Quantitative Methoden in der Entscheidungsunterstützung

Anita Schöbel und Marie Schmidt

12. November 2009

Inhaltsverzeichnis

Kapitel 1

Vom Problem zum Modell

Viele Probleme in Wirtschaft, Industrie und sogar im täglichen Leben sind Optimierungsprobleme. Der Prozess bei der Lösung eines Anwendungsproblems ist in Abbildung 1.1, dem so genannten *Modellierungszyklus*, veranschaulicht: Zunächst versucht man, das praktische Problem in die Sprache der Mathematik zu übersetzen. Dieser Prozess wird als *Modellierung* bezeichnet. Dann sucht man im nächsten Schritt ein passendes Lösungsverfahren und interpretiert im dritten Schritt die daraus gewonnenen Ergebnisse. Passt alles, ist das Problem gelöst. In vielen Fällen merkt man aber bei der Interpretation der Ergebnisse, welche weiteren Bedingungen noch wichtig gewesen wären oder wo das Modell vielleicht zu ungenau war oder das Problem nicht richtig beschrieben hat, so dass man das Modell anpassen und den Zyklus mit dem neuen Modell erneut beginnen wird.

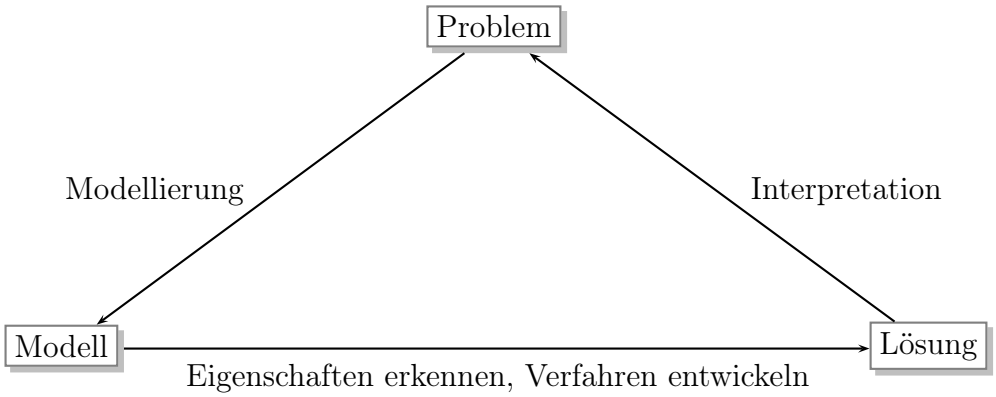


Abbildung 1.1: Lösen eines Anwendungsproblems

Dieses Kapitel widmet sich hauptsächlich der Modellierung, also dem Übersetzen der Problembeschreibung in die Sprache der Mathematik. Die Fragestellungen bei der Modellierung sind die folgenden:

- *Was kann ich beeinflussen?*
  → Finde die Entscheidungsfreiheiten und erhalte daraus *Variablen*.
- *Welche Bedingungen sind zu beachten?*
  → Erkenne die Zulässigkeitsbedingungen und formuliere sie als *Nebenbedingungen* in Form von Gleichungen oder Ungleichungen.

- *Wie bewertet man die Qualität der Lösung?*
  → Erkenne das richtige Gütekriterium und definiere daraus die *Zielfunktion.*

Beispiel 1: Kaninchengehege

Jemand möchte ein rechteckiges Gehege für seine Kaninchen abzäunen. Dazu hat er bereits 5 Meter Kaninchenzaun besorgt. Wie sollen die Kantenlängen des Rechtecks gewählt werden, so dass die Kaninchen eine möglichst große Fläche bekommen?

Variablen: Es ist schon vorgegeben, dass das Kaninchengehege rechteckig werden soll, die Variablen sind damit die Länge x und die Breite y des Geheges.

Nebenbedingungen: Die Gesamtlänge des Zauns darf 5 Meter nicht überschreiten. Sie setzt sich zusammen aus den Kantenlängen:

$$2x + 2y \leq 5.$$

Außerdem können die Kantenlängen nicht negativ werden:

$$x \geq 0$$
$$y \geq 0.$$

Diese Nichtnegativitätsbedingungen werden oft vergessen, da sie so offensichtlich erscheinen (schließlich gibt es keine Zäune mit negativer Länge). Ohne diese wichtigen Informationen ergeben sich aber eventuell Lösungen mit $x < 0$ oder $y < 0$, die in der Anwendung natürlich nicht von Nutzen sind.

Zielfunktion: Maximiere die Fläche des Rechtecks xy.

Es ergibt sich also das folgende Modell:

$$\begin{aligned} \max\ & x \cdot y \\ \text{s.d. } & 2x + 2y \leq 5 \\ & x, y \geq 0 \end{aligned}$$

Um eine Lösung zu finden, beobachten wir die folgende Eigenschaft: Eine Lösung mit $2x+2y < 5$ *kann nie optimal sein kann, denn in diesem Fall kann man das Rechteck noch vergrößern, indem man ein kleines Stückchen an die Länge oder Breite anfügt.*
Also setzen wir $2x + 2y = 5$ *und lösen nach* y *auf. Wir erhalten:*

$$y = \frac{1}{2} \cdot (5 - 2x) = \frac{5}{2} - x$$

Daraus ergibt sich ein vereinfachte Problem mit nur noch einer Variablen, nämlich der Länge x*:*

$$\begin{aligned} max\ & x \cdot \left(\frac{5}{2} - x\right) \\ s.d.\ & 0 \leq x \leq \frac{5}{2} \end{aligned}$$

Die Zielfunktion $f(x) = \frac{5}{2}x - x^2$ dieses Problems ist eine Parabel. Durch Ableiten und Nullsetzen erhält man

$$f'(x) = \frac{5}{2} - 2x$$
$$\frac{5}{2} - 2x = 0 \Leftrightarrow 2x = \frac{5}{2} \Leftrightarrow x = \frac{5}{4}.$$

Zeichnen der Parabel oder Berechen der zweiten Ableitung an der Stelle $x^* = \frac{5}{4}$ ergibt, dass es sich hier tatsächlich um ein Maximum handelt. Nun bestimmt man das zu x^* passende y^* durch Einsetzen:

$$y^* = \frac{5}{2} - x^* = \frac{5}{2} - \frac{5}{4} = \frac{5}{4}$$

Folglich gibt ein quadratisches Gehege mit Seitenlängen von 1.25 *Metern den Kaninchen den meisten Auslauf, nämlich* 1.5625 *Quadratmeter.*

Die Lösung soll nun in der Praxis interpretiert werden — in unserem Beispiel geht der Kaninchenbesitzer mit den fertig berechneten Maßen in den Garten und möchte den Zaun aufstellen. Als er seinen Kaninchenzaun herauskramt, sieht er aber, dass dieser aus einzelnen 50cm-Stücken besteht, die nicht gebogen werden können. Damit kann er die berechnete Lösung, Länge=Breite=1.25 Meter nicht realisieren. Das Modell muss also verändert und erneut gelöst werden.

Wir passen das Modell an. Die Variablen und die Zielfunktion können übernommen werden, aber wir müssen die neue Bedingung beachten, dass die Seitenlängen Vielfache von $0.50 = \frac{1}{2}$ sein müssen. Dazu gibt es (mindestens) zwei Möglichkeiten. Der erste Versuch schränkt die zulässigen Werte für x auf Vielfache von 50cm ein. Wir erhalten also folgende Formulierung.

$$\begin{aligned} \max\ & x\left(\frac{5}{2} - x\right) \\ \text{s.d. } & 0 \leq x \leq \frac{5}{2} \\ & x \in \left\{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}\right\} \end{aligned}$$

Diese Formulierung ist für größere Probleme unschön, da man nichts anderes machen kann, als alle möglichen Werte für x durchzuprobieren. Zum Lösen besser geeignet ist daher die folgende Formulierung, in der man eine neue Variable $z := 2x$ einführt und fordert dass z nur ganzzahlige Werte annimmt. Wir ersetzen in unserem Modell also x durch $\frac{z}{2}$ und erhalten

$$\begin{aligned} \max\ & \frac{z}{2}\left(\frac{5}{2} - \frac{z}{2}\right) \\ \text{s.d. } & 0 \leq z \leq 5 \\ & z \in \mathbb{Z} \end{aligned}$$

Um auch hier eine Lösung zu erhalten, ist es in unserem kleinen Beispiel am einfachsten, für z alle ganzzahligen Zahlenkombinationen zwischen 0 und 5 ducrhzuprobieren (bzw. alle möglichen Lösungen $\left\{0, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}\right\}$ in dem ersten Modell). Man erhält die Optimallösung $x^* = 1$ Meter und $y^* = 1.5$ Meter (oder $x^* = 1.5$ Meter und $y^* = 1$ Meter) mit einem optimalen Zielfunktionswert von 1.5 Quadratmetern.

Diese Lösung lässt sich realisieren, das Kaninchengehege kann also gebaut werden und der Modellierungszyklus muss nicht noch einmal durchlaufen werden.

Beispiel 2: Weineinkauf

Das zweite Beispiel ist dem Buch [MM98] entnommen und beschäftigt sich mit einem Weintrinker, der zusammen mit seiner Familie und Gästen $m = 500$ Flaschen Wein pro Jahr verbraucht. Er bezieht den Wein von einem 200 km entfernten Weingut, das er mit seinem eigenen Auto zum Weineinkauf besucht. Bei einem Preis von $0,25$€ pro Kilometer kostet ihn eine Fahrt 100€. Er möchte also nicht zu oft einkaufen. Andererseits wird durch den von ihm an Wein gehaltenen Lagerbestand Kapital gebunden, für das er sonst $p = 4\%$ Zinsen erhalten würde. Eine Flasche Wein kostet durchschnittlich $s = 4$€. Wie oft pro Jahr soll der Weintrinker zu dem Weingut fahren und wie viele Flaschen Wein soll er bei seinen Besuchen dort kaufen?
Wir erarbeiten das Modell wieder, indem wir uns Gedanken über Variablen, Nebenbedingungen und die Zielfunktion machen.

Variablen: Variablen sind die Anzahl n der Fahrten pro Jahr und die Einkaufsmenge x (in Flaschen). Da aber $x \cdot n = 500$ gelten soll, reicht es, eine der beiden Variablen zu wählen. Wir entscheiden uns hier, die Anzahl der pro Besuch erstandenen Flaschen x zu wählen.

Nebenbedingungen: Natürlich muss die Anzahl der Flaschen positiv sein und man kann auch nur ganze Flaschen kaufen. Wir erhalten also

$$x \in \mathbb{Z}, x \geq 1$$

Zielfunktion: Minimiert werden sollen die Kosten. Diese setzen sich zusammen aus den Fahrtkosten multipliziert mit der Häufigkeit der Fahrten und dem Zinsverlust, der durch das gebundene Kapital entsteht. Wir gehen approximativ davon aus, dass durchschnittlich die Hälfte des für den Wein bezahlten Geldes gebunden ist, also beträgt der Zinsverlust pro Jahr $\frac{x}{2} \cdot s$. Die Anzahl der pro Jahr nötigen Fahrten ergibt sich zu $\frac{m}{x} = \frac{500}{x}$. Damit können wir die zu minimierenden Kosten aufschreiben:

$$k(x) = \text{Fahrtkosten} \cdot \frac{m}{x} + \frac{p}{100} \cdot \frac{x}{2} \cdot s$$

Das Modell ist also das folgende:

$$\begin{aligned} \min\ & k(x) = \text{Fahrtkosten} \cdot \frac{m}{x} + \frac{p}{100} \cdot \frac{x}{2} \cdot s \\ \text{s.d. } & x \geq 0, \\ & x \in \mathbb{Z} \end{aligned}$$

Die Lösung wird wie im vorhergegangenen Beispiel durch Ableiten und Nullsetzen der Zielfunktion bestimmt:

$$\begin{aligned} k'(x) =& \frac{-\mathit{Fahrtkosten} \cdot m}{x^2} + \frac{p \cdot s}{200} \\ k'(x) =& 0 \Leftrightarrow x = x^* = \sqrt{\frac{200 \cdot m \cdot \mathit{Fahrtkosten}}{p \cdot s}} \\ k''(x) >& 0, \text{ es liegt also ein Minimum vor.} \end{aligned}$$

Man erhält die klassische Bestellmengenformel in Abhängigkeit der Fahrtkosten, der Bestellmenge m, des Zinssatzes p und des Durschnittspreises s.

Angewendet auf unser Beispiel ergibt sich damit (durch Runden des Ergebnisses) eine optimale Bestellmenge von 790 Flaschen.

$$x^* = \frac{200 \cdot 500 \cdot 100}{4 \cdot 4} \approx 790$$

Ist unser Problem damit gelöst? Als der Weintrinker erfährt, dass er 790 Flaschen kaufen soll und entsprechend nur alle 19 Monate zum Weingut fahren muss, meldet er gleich mehrere Bedenken an:

1. Er hatte vergessen zu erwähnen, dass in seinen Keller nur bis zu 400 Flaschen Wein passen.
2. In sein Auto passen sogar nur 200 Flaschen.
3. Außerdem erklärt der Weintrinker, dass er gerne mindestens zwei Mal im Jahr zu dem Weingut fahren möchte, um sich dort über die neuen Weine zu informieren.

Diese neuen Nebenbedingungen verändern das Modell. Wieder können wir die Variablen und die Zielfunktion unverändert lassen, müssen aber folgende Nebenbedingungen anfügen:

1. $x \leq 400$ weil in den Keller nur 400 Flaschen Wein passen.
2. $x \leq 200$ weil in das Auto nur 200 Flaschen Wein passen.
3. $n \geq 2$, weil der Weintrinker zwei Mal im Jahr zum Weingut fahren möchte. Durch die Beziehung $n \cdot x = 500$ kann man $n = \frac{500}{x}$ schreiben und erhält

$$\frac{500}{x} \geq 2 \text{ oder nach } x \text{ aufgelöst } x \leq 250.$$

Man sieht, dass die erste und die dritte Nebenbedingung redundant sind, denn wenn $x \leq 200$ gilt, sind diese beiden automatisch erfüllt. Wir fügen also nur die zweite Nebenbedingung zu unserem Modell hinzu und erhalten:

$$\begin{aligned} &\min k(x) \\ &\text{s.d. } x \leq 200 \\ &x \geq 0 \end{aligned}$$

Unsere ursprünglich bestimmte Lösung $x^* = 790$ ist also in dem veränderten Modell nicht zulässig. Um uns einen Überblick zu verschaffen, zeichnen wir die Zielfunktion und stellen fest, dass sie im Bereich von 0 bis $x^* = 790$ monoton fallend ist. Die Kosten werden also immer größer, je weniger Weinflaschen wir pro Besuch kaufen. Daraus ergibt sich, dass die größte zulässige Bestellmenge $x^*_{neu} = 200$ kostenminimal für unser Modell ist.
Der Weintrinker sollte also pro Einkauf 200 Flaschen mitnehmen. Er muss entsprechend in zwei Jahren insgesamt fünf Fahrten zu seinem Weingut einplanen.

Beispiel 3: Projektvergabe - Ein Rucksackproblem

Manche Forschungsorganisation vergeben Forschungsprojekte nach folgendem Schema:
Forscher können Anträge zu einem festen Zeitpunkt einreichen. Sind zu diesem Zeitpunkt n Anträge eingegangen, werden diese zunächst von unabhängigen Gutachtern mit Punktzahlen bewertet. Diese Punkte drücken den geschätzten gesellschaftlichen Nutzen des vorgeschlagenen Projektes aus. Die Punktzahl für Antrag i wird mit b_i bezeichnet, $i = 1, \ldots, n$. Gleichzeitig werden die für die Projekte $1, \ldots, n$ anfallenden Kosten c_i ermittelt. Das Gesamtbudget der Forschungsorganisation ist auf B beschränkt.
Welche Forschungsprojekte sollen im Rahmen des gegebenen Budgets gefördert werden, so dass der Gesamtnutzen maximiert wird?
Auch hier wollen wir zunächst ein passendes Modell erarbeiten.

Variablen: Man definiert für jedes Projekt i eine Variable x_i, die allerdings nur zwei Werte annehmen soll:

$$x_i = \begin{cases} 1 & \text{falls Projekt } i \text{ gewählt wird} \\ 0 & \text{sonst} \end{cases}$$

Nebenbedingungen: Die Summe der Kosten der ausgewählten Projekte darf das Budget B nicht überschreiten. Das lässt sich mit Hilfe der eben definierten Variablen folgendermaßen ausdrücken:

$$\sum_{i=1}^{n} x_i \cdot c_i \leq B$$

Man beachte, dass in diese Summe die Kosten der ausgewählten Projekte eingehen, da für diese $x_i = 1$ gilt. Für nicht ausgewählte Projekte ist dagegen $x_i = 0$, also auch $x_i \cdot c_i = 0$; sie werden in der Summe also nicht berücksichtigt. Es gehen also tatsächlich genau die ausgewählten Projekte in die Bestimmung des benötigten Budgets ein.

Zielfunktion: Das gleiche Prinzip wie bei der Nebenbedingung wird beim Aufstellen der Zielfunktion angewandt: Die Multiplikation der Bewertung b_i mit x_i sorgt dafür, dass nur die Bewertungen ausgewählter Projekte in die Gesamtbewertung einfließt. Wir erhalten entsprechend

$$\max \sum_{i=1}^{n} x_i \cdot b_i$$

Das Gesamtmodell ergibt sich damit zu

$$\begin{aligned} \max \ & \sum_{i=1}^{n} x_i \cdot b_i \\ \text{s.d.} \ & \sum_{i=1}^{n} x_i \cdot c_i \leq B \\ & x_i \in \{0,1\} \quad i = 1, \ldots, n \end{aligned}$$

Zur Verdeutlichung schauen wir ein Zahlenbeispiel an, in dem ein Budget von $B = 10$ zur Verfügung steht und insgesamt $n = 4$ Projekte vorliegen. Ihre Bewertungen und ihre Kosten sind in Tabelle 1.2 zusammengefasst.

Projekte		P1	P2	P3	P4
Kosten	c_i	6	5	5	2
Bewertung	b_i	9	7	4	1

Abbildung 1.2: Projekte mit Kosten und Bewertung

Das Modell zu diesem Problem lautet also:

$$\begin{aligned} \max \ & 9x_1 + 7x_2 + 4x_3 + 1x_4 \\ \text{s.d.} \ & 6x_1 + 5x_2 + 5x_3 + 2x_4 \leq 10 \\ & x_i \in \{0,1\} \end{aligned}$$

Durch Probieren erhält man die zulässigen Lösungen

- $x_1 = x_2 = 0, x_3 = x_4 = 1$ mit einem Nutzen von $1 + 4 = 5$
- $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$ mit einem Nutzen von $1 + 7 = 8$
- $x_1 = 1, x_2 = x_3 = 0, x_4 = 1$ mit einem Nutzen von $1 + 9 = 10$
- $x_1 = 0, x_2 = x_3 = 1, x_4 = 0$ mit einem Nutzen von $4 + 7 = 11$.

Die letzte Lösung ergibt den größten Nutzen, also werden die Projekte 2 und 3 zur Förderung gewählt.

Zum Vergleich schauen wir uns ein ähnliches Modell an, in dem wir aber nicht fordern, dass die Variablen $x_i \in \{0,1\}$ sein müssen. Das würde bedeuten, dass man Projekte auch anteilig fördern kann, also z.B. ein halbes Projekt 1 und ein Viertel Projekt 2. In diesem Fall ist das Problem leicht zu lösen:
Man bestimmt zunächst das Nutzen-Kosten-Verhältnis ($\frac{b_i}{c_i}$) aller Projekte und nimmt vom Projekt mit dem besten Verhältnis so viel wie möglich, dann von dem Projekt mit dem zweitbesten Verhältnis usw. Das Kosten-Nutzen-Verhältnis ist in Tabelle 1.3 dargestellt.
Angewendet auf unser Beispiel würde man entsprechend die folgende Lösung erhalten: Zuerst wählt man P1, das heißt, man setzt $x_1 = 1$ und hat dann noch ein Budget von $B - c_1 = 10 - 6 = 4$

Projekte		P1	P2	P3	P4
Kosten	c_i	6	5	5	2
Bewertung	b_i	9	7	4	1
Kosten-Nutzen-Verhältnis	$\frac{b_i}{c_i}$	$\frac{3}{2}$	$\frac{7}{5}$	$\frac{4}{5}$	$\frac{1}{2}$

Abbildung 1.3: Kosten-Nutzen-Verhältnis der Projekte

übrig. Das zweitgrößte Kosten-Nutzen-Verhältnis hat P2, also wird nun dieses Projekt ausgewählt. Da das Gesamtbudget von P1 und P2 aber schon 11 *beträgt, also die Budgetgrenze von* 10 *überschreitet, kann x_2 nicht ganz gefördert werden, sondern nur anteilig. Der noch ins Budget passende Anteil beträgt $x_2 = \frac{4}{5}$. Damit ist das Budget voll ausgenutzt, denn*

$$6x_1 + 5x_2 + 5x_3 + 2x_4 = 6 + \frac{4}{5} \cdot 5 = 10.$$

Die erreichte Bewertung beträgt

$$9x_1 + 7x_2 + 4x_3 + 1x_4 = 9 + 7 \cdot \frac{4}{5} = 14,6.$$

Für unser ursprüngliches Problem ist diese Lösung nicht zulässig; hier muss entschieden werden, ob ein Projekt ganz oder garnicht gefördert wird. Wie man in diesem Beispiel sieht, kann die Lösung eines Problems mit ganzzahligen oder boolschen Variablen ohne die Ganzzahligkeitsbedingung sehr stark von der eigentlichen Lösung abweichen. Im allgemeinen kann man also hieraus nicht auf die Lösung des ganzzahligen oder boolschen Problems schließen.

Beispiel 4: Aufgabenverteilung

Es sollen n Aufgaben auf m Personen verteilt werden (z.B. für eine Partyvorbereitung oder in einer Firma). Dabei soll jede Aufgabe von einer einzigen Person erledigt werden und jede Person soll höchstens eine Aufgabe übernehmen. Wir nehmen an, dass wir mehr Personen als Aufgaben haben, also $m \geq n$.
Außerdem sei bekannt, wie lange Person i zur Ausführung von Aufgabe j braucht, nämlich c_{ij} Zeiteinheiten. Wer soll welche Aufgabe übernehmen, so dass der Gesamtaufwand möglichst klein ist?

Dieses Problem ist als *Zuordnungsproblem* bekannt. Wir modellieren es als ganzzahliges Programm. (In Kapitel 3 wird uns das gleiche Problem wieder begegnen; wir werden es dort als Graph veranschaulichen.)

Variablen: Auch dieses Problem wird mit boolschen Variablen modelliert:

$$x_{ij} = \begin{cases} 1 & \text{falls Person } i \text{ Aufgabe } j \text{ erledigt} \\ 0 & \text{sonst} \end{cases}$$

Nebenbedingungen: Wir haben verschiedene Typen von Nebenbedingungen.

- Alle Aufgaben müssen erledigt werden.

$$\sum_{i=1}^{m} x_{ij} \geq 1 \qquad \forall \text{ Aufgaben } j$$

- Niemand darf mehr als eine Aufgabe erhalten.

$$\sum_{j=1}^{n} x_{ij} \leq 1 \qquad \forall \text{ Personen } i$$

- und natürlich die Beschränkung der Variablen:

$$x_{ij} \in \{0,1\} \forall \text{ Aufgaben } j \text{ und Personen } i$$

Zielfunktion: In der Zielfunktion sollen die Kosten minimiert werden. Wir erhalten:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

Damit ergibt sich das folgende Modell:

$$\begin{aligned}
&\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
&\sum_{i=1}^{m} x_{ij} \geq 1 \qquad \forall \text{ Aufgaben } j \\
&\sum_{j=1}^{n} x_{ij} \leq 1 \qquad \forall \text{ Personen } i \\
&x_{ij} \in \{0,1\} \qquad \forall \text{ Aufgaben } j \text{ und Personen } i
\end{aligned}$$

Kapitel 2

Lineare und ganzzahlige Programmierung

In diesem Abschnitt soll eine kurze Einführung in die lineare und in die ganzzahlige Optimierung gegeben werden. Beide Themen sind sehr gut untersucht und bieten in der Mathematik genügend Stoff für umfassende Vorlesungsreihen. Wir werden hier entsprechend nicht auf die mathematischen Einzelheiten und Herleitungen eingehen, sondern versuchen vielmehr, die geometrische Veranschaulichung der linearen und der ganzzahligen Optimierung zu vermitteln, Unterschiede zu verstehen und Lösungsverfahren zu nennen.

2.1 Lineare Programmierung

Wir beginnen mit einem Beispiel zur linearen Programmierung, nämlich mit dem oft zitierten Standardbeispiel zur Produktionsplanung.

Lineare Optimierung am Beispiel der Produktionsplanung

Eine Firma stellt zwei Produkte P1 und P2 her. Der Verkaufserlös von P1 beträgt 3 €, der von P2 beträgt 5 €. Eine Absatzanalyse zeigt, dass die Firma davon ausgehen kann, dass alle produzierten Einheiten auch verkauft werden.
Zur Herstellung der Produkte sind drei Maschinen **A**, **B**, **C** nötig. Die Bearbeitungszeiten der Produkte P1 und P2 auf den drei Maschinen sind wie folgt angegeben:

	Zeit für P1	Zeit für P2	Kapazität der Maschine
Maschine A	1	2	170
Maschine B	1	1	150
Maschine C		3	180

Die Maschinen stehen dabei nicht die ganze Zeit zur Verfügung, sondern nur für eine bestimmte Stundenzahl (z.B. pro Monat). Diese ist in der letzten Spalte eingetragen. Es stellt sich die Frage, wie viel die Firma von welchem Produkt produzieren soll, um ihren Gewinn zu maximieren.

Nach dem Schema aus Kapitel 1 stellen wir ein Modell für dieses Problem auf.

Variablen: Wir benötigen zwei Variablen:

$$\begin{aligned} x_1 &= \text{Menge an P1} \\ x_2 &= \text{Menge an P2} \end{aligned}$$

Zielfunktion: Die Firma will ihren Verkaufserlös maximieren. Dieser kann wie folgt berechnet werden:

$$\text{Gewinn} = f(x_1, x_2) = 3x_1 + 5x_2.$$

Nebenbedingungen: Bei den Nebenbedingungen müssen die Kapazitäten der drei Maschinen A, B und C beachtet werden. Beispielsweise kann man (pro Monat) nicht mehr als $\frac{180}{3} = 60$ Einheiten von P2 herstellen, sonst wäre Maschine C überlastet. Für die erste und zweite Maschine müssen die Mengen für beide Produkte berücksichtigt werden. Wir erhalten die folgenden Bedingungen:

$$\begin{aligned} x_1 + 2x_2 &\leq 170 \\ x_1 + x_2 &\leq 150 \\ 3x_2 &\leq 180 \\ x_1, x_2 &\geq 0. \end{aligned}$$

Die letzte Bedingung sagt, dass nur positive Mengen produziert werden dürfen und darf nicht vergessen werden!

Das Modell sieht also wie folgt aus.

$$\max 3x_1 + 5x_2.$$

so dass

$$\begin{aligned} x_1 + 2x_2 &\leq 170 \\ x_1 + x_2 &\leq 150 \\ 3x_2 &\leq 180 \\ x_1, x_2 &\geq 0. \end{aligned}$$

Die Menge aller möglichen Produktionspläne (x_1, x_2) kann man geometrisch veranschaulichen. Betrachten wir dazu die erste Ungleichung $x_1 + 2x_2 \leq 170$. Die Menge aller Punkte (x_1, x_2), für die $x_1 + 2x_2 = 170$ gilt, ist eine Gerade, die man in der Standardform als

$$x_2 = \frac{170}{2} - x_1$$

schreiben kann. Sie hat also den y-Achsenabschnitt 85 und eine Steigung von -1. Alle Punkte unterhalb der Geraden erfüllen die Ungleichung $x_1 + 2x_2 \leq 170$, die Punkte oberhalb der Geraden führen zu Produktionsplänen, die die Kapazität der Maschine A überschreiten. Analog erhält man Geraden für die Maschinen B und C; außerdem wird der zulässige Bereich wegen der Nicht-Negativitätsbedingungen $x_1, x_2 \geq 0$ durch die beiden Achsen begrenzt. Ein Produktionsplan (x_1, x_2), der alle Nebenbedingungen erfüllt (also realisierbar ist) wird eine *zulässige*

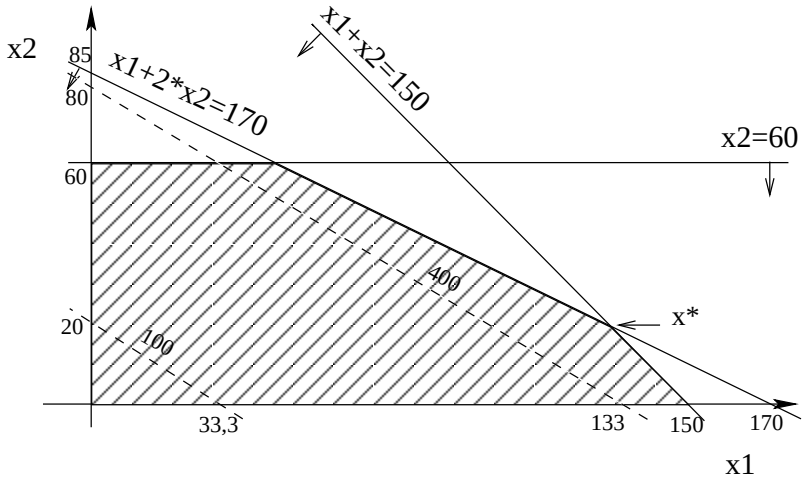


Abbildung 2.1: Der zulässige Bereich für das Beispiel aus der Produktionsplanung zusammen mit den Niveaulinien $L_=(100)$ und $L_=(400)$.

Lösung genannt. Abbildung 2.1 zeigt den zulässigen Bereich des Problems in Abhängigkeit von x_1 und x_2.
Um das lineare Programm optimal zu lösen, reicht es nicht, einfach einen Punkt aus dem zulässigen Bereich zu wählen. Wir sind vielmehr daran interessiert, den Punkt zu finden, der den Gewinn für die Firma maximiert. Dazu geht man folgendermaßen vor.
Man überlegt sich, welche Punkte zu einem Gewinn von z.B. 100 € führen, d.h. welche Punkte die Gleichung

$$\text{Gewinn} = 3x_1 + 5x_2 = 100$$

erfüllen. Auch das ist eine Gerade, die man äquivalent als $x_2 = 20 - \frac{3}{5}x_1$ schreiben kann. Man bezeichnet sie als *Niveauline* $L_=(100)$. Zeichnet man sie ein, sieht man, dass es Punkte gibt, die gleichzeitig auf dieser Geraden und im zulässigen Bereich liegen: Ein Gewinn von 100 € ist also erzielbar. Die Optimallösung ist das aber sicher nicht. Betrachten wir analog die Produktionspläne, die zu einem Gewinn von 400 € führen. Hierfür erhalten wir eine andere Niveaulinie, nämlich die Niveaulinie $L_=(400)$ zum Niveau 400. Auch das ist eine Gerade $3x_1 + 5x_2 = 400$ oder, äquivalent, $x_2 = 80 - \frac{3}{5}x_1$, und auch diese enthält zulässige Lösungen. Man beobachtet, dass die beiden Niveaulinien parallel sind: Eine Erhöhung des Zielfunktionswertes führt zu einer parallelen Verschiebung der Niveaulinie. Welcher Gewinn kann nun maximal erzielt werden und wie sieht der bestmögliche Produktionsplan aus?

Dazu verschiebt man die Niveaulinie so lange, bis sie gerade noch Punkte aus dem zulässigen Bereich enthält; in unserem Fall also bis zu einem Gewinn $z^* = 490$. Der einzige Punkt, der dann auf der Niveaulinie $L_=(490)$ und im zulässigen Bereich liegt, ist der Punkt mit den Koordinaten ${x_1}^* = 130$ und ${x_2}^* = 20$. Das sind also die optimalen Produktionsmengen; sie führen zu einem Gewinn von 490 €.

Das zeichnerische Vorgehen lässt sich für lineare Programme mit zwei Variablen ausführen. Wir fassen es nochmal zusammen:

1. Zeichne den zulässigen Bereich F.

2. Zeichne die Zielfunktion für einen festen Gewinn z durch ihre *Niveaulinie* $L_=(z)$

3. Verschiebe $L_=(z)$, bis z maximal und $L(z) \cap \mathcal{F} \neq \emptyset$. Optimal sind dann alle Punkte in der so erhaltenen Schnittmenge.

In unserem Fall haben wir eine Lösung erhalten, die an einer Ecke des zulässigen Bereiches liegt. Es bedeutet also, dass zwei der Ungleichungen mit Gleichheit erfüllt sind; in unserem Fall sind die Maschinen A und B voll ausgelastet.

Grundlegende Ergebnisse der linearen Programmierung

Wir fassen hier die wesentlichen Ergebnisse für die Lösung linearer Programme zusammen. Dabei liegt ein *lineares Programm* vor, wenn

- eine lineare Funktion
- über einem Polyeder

minimiert oder maximiert werden soll. Man kann lineare Programm immer mit Hilfe von linearen Nebenbedingungen und einer linearen Zielfunktion wie oben im Beispiel formulieren. Dabei dürfen die Nebenbedingungen Ungleichungen (z.B. $x_1 + x_2 \leq 100$ oder $2x_1 - x_2 + x_5 \geq 10$) oder Gleichungen (z.B. $x_1 + 4x_2 - x_3 = 25$) sein.

Für alle lineare Programme gilt der folgende Satz:

Satz 2.1 (Hauptsatz der linearen Programmierung) *Wenn es überhaupt eine optimale Lösung gibt, dann gibt es auch eine optimale Lösung, die an einer Ecke des zulässigen Polyeders liegt.*

Viele der Algorithmen, die lineare Programme lösen, basieren auf dieser Tatsache. Die wichtigsten Klassen von Verfahren sind

1. Simplex-Verfahren
2. Innere Punkte Verfahren
3. Parametrische Suche

Für Algorithmen ist es im allgemeinen wichtig, dass ihre Laufzeit auch bei großen Problemen nicht zu stark wächst. Man unterscheidet zwischen *polynomiellen Verfahren*, bei denen die Laufzeit in Abhängigkeit von der Größe des linearen Programms durch ein Polynom beschränkt ist, und *exponentiellen Verfahren*, bei denen das nicht der Fall ist. Obwohl das auf Dantzig (1947) zurückgehende Simplex-Verfahren bei praktischen Problemen meist sehr schnell ist, kann es im schlimmsten Fall eine exponentielle Laufzeit aufweisen. Es war lange nicht klar, ob es ein polynomielles Verfahren für die Lösung linearer Programme gibt, bis Khachian 1979 mit seiner Ellipsoid-Methode ein erstes polynomielles Verfahren präsentierte. Heutzutage werden solche Innere-Punkt-Verfahren für große lineare Programme standardmäßig verwendet. Durch

Verfahren der parametrischen Suche konnte Megiddo später dann sogar nachweisen, dass lineare Programmierung in fester Dimension in linearer Zeit lösbar ist.

Um lineare Programme zu lösen, kann man verschiedene Softwarepakete benutzen, wie z.B. Xpress oder CPLEX oder das unter der Gnu-Lizenz stehende glpk. Auch Excel enthält einen Solver.

2.2 Ganzzahlige Programmierung

Beispiel zur Produktionsplanung mit unteilbaren Produkten

Wir betrachten ein ähnliches Beispiel wie eben: Eine Firma stellt zwei Produkte P1 und P2 her. Der Verkaufserlös von P1 beträgt 3 €, der von P2 beträgt 5 €. Wiederum gehen wir davon aus, dass alle produzierten Einheiten auch verkauft werden können. Zur Herstellung der Produkte sind auch in diesem Beispiel drei Maschinen **A**, **B**, **C** nötig, mit den folgenden Bearbeitungszeiten und Kapazitäten:

	Zeit für P1	Zeit für P2	Kapazität der Maschine
Maschine A	1	2	3.4
Maschine B	1	1	3.0
Maschine C		3	3.6

Erneut stellt sich die Frage, wie viel die Firma von welchem Produkt produzieren soll, um ihren Gewinn zu maximieren. Dabei können von beiden Produkten P1 und P2 nur ganze Einheiten produziert und verkauft werden.

Das Modell für dieses Problem ist analog zu dem ersten Beispiel in diesem Abschnitt schnell angegeben.

Variablen:

$$\begin{aligned} x_1 &= \text{Menge an P1} \\ x_2 &= \text{Menge an P2} \end{aligned}$$

Zielfunktion:

$$\text{Gewinn} = f(x_1, x_2) = 3x_1 + 5x_2.$$

Nebenbedingungen:

$$\begin{aligned} x_1 + 2x_2 &\leq 3.4 \\ x_1 + x_2 &\leq 3.0 \\ 3x_2 &\leq 3.6 \\ x_1, x_2 &\in \mathbb{N}. \end{aligned}$$

Zusätzlich sollen die Variablen nur ganzzahlige Werte annehmen.

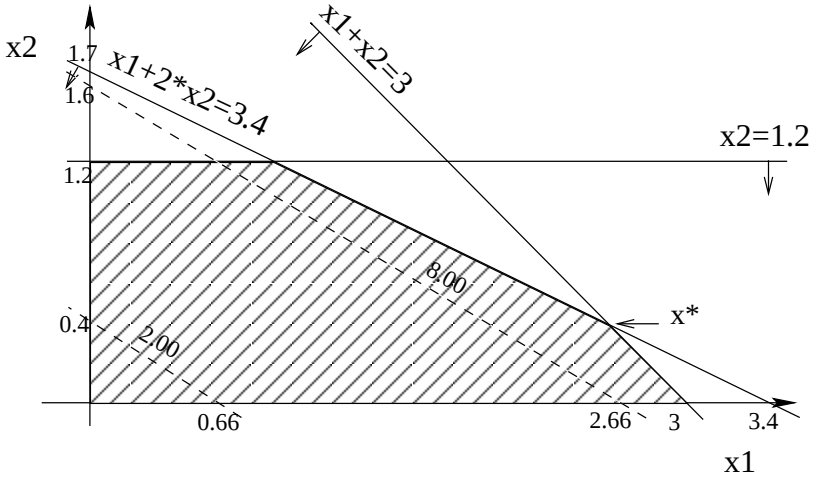


Abbildung 2.2: Der durch die Nebenbedingungen begrenzte Bereich für das Beispiel aus der ganzzahligen Produktionsplanung mit den beiden Niveaulinien $L_{=}(2)$ und $L_{=}(8)$.

Der zulässige Bereich des Problems wird durch die fünf Nebenbedingungen begrenzt. Dieses Polyeder ist in Abbildung 2.2 dargestellt.
Wieder untersuchen wir Niveaulinien, um das Problem zu lösen. Die Niveaulinie zu $z = 2$ ist $3x_1 + 5x_2 = 2$ oder $x_2 = 0,4 - \frac{3}{5}x_1$. Zum Niveau $z = 8$ erhält man $3x_1 + 5x_2 = 8$ bzw. $x_2 = 1,6 - \frac{3}{5}x_1$ als Niveaulinie. Mit dem zeichnerischen Verfahren würden sich für dieses Beispiel als optimale Lösung die Produktionsmengen $x_1^* = 2.6$ und $x_2{}^* = 0.4$ ergeben. Diese sind allerdings nicht ganzzahlig! Somit sind sie keine zulässige Lösung unseres Problems.

Wie findet man nun aber den besten Produktionsplan, der nur ganzzahlige Produktionsmengen enthält?

Dazu betrachten wir noch einmal den eben untersuchten Bereich und stellen fest, dass das durch die fünf Nebenbedingungen entstehende Polyeder jede Menge unzulässige Punkte enthält! Genauer sind nur Punkte mit ganzzahligen Koordinaten zulässig, die innerhalb des Polyeders liegen. Diese zulässige Menge für das ganzzahlige Programm ist in Abbildung 2.3 dargestellt.
In unserem Beispiel verschieben wir die Zielfunktion, bis sie den letzten ganzzahligen Punkt enthält. Es ergeben sich die optimalen Produktionsmengen von $x_1^* = 3$ und $x_2^* = 0$.

Bevor wir kurz auf Lösungsverfahren eingehen, soll hier aufgeführt werden, wozu ganzzahlige Variablen nützlich sind. Ganzzahlige Variablen braucht man wie eben in Fällen, in denen Produkte oder Ressourcen unteilbar sind, etwa ganze Lastwagen, ganze Passagiere, oder wie in Kapitel 1 ganze Flaschen oder ganze Zaunstücke.
Ein bedeutender Spezialfall von ganzzahligen Variablen sind *Boolesche Variablen*, die nur die Werte 0 und 1 annehmen dürfen. Diese sind insbesondere nötig, um ja/nein Entscheidungen oder um Beziehungen zwischen verschiedenen Ereignissen zu modellieren. Fragestellungen, die man damit formulieren kann, sind z.B.:

- Soll eine Anlage eingerichtet werden oder nicht?

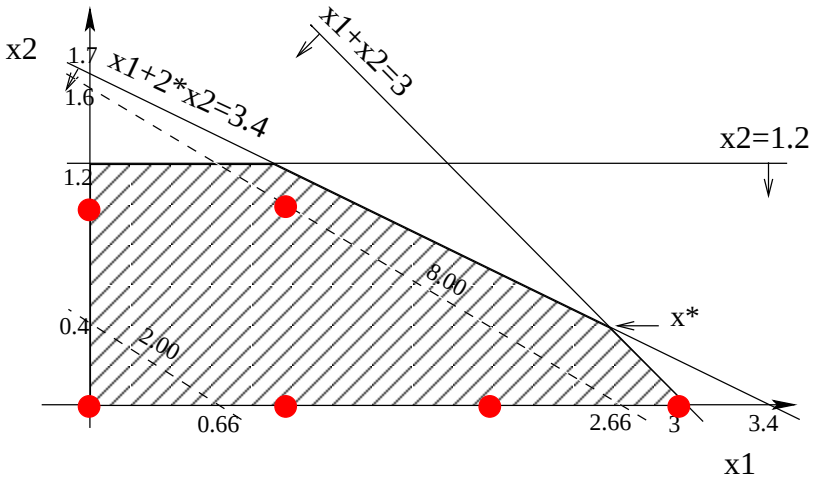


Abbildung 2.3: Die dicken Punkte markieren die Produktionsmengen, die für das ganzzahlige Problem zulässig sind. Der zulässige Bereich besteht also aus genau sechs Lösungen.

- Soll eine Fabrik heruntergefahren werden oder nicht?
- Soll ein Antrag gefördert werden oder nicht?
- Wenn Anlage 1 eingerichtet wird, muss Lager B verwendet werden.
- Wenn ein Produkt von A bezogen wird, fallen Fixkosten in der Höhe von C_{fix} an.
- Wenn ein Lager gebaut wird, kann man von dort Waren liefern lassen.

In Kapitel 1 haben wir bei dem Rucksackproblem und bei dem Zuordnungsproblem schon zwei Beispiele kennen gelernt, in denen boolesche Variablen verwendet werden.

Grundlegende Ergebnisse der ganzzahligen Programmierung

Ein lineares **ganzzahliges** Programm liegt vor, wenn

- eine lineare Funktion
- über einem Polyeder

minimiert oder maximiert werden soll und

- die Variablen nur ganzzahlige Werte annehmen dürfen.

Um ganzzahlige Programme zu lösen, hilft es im allgemeinen nicht, das zugehörige lineare Programm zu lösen und diese Lösung dann auf eine ganzzahlige Lösung zu runden. Es gibt viele Beispiele, in denen die gerundete Lösung nicht optimal ist und — schlimmer noch ! — Beispiele, in denen die gerundete Lösung noch nicht einmal zulässig ist.

Immerhin erhält man durch die Lösung des zugehörigen linearen Programms eine Abschätzung über die Lösung des ganzzahligen Problems: Die gesuchte Lösung des ganzzahligen Problems kann niemals besser sein als die Lösung des zugehörigen linearen Programmes.

Lösungstechnisch ist die ganzzahlige Programmierung viel schwieriger als die lineare Programmierung. Dies spiegelt sich auch bei den Verfahren wieder, die man zur Lösung von ganzzahligen Programmen verwendet. Im Gegensatz zur linearen Programmierung gibt es für die meisten ganzzahligen oder booleschen Probleme keine polynomiellen Verfahren, und es wird von den meisten Wissenschaftlern als unwahrscheinlich angesehen, dass es möglich ist, solche zu entwerfen. In diesem Zusammenhang fällt oft der Begriff *NP-vollständig*.

Exkurs: Was bedeutet: " NP-vollständig" ?

Es gibt eine Klasse von Problemen, die man als NP-vollständig bezeichnet. Derzeit sind keine Lösungsverfahren mit polynomieller Laufzeit für diese Probleme bekannt. Man weiß aber das folgende:

- *Wenn man für ein NP-vollständiges Problem einen polynomiellen Algorithmus finden würde, dann könnte man alle NP-vollständigen Probleme polynomiell lösen.*
- *Es gibt Techniken, mit denen man zeigen kann, dass ein neues Problem NP-vollständig ist. Man zeigt, dass das neue Problem in einem gewissen (wohldefinierten) Sinn schwerer ist als ein schon bekanntes NP-vollständiges Problem.*
- *Natürlich gibt es auch viele Probleme, für die man polynomielle Verfahren kennt. Die Klasse der polynomiell lösbaren Probleme wird mit P bezeichnet.*

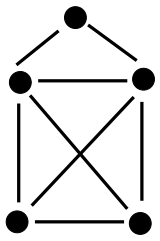
Die spannende Frage ist nun, ob P=NP oder P≠NP richtig ist. Im ersten Fall wären alle NP-vollständigen Probleme polynomiell lösbar, im zweiten Fall nicht. Die meisten Wissenschaftler vermuten $P \neq NP$, das heißt, sie nehmen an, dass man die NP-schweren Probleme nicht in polynomieller Zeit lösen kann. Geklärt ist die Frage $P = NP$? aber noch nicht.

Was für Konsequenzen hat es nun, wenn ein praktisches Problem zu dieser Klasse von besonders schwierigen Problemen gehört? Weiß man, dass das zu lösende Problem NP-schwer ist, und ist es sehr groß, so lässt es sich oft nicht exakt lösen. In diesem Fall ist es besser, so genannte *Heuristiken* zu verwenden, die nicht die optimale Lösung finden, aber gute Näherungslösungen liefern. Entsprechend teilt man die Verfahren für ganzzahlige Probleme auch in diese beiden Klassen ein: Exakte und heuristische Verfahren.
Algorithmen für exakte Verfahren sind Branch & Bound Verfahren und Cutting Plane Techniken. Bei beiden Verfahren wird jeweils eine Folge von linearen Programmen gelöst. Typische Heuristiken sind Greedy-Verfahren, Verbesserungsstrategien, IP-basierte Heuristiken oder Metaheuristiken (Simulated Annealing, genetische Verfahren, Ameisenverfahren ...).

Kapitel 3

Einführung in die Graphentheorie und kürzeste Wege

3.1 Was sind Graphen?



Haus des Nikolaus

Ein Netzwerk (oder ein Graph) ist — informell ausgedrückt — ein Gebilde aus Punkten und Strecken. Die Punkte nennt man *Knoten* oder *Ecken* des Graphen, die Strecken verlaufen immer zwischen zwei solchen Knoten und werden als *Kanten* bezeichnet. Dabei ist es egal, ob die Kanten gerade oder krumm gezeichnet werden. Ein typischer Graph ist z.B. das „Haus des Nikolaus“ . Als Graph besteht es aus fünf Knoten und acht Kanten. Graphen sind in der Theorie interessant, treten aber vor allem in vielen Anwendungen auf und führen zu unterschiedlichen Fragestellungen. Drei Beispiele sollen das verdeutlichen:

Beispiel 1: In den Sozialwissenschaften werden Gruppenstrukturen mittels Graphen analysiert. Die einzelnen Gruppenmitglieder sind die Knoten; man zeichnet eine Kante zwischen zwei Gruppenmitgliedern A und B, wenn sich die beiden gut verstehen. Die Anzahl der Kanten, die bei einem Gruppenmitglied ankommen (mathematisch der *Knotengrad*) ist dann ein Maß für seine Beliebtheit innerhalb der Gruppe. Gibt es Untergruppen, in denen jeder mit jedem verbunden ist, so spricht man von einer *Clique*. Das Auffinden von Cliquen in einem (großen) Graphen ist ein spannendes Problem aus der Mathematik und Informatik.

Beispiel 2: In Navigationssystemen ist das Straßennetz als Graph modelliert. Kreuzungen und Ziele sind die Knoten, die verbindenden Straßen sind die Kanten. Hier sind zusätzlich Attribute angefügt, die die Länge bzw. den Zeitbedarf zum Befahren einer Kante angeben. Ein Weg von einem Punkt X zu einem anderen Punkt Y zu berechnen, ist als *kürzestes Wege Problem* bekannt. Hierfür gibt es schnelle Verfahren, die erlauben, dass das Navigationsgerät in wenigen Momenten einen Weg ausgibt und den auch automatisch verbessert, wenn man z.B. falsch abbiegt. Ein einfaches Verfahren zur Bestimmung der kürzesten Wege wird in Abschnitt 3.4 beschrieben.

Beispiel 3: Aber auch in der Technik spielen Graphen eine Rolle. So lassen sich beispielsweise Platinen als Graphen auffassen. Die Lötstellen und die gewünschten Kreuzungspunkte entsprechen den Knoten und die sie verbindenden Leitungen entsprechen den Kanten.

Beim Aufbau einer Platine sollte vermieden werden, dass sich Leitungen (innerhalb der selben Lage) überkreuzen. Einen Graphen, der so gezeichnet ist, dass sich keine zwei Kanten schneiden, nennt man *planar*. A priori ist bei vielen Graphen nicht klar, ob man sie planar zeichnen kann oder nicht, bzw. mit wie vielen Überschneidungen man auskommt.

Auch in der Biologie treten häufig Graphen auf. Das Lymphsystem des menschlichen Körpers ist ein Graph, bestehend aus den Lymphknoten, die durch die Lymphgefäße miteinander verbunden sind. In der Vererbungslehre sind Stammbäume wichtig, die ebenfalls als Graphen aufgefasst werden können: Die einzelnen Individuen sind die Knoten, die Eltern-Kind Beziehungen werden als Kanten eingetragen. In den folgenden Abschnitten werden Graphen insbesondere verwendet, um die Stoffwechselvorgänge im Körper zu modellieren.

An den Beispielen wird deutlich, dass es verschiedene Typen von Graphen geben kann. So haben manche Graphen *gerichtete Kanten*, bei anderen treten *ungerichtete Kanten* auf. Eine gerichtete Kante deutet an, dass die beiden sie verbindenden Knoten nicht die gleiche Beziehung zueinander haben. In den Sozialwissenschaften kann man damit modellieren, dass zwar Person A Person B mag, aber nicht umgekehrt. Im zweiten Beispiel werden gerichtete Kanten gebraucht, wenn Einbahnstraßen vorhanden sind. Bei dem Platinen-Beispiel liegen nur ungerichtete Graphen vor.

Mathematisch definiert man Graphen wie folgt:
Ein *gerichteter Graph* $G = (V, E)$ ist ein Tupel, bestehend aus einer nicht-leeren Menge V von *Knoten* und einer Menge $E \subseteq \{(i, j) : i, j \in V\}$ von gerichteten Kanten.
Ein *ungerichteter Graph* $G = (V, E)$ ist ein Tupel, bestehend aus einer nicht-leeren Menge V von *Knoten* und einer Menge $E \subseteq \{\{i, j\} : i, j \in V\}$ von ungerichteten Kanten.

In beiden Definitionen ist sichergestellt, dass die Kanten zwischen jeweils zwei Knoten verlaufen; einmal als geordnete Tupel (i, j) und einmal als ungeordnete Tupel (also Mengen aus jeweils zwei Elementen) $\{i, j\}$. Die geordneten Tupel führen zu gerichteten Kanten, die als Pfeile gezeichnet werden, während die ungeordneten Tupel ungeordnete Kanten repräsentieren. Es ist zu beachten, dass die *Mengen* $\{i, j\}$ und $\{j, i\}$ gleich sind, die *Tupel* (i, j) und (j, i) aber unterschiedlich (falls $i \neq j$).
Die folgende Abbildung zeigt zwei Graphen, beide mit den Knoten $V = \{1, 2, 3, 4, 5\}$.

- Der gerichtete Graph G_1 (links) enthält die Kanten

$$a = (1, 2), b = (2, 1), c = (3, 2), d = (3, 5), e = (5, 4), f = (4, 3).$$

- Die Kantenmenge des ungerichteten Graphen G_2 (rechts) besteht aus

$$a = \{1, 3\}, b = \{1, 4\}, c = \{2, 3\}, d = \{3, 4\}, e = \{3, 5\}.$$

Ist $e = (i, j)$ oder $e = \{i, j\}$ eine Kante, die zwischen den Knoten i und j verläuft, so sagt man, dass e mit i und j *inzidiert*. In manchen Anwendungen sind mehrere Kanten zwischen dem gleichen Knotenpaar erlaubt; das verkompliziert die Definition, da man eine Kante dann nicht mehr eindeutig durch die Angabe ihres Start- und ihres Endknotens beschreiben kann. In diesem Text soll auf Graphen mit solchen *Mehrfachkanten* verzichtet werden.

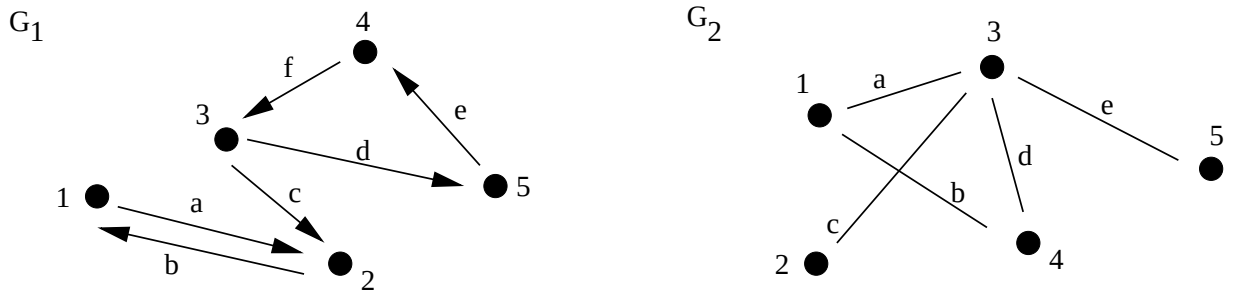


Abbildung 3.1: Ein gerichteter und ein ungerichteter Graph

Ein wichtiger Begriff in Graphen ist ein *Weg*. Anschaulich ist ein Weg in einem Graphen ein durchgehender Kantenzug zwischen zwei Knoten i und j. Formal definiert man einen Weg zwischen den Knoten i und j als eine Folge $P = (e_1, e_2, \ldots, e_K)$ von Kanten, die das folgende erfüllt:

- die erste Kante e_1 inzidiert mit i ,
- die letzte Kante e_K inzidiert mit j,
- alle Endknoten einer Kanten e_k im Weg (außer dem ersten Knoten i und dem letzten Knoten j) inzidieren mit der vorhergehenden Kante e_{k-1} oder mit der darauf folgenden Kante e_{k+1}.

In dem ungerichteten Graphen aus Abbildung 3.1 beschreibt die Kantenfolge (a, b, d, e) beispielsweise einen Weg, die Kantenfolge (e, c, d) aber nicht.
In gerichteten Graphen muss man dabei noch unterscheiden, ob die Kanten $e \in P$ *vorwärts* oder *rückwärts* gerichtet sind. In dem Weg (a, c, d) in dem gerichteten Graphen aus Abbildung 3.1 sind die Kanten a und d vorwärts gerichtet und die Kante c rückwärts gerichtet. Sind in einem Weg in einem gerichteten Graphen alle Kanten vorwärts gerichtet, so nennt man den Weg *gerichteten* Weg. Ein Beispiel dafür ist der Weg (f, c, b).
Basierend auf der Definition von Wegen, kann man die folgenden beiden wichtigen Begriffe einführen:

- Einen Weg, der von einem Knoten i wieder zurück nach i verläuft, nennt man *Kreis*. Ein Beispiel für einen Kreis in dem gerichteten Graphen G_1 ist die Kantenfolge (d, e, f), ein Kreis in G_2 wird z.B. durch (c, d, b, a, c) beschrieben.

- Ein Graph heißt *zusammenhängend* wenn es zu je zwei Knoten $i, j \in V$ einen Weg von i nach j gibt. Die beiden Graphen G_1 und G_2 aus Abbildung 3.1 sind also beide zusammenhängend.

  Für gerichtete Graphen kann man den Zusammenhangsbegriff noch verstärken: Ein gerichteter Graph heißt *stark zusammenhängend* wenn es für je zwei Knoten $i, j \in V$ einen gerichteten Weg von i nach j gibt. Der Graph G_1 ist also nicht stark zusammenhängend, da es z.B. keinen Weg von 2 nach 3 gibt.

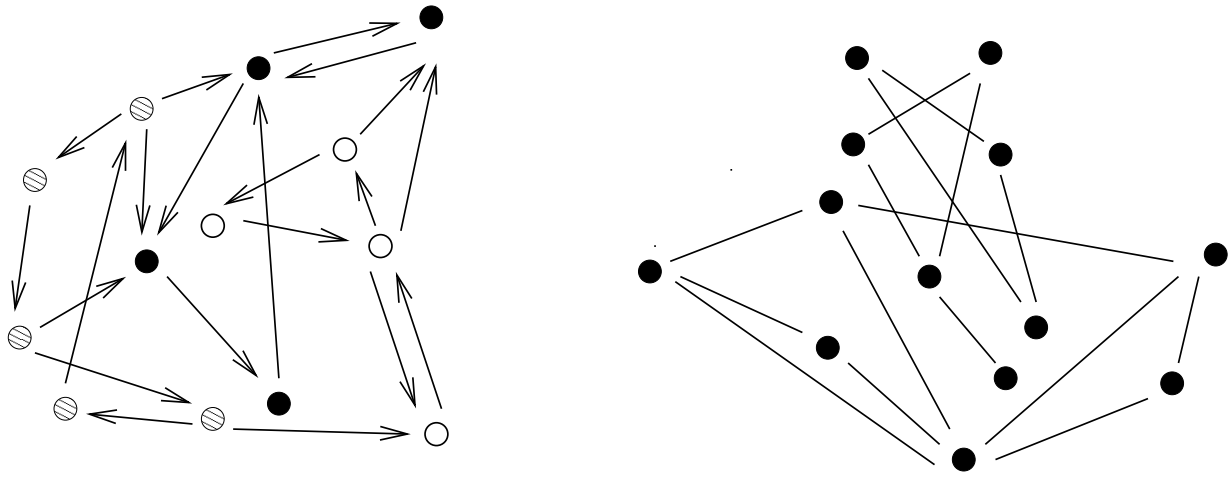


Abbildung 3.2: Ein gerichteter Graph mit einer Zusammenhangskomponente, der in drei starke Zusammenhangskomponenten zerfällt und ein ungerichteter Graph mit drei Zusammenhangskomponenten

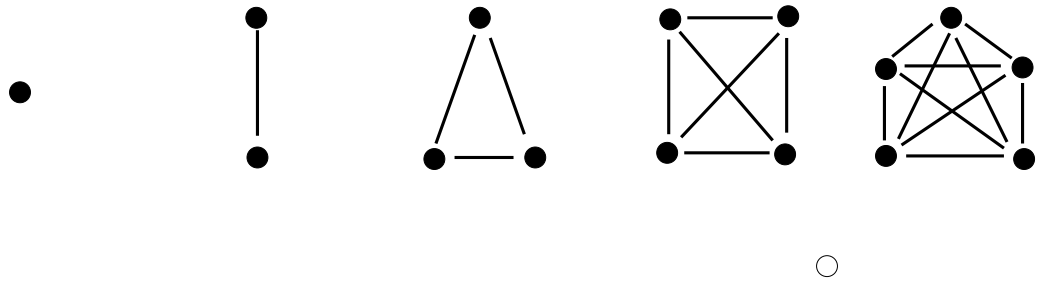


Abbildung 3.3: Die vollständigen Graphen K_1, K_2, K_3, K_4 und K_5 mit 1,2,3,4,5 Knoten.

Ist ein Graph nicht (stark) zusammenhängend, so kann man ihn in (stark) zusammenhängende Komponenten zerlegen. Der Graph G_1 zerfällt entsprechend in die beiden starken Zusammenhangskomponenten $\{1, 2\}$ und $\{3, 4, 5\}$. In Abbildung 3.2 ist ein ungerichteter Graph und ein gerichteter Graph dargestellt. Der ungerichtete Graph zerfällt in drei Zusammenhangskomponenten. Der gerichtete Graph besteht aus einer Zusammenhangskomponente, zerfällt aber in drei starke Zusammenhangskomponenten (gekennzeichnet durch die unterschiedliche Markierung der Knoten).

3.2 Besondere Graphen

In diesem Abschnitt soll auf besondere Graphen eingegangen werden, die häufig verwendet werden. Zu jedem der Graphen werden ein paar typische Fragestellungen und Anwendungen genannt.

Wir beginnen mit *vollständigen Graphen*: Ein Graph heißt *vollständig*, wenn alle möglichen Kanten vorhanden sind. Bei einem ungerichteten Graphen mit Knotenmenge V heißt das also,

dass für jedes Paar von Knoten $i, j \in V$ die zugehörige Kante $\{i, j\}$, die die beiden Knoten verbindet, in der Kantenmenge E enthalten ist. Der vollständige Graph mit n Knoten wird mit K_n bezeichnet. Abbildung 3.3 zeigt die vollständigen Graphen K_1, K_2, K_3, K_4 und K_5.
Vollständige Graphen haben u.a. in den Sozialwissenschaften eine wichtige Bedeutung: Stellt man sich wie in Beