a^{(i)} \mid i \in I\}$. Dann gibt es eindeutig bestimmte Zahlen $\alpha_{ij} \in \mathbb{R}$, $i \in I$ und $j \in \underline{n}$, mit

$$a^{(j)} = \sum_{i \in I} \alpha_{ij} a^{(i)}, \quad 1 \leq j \leq n. \tag{24.29}$$

Da $\{a^{(i)} \mid i \in I\}$ eine Basis ist, folgt $\alpha_{ii} = 1$ für alle $i \in I$ und $\alpha_{ij} = 0$ für alle $i, j \in I$ mit $i \neq j$.

Lemma 24.20. *Für jedes $x \in P$ gilt*

$$c^T x = c^T x^{(0)} - \sum_{j \notin I} (d_j - c_j) x_j, \tag{24.30}$$

wobei

$$d_j = \sum_{i \in I} \alpha_{ij} c_i, \quad j \notin I. \tag{24.31}$$

Beweis. Für die Ecke $x^{(0)}$ von P gilt $\sum_{i \in I} x^{(0)}_i a^{(i)} = b$. Andererseits folgt mit (24.30)

$$b = \sum_{j=1}^{n} x_j a^{(j)} = \sum_{j=1}^{n} x_j \left(\sum_{i \in I} \alpha_{ij} a^{(i)} \right) = \sum_{i \in I} \left(\sum_{j=1}^{n} \alpha_{ij} x_j \right) a^{(i)}.$$

Durch Koeffizientenvergleich erhellt sich $x^{(0)}_i = \sum_{j=1}^{n} \alpha_{ij} x_j = x_i + \sum_{j \notin I} \alpha_{ij} x_j$ für alle $i \in I$. Somit folgt

$$c^T x = \sum_{j=1}^{n} c_j x_j = \sum_{i \in I} c_i x_i + \sum_{j \notin I} c_j x_j = \sum_{i \in I} c_i x^{(0)}_i - \sum_{j \notin I} \sum_{i \in I} (\alpha_{ij} c_i - c_j) x_j.$$

Damit ist die Aussage bewiesen. □

Satz 24.21. *Unter den obigen Voraussetzungen gilt:*

1. *Sei $d_j \geq c_j$ für alle $j \notin I$. Dann ist $x^{(0)}$ eine optimale Lösung des LP (24.28).*
2. *Es gibt ein $j \notin I$ mit $d_j < c_j$, so dass $\alpha_{ij} \leq 0$ für alle $i \in I$. Dann ist das LP (24.28) nicht lösbar.*
3. *Für jedes $j \notin I$ mit $d_j < c_j$ gibt es ein $i \in I$ mit $\alpha_{ij} > 0$. Dann gibt es eine Ecke $x^{(1)}$ von P mit $c^T x^{(1)} \geq c^T x^{(0)}$. Der Fall $c^T x^{(1)} = c^T x^{(0)}$ kann nur eintreten, wenn $x^{(0)}$ entartet ist.*

Beweis. Sei $d_j \geq c_j$ für alle $j \notin I$. Sei $x \in P$. Mit Lemma 24.20 und der Voraussetzung folgt $c^T x \leq c^T x^{(0)}$. Also ist $x^{(0)}$ optimal.

Sei $l \notin I$ mit $d_l < c_l$ und $\alpha_{il} \leq 0$ für alle $i \in I$. Sei $\delta > 0$. Wir definieren $x \in \mathbb{R}^n$ durch

$$x_i = \begin{cases} x_i^{(0)} - \delta\alpha_{il} & \text{falls } i \in I \\ \delta & \text{falls } i = l \\ 0 & \text{sonst.} \end{cases} \tag{24.32}$$

Aufgrund der Annahme ist $x \geq 0$ und mit (24.29) gilt

$$\begin{aligned} Ax &= \sum_{i \in I} x_i a^{(i)} + \delta a^{(l)} \\ &= \sum_{i \in I} x_i^{(0)} a^{(i)} - \delta \sum_{i \in I} \alpha_{il} a^{(i)} + \delta a^{(l)} \\ &= \sum_{i \in I} x_i^{(0)} a^{(i)} - \delta a^{(l)} + \delta a^{(l)} = b. \end{aligned}$$

Also ist $x \in P$. Nach (24.30) gilt $c^T x = c^T x^{(0)} - (d_l - c_l)\delta$. Wegen $d_l < c_l$ ist $c^T x \geq c^T x^{(0)}$. Da $\delta > 0$ beliebig wählbar ist, wird $c^T x$ beliebig groß. Also ist das LP (24.28) nicht lösbar.

Seien $i \in I$ und $l \notin I$ mit $d_l < c_l$ und $\alpha_{il} > 0$. Wir setzen

$$\delta = \min\{\frac{x_i^{(0)}}{\alpha_{il}} \mid i \in I, \alpha_{il} > 0\}.$$

Sei x wie in (24.32) definiert. Sei $k \in I$ mit $\delta = x_k^{(0)}/\alpha_{kl}$. Für jedes $i \in I$ mit $\alpha_{il} > 0$ gilt

$$x_i = x_i^{(0)} - \frac{x_k^{(0)}}{\alpha_{kl}}\alpha_{il} \geq x_i^{(0)} - \frac{x_i^{(0)}}{\alpha_{il}}\alpha_{il} = 0,$$

also $x \geq 0$. Wie im zweiten Fall folgt $x \in P$ und $c^T x = c^T x^{(0)} - (d_l - c_l)\delta$.

Wegen $x_k = x_k^{(0)} - \frac{x_k^{(0)}}{\alpha_{kl}}\alpha_{kl} = 0$ liegt die zu x gehörige Basis in $\{a^{(i)} \mid i \in (I \setminus \{k\}) \cup \{l\}\}$. Angenommen, diese Menge wäre linear abhängig. Dann ist

$$\sum_{i \in I, i \neq k} \kappa_i a^{(i)} + \kappa_l a^{(l)} = 0,$$

wobei nicht alle Koeffizienten verschwinden. Da $\{a^{(i)} \mid i \in I\}$ eine Basis ist, folgt $\kappa_l \neq 0$ und somit

$$a^{(l)} = \sum_{i \in I, i \neq k} \left(-\frac{\kappa_i}{\kappa_l}\right) a^{(i)}.$$

Durch Koeffizientenvergleich mit (24.29) erhellt sich widersprüchlicherweise $\alpha_{kl} = 0$. Also ist obige Menge linear unabhängig. Nach Satz 24.11 ist der zugehörige Vektor x eine Ecke von P, der die gewünschten Eigenschaften besitzt. □

Wir untersuchen den dritten Fall des obigen Satzes näher. In ihm wird aus einer gegebenen Ecke $x^{(0)}$ durch einen Basiswechsel eine neue Ecke $x^{(1)}$ bestimmt. Dabei wird die Basis $\{a^{(i)} \mid i \in I\}$ von $x^{(0)}$ durch die Basis $\{a^{(i)} \mid i \in I'\}$ von $x^{(1)}$ mit $I' = (I \setminus \{k\}) \cup \{l\}$ ersetzt. Die dazu notwendigen Daten werden in einem so genannten *Tableau* angeordnet

		x_i	x_j	x_k	x_l	…
x_i	$x_i^{(0)}$	1	α_{ij}	0	α_{il}	
x_k	$x_k^{(0)}$	0	α_{kj}	1	α_{kl}	
⋮	⋮					
	$c^T x^{(0)}$	0	f_j	0	f_l	…

(24.33)

wobei $f_i = d_i - c_i$ für alle $1 \leq i \leq n$. Für den Basiswechsel spielt das so genannte *Pivotelement* $\alpha_{kl} > 0$ eine wichtige Rolle. Das Tableau zur neuen Basis wird anhand eines so genannten *Austauschschritts* erhalten, in den die k-te Zeile und l-te Spalte involviert sind

		x_i	x_j	x_k	x_l	…
x_i	$x_i^{(1)}$	1	α'_{ij}	α'_{ik}	0	
x_l	$x_l^{(1)}$	0	α'_{lj}	α'_{kl}	1	
⋮	⋮					
	$c^T x^{(1)}$	0	f'_j	f'_k	0	…

(24.34)

Nach (24.29) ist

$$a^{(k)} = \sum_{i \in I, i \neq k} \left(-\frac{\alpha_{il}}{\alpha_{kl}}\right) a^{(i)} + \frac{1}{\alpha_{kl}} a^{(l)} \tag{24.35}$$

und folglich

$$\begin{aligned} a^{(j)} &= \sum_{i\in I, i\neq k} \alpha_{ij} a^{(i)} + \alpha_{kj} a^{(k)} \\ &= \sum_{i\in I, i\neq k} \left(\alpha_{ij} - \frac{\alpha_{il}}{\alpha_{kl}}\alpha_{kj}\right) a^{(i)} + \frac{\alpha_{kj}}{\alpha_{kl}} a^{(l)}, \quad j \neq l. \end{aligned} \tag{24.36}$$

Sei A_i die i-te Zeile von $A = (\alpha_{ij})$ bzw. (24.33) und A'_i die i-te Zeile von $A' = (\alpha'_{ij})$ bzw. (24.34). Mit (24.35) und (24.36) folgt

$$A'_i = \begin{cases} A_i - \frac{\alpha_{il}}{\alpha_{kl}} A_k & \text{falls } i \neq l \\ \frac{1}{\alpha_{kl}} A_k & \text{falls } i = l. \end{cases} \tag{24.37}$$

Weiter ist $x_l^{(1)} = \frac{1}{\alpha_{kl}} x_k^{(0)}$ und $x_i^{(1)} = x_i^{(0)} - \frac{\alpha_{il}}{\alpha_{kl}} x_k^{(0)}$ für alle $i \in I$ mit $i \neq l$. Also gilt (24.37) auch für die Eckenspalte.

Sei $f = d - c$, wobei $d_i = 0$ für alle $i \in I$, und $f' = d' - c$, wobei $d'_i = 0$ für alle $i \in I'$. Nach (24.31) ist $d = \sum_{i\in I} c_i A_i$ und $d' = \sum_{i\in I'} c_i A'_i$. Mit (24.31) und (24.37) folgt

$$\begin{aligned} f' &= \sum_{i\in I'} c_i A'_i - c \\ &= \sum_{i\in I, i\neq k} c_i \left(A_i - \frac{\alpha_{il}}{\alpha_{kl}} A_k\right) + \frac{1}{\alpha_{kl}} c_l A_k - c \\ &= \sum_{i\in I, i\neq k} c_i A_i - \sum_{i\in I, i\neq k} \frac{\alpha_{il} c_i}{\alpha_{kl}} A_k + \frac{c_l}{\alpha_{kl}} A_k - c \\ &= \left(\sum_{i\in I} c_i A_i - c_k A_k\right) - \left(\frac{d_l}{\alpha_{kl}} A_k - c_k A_k\right) + \frac{c_l}{\alpha_{kl}} A_k - c \\ &= d - \frac{f_l}{\alpha_{kl}} A_k - c \\ &= f - \frac{f_l}{\alpha_{kl}} A_k. \end{aligned} \tag{24.38}$$

Nach (24.30) gilt schließlich $c^T x^{(1)} = c^T x^{(0)} - \delta f_l = c^T x^{(0)} - \frac{f_l}{\alpha_{kl}} x_k^{(0)}$. Also wird das Tableau (24.34) aus dem Tableau (24.33) durch die in (24.37) beschriebene Gauss-Elimination erhalten.

Um das erste Tableau aufzustellen, werden zwei Fälle unterschieden:

1. Sei ein LP in kanonischer Form vorgelegt

$$\begin{aligned} &\max c^T y. \\ \text{s.d. } &Ay \leq b \\ &y \geq 0 \end{aligned} \tag{24.39}$$

Sei $b \geq 0$. Dieses LP wird in Standardform transformiert

$$\begin{aligned} &\max c^T x, \\ \text{s.d. } &(A \mid I_m)\, x = b \\ &x \geq 0 \end{aligned} \tag{24.40}$$

wobei I_m die $m \times m$-Einheitsmatrix bezeichnet. Dieses LP besitzt $x = (0, b)^T$ als zulässige Lösung. Nach Satz 24.11 ist x eine Ecke mit den Einheitsvektoren von I_m als Basis. Daraus erhalten wir das Anfangstableau

		x_1	$\dots$	x_n	x_{n+1}	$\dots$	x_{n+m}
x_{n+1}	b_1	α_{11}	$\dots$	α_{1n}	1		0
$\dots$						$\ddots$	
x_{n+m}	b_m	α_{m1}	$\dots$	α_{mn}	0		1
	0	$-c_1$	$\dots$	$-c_n$	0	$\dots$	0

(24.41)

2. Sei ein LP in Standardform gegeben

$$\begin{aligned} &\max c^T x \\ \text{s.d. } &Ax = b \\ &x \geq 0 \end{aligned} \tag{24.42}$$

O.B.d.A. sei $b \geq 0$. Wir betrachten zuerst das LP

$$\begin{aligned} &\min 1^T y \\ \text{s.d. } &Ax + y = b \\ &x \geq 0,\ y \geq 0 \end{aligned} \tag{24.43}$$

Dieses LP hat wegen $b \geq 0$ die zulässige Lösung $(x, y) = (0, b)$, die nach Satz 24.11 eine Ecke darstellt.

Lemma 24.22. *Das LP (24.42) hat eine zulässige Lösung genau dann, wenn das LP (24.43) den optimalen Wert 0 hat.*

Beweis. Sei x zulässig in (24.42). Dann ist $y^* = b - Ax = 0$ und somit (x, y^*) eine optimale Lösung von (24.43). Sei umgekehrt (x^*, y^*) eine optimale Lösung von (24.43) mit $1^T y^* = 0$. Dann ist $y^* = 0$ und somit $Ax^* = Ax^* + y^* = b$. Also ist x^* zulässig in (24.42). □

Das LP (24.43) kann mithilfe des weiter unten beschriebenen Simplex-Verfahrens gelöst werden. Ist 0 der optimale Wert, dann liefert die optimale Lösung (x^*, y^*) wegen Satz 24.11 eine (möglicherweise entartete) Ecke x^* von (24.42).

Der *Simplex-Algorithmus* fasst unsere Überlegungen zusammen:

1. Stelle ein erstes Tableau auf.
2. Teste f_j:
   a) Falls $f_j \geq 0$ für alle $j \notin I$, dann ist die Lösung optimal.
   b) Es gibt ein $j \notin I$ mit $f_j < 0$, so dass $\alpha_{ij} \leq 0$ für alle $i \notin I$. Dann hat das LP keine optimale Lösung.
   c) Sonst wähle ein $l \notin I$ mit $f_l < 0$ und bestimme ein $k \in I$ mit $\frac{x_k^{(0)}}{\alpha_{kl}} \leq \frac{x_i^{(0)}}{\alpha_{il}}$ für alle $i \in I$ mit $\alpha_{il} > 0$.
3. Stelle ein neues Tableau auf, indem x_k und x_l vertauscht werden und gehe nach 2.

Beispiel 24.23. Gegeben sei das kanonische LP

$$\begin{aligned} & \max x_1 + x_2. \\ \text{s.d. } & x_1 + 2x_2 \leq 4 \\ & 2x_1 - x_2 \leq 3 \\ & x_2 \leq 1 \\ & x_i \geq 0 \quad \forall i \end{aligned}$$

Dieses LP wird durch Einführen von Schlupfvariablen transformiert in ein Standard-LP

$$\begin{aligned} & \max x_1 + x_2. \\ \text{s.d. } & x_1 + 2x_2 + x_3 = 4 \\ & 2x_1 - x_2 + x_4 = 3 \\ & x_2 + x_5 = 1 \\ & x_i \geq 0 \quad \forall i \end{aligned}$$

Dieses LP besitzt $x^{(0)} = (0, 0, 4, 3, 1)^T$ als Startecke. Das zugehörige Anfangstableau lautet

		x_1	x_2	x_3	x_4	x_5
x_3	4	1	2	1	0	0
x_4	3	**2**	−1	0	1	0
x_5	1	0	1	0	0	1
	0	−1	−1	0	0	0

Wir wählen $l = 1$ und bestimmen k durch $\min\{\frac{4}{1}, \frac{3}{2}\} = \frac{3}{2}$, also $k = 4$. Als nächstes Tableau ergibt sich

		x_1	x_2	x_3	x_4	x_5
x_3	$\frac{5}{2}$	0	$\frac{5}{2}$	1	$-\frac{1}{2}$	0
x_1	$\frac{3}{2}$	1	$-\frac{1}{2}$	0	$\frac{1}{2}$	0
x_5	1	0	**1**	0	0	1
	$\frac{3}{2}$	0	$-\frac{3}{2}$	0	$\frac{1}{2}$	0

Die nächste Ecke ist $x^{(1)} = (\frac{3}{2}, 0, \frac{5}{2}, 0, 1)^T$. Wir wählen $l = 2$ und bestimmen k anhand $\min\{\frac{5}{2}/\frac{5}{2}, \frac{1}{1}\} = 1$, also etwa $k = 5$. Das nächste Tableau lautet

		x_1	x_2	x_3	x_4	x_5
x_3	0	0	0	1	$-\frac{1}{2}$	$-\frac{5}{2}$
x_1	2	1	0	0	$\frac{1}{2}$	$\frac{1}{2}$
x_2	1	0	1	0	0	1
	3	0	0	0	$\frac{1}{2}$	$\frac{3}{2}$

Als Ecke erhalten wir $x^{(2)} = (2, 1, 0, 0, 0)^T$. Diese Lösung ist optimal. Das ursprüngliche kanonische LP besitzt demnach die optimale Lösung $(2, 1)^T$.

Es gibt lineare Programme, die das Simplex-Verfahren nur in exponentieller Laufzeit lösen kann. Allerdings lässt sich mithilfe eines probabilistischen Modells zeigen, dass der Simplex-Algorithmus im Durchschnitt polynomiale Laufzeit besitzt. Andererseits gibt es polynomiale Verfahren, um lineare Programme zu lösen. Dazu zählen die Algorithmen von Khachiyan und Karmarkar. Lineare Programmierung liegt also in der Klasse P. Demgegenüber ist ganzzahlige lineare Programmierung NP-vollständig. Wer allerdings glaubt, dass die polynomialen Verfahren in punkto Laufzeit besser sind als das Simplex-Verfahren, der irrt. Der Simplex-Algorithmus ist den polynomiellen Verfahren im Durchschnitt überlegen.

Selbsttestaufgaben

24.1. Die Funktion

$$c^T x = c_1 x_1 + \ldots + c_n x_n$$

ist durch geeignete Wahl der Variablen $x_1, \ldots, x_n$ zu maximieren unter Einhaltung der Nebenbedingungen

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{k1}x_1 + \ldots + a_{kn}x_n &= b_k \\ a'_{11}x_1 + \ldots + a'_{1n}x_n &\leq b'_1 \\ &\vdots \\ a'_{l1}x_1 + \ldots + a'_{ln}x_n &\leq b'_l \end{aligned}$$

und der Vorzeichenbedingungen

$$x_1 \geq 0, \quad \ldots \quad , x_r \geq 0.$$

Zeige, dass diese Aufgabe in *kanonische Form* gebracht werden kann

$$\begin{aligned} &\max c^T x. \\ \text{s.d. } &Ax \leq b \\ &x \geq 0 \end{aligned} \tag{24.44}$$

Zeige, dass sich obige Aufgabe in *Standardform* bringen lässt

$$\begin{aligned} &\max c^T x. \\ \text{s.d. } &Ax = b \\ &x \geq 0 \end{aligned} \tag{24.45}$$

24.2. Ein alkoholisches Getränk soll aus drei flüssigen Zutaten hergestellt werden, wobei hinsichtlich Anteil und Mischung der Zutaten Einschränkungen bestehen:

	Zutaten			Gehalt	
Bestandteile	A	B	C	min	max
Alkohol	10 %	15 %	5 %	11 %	12 %
Zucker	8 g/l	10 g/l	22 g/l	10 g/l	16 g/l
Aromastoffe	2 E	6 E	10 E	2 E	7 E
A	1	0	0	20 %	
B	0	1	0		40 %
C	0	0	1		50 %
Preis/l	7	5	3		

24.3. Ein Tischler fertigt Tische und Stühle. Er verkauft Tische für 40 Euro und Stühle für 10 Euro. Der Tischler arbeitet 40 Stunden pro Woche. Für einen Tisch benötigt er 8 Stunden und für einen Stuhl 4 Stunden. Er stellt mindestens drei Mal so viele Stühle wie Tische her.

Stelle ein zugehöriges LP hinsichtlich Profitmaximierung auf und löse es. Wie lautet die optimale Lösung des zugeordneten ILP?

24.4. Beweise den Satz 24.8.

24.5. Zeige, dass der zulässige Bereich $P = \{x \mid \mathbf{Ax} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ konvex ist.

24.6. Zeige, dass die Menge aller optimalen Lösungen eines LP konvex ist.

24.7. Sei $A \subseteq \mathbb{R}^n$. Zeige

- $A \subseteq \text{conv}(A)$.
- $\text{conv}(A)$ ist konvex.
- Ist $A' \subseteq \mathbb{R}^n$ konvex mit $A \subseteq A'$, dann ist $\text{conv}(A) \subseteq A'$.

24.8. Sei $A \subseteq \mathbb{R}^n$ eine konvexe Menge und $f : \mathbb{R}^n \to \mathbb{R}^n$ eine lineare Abbildung. Zeige, dass $f(A)$ ebenfalls konvex ist.

24.9. Löse das LP

$$\begin{aligned} &\max 49x_1 + 24x_2 \\ \text{s.d. } &x_1 + 2x_2 \leq 8 \\ &25x_1 + 12x_2 \leq 67 \\ &x_i \geq 0 \quad \forall i \end{aligned}$$

Zeige, dass die optimale Lösung des zugehörigen ILP nicht durch Runden der optimalen Lösung des LP hervorgeht.

24.10. Löse das folgende LP durch Inspektion der Ecken

$$\begin{aligned} &\min x_1 + 2x_2 - x_3 + x4 + 2x_5 \\ \text{s.d. } &x_1 - x_2 + 5x_3 - x_4 + x_5 = 8 \\ &5x_1 - 4x_2 + 13x_3 - 2x_4 + x_5 = 20 \\ &x_i \geq 0 \quad \forall i \end{aligned}$$

24.11. Löse das folgende LP anhand des Simplex-Algorithmus'

$$\begin{aligned} & \max 2x_1 + 3x_2 \\ \text{s.d. } & x_1 - x_2 \leq 2 \\ & -x_1 + x_2 \leq 2 \\ & 2x_1 + x_2 \leq 8 \\ & x_1 + 3x_2 \leq 12 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

24.12. Sei $G = (V, E)$ ein Graph. Eine *Kantenüberdeckung* von G ist eine Menge von Kanten $E' \subseteq E$, so dass jeder Knoten in G mit mindestens einer Kante in E' inzidiert. Eine *minimale Kantenüberdeckung* von G ist eine Kantenüberdeckung von G mit minimaler Mächtigkeit.

Spezifiziere das Problem, eine minimale Kantenüberdeckung von G zu finden, als ILP. Relaxiere dieses ILP und gib das zugehörige duale LP an. Restringiere die Variablen des dualen LP auf ganzzahlige Werte und interpretiere dieses Problem.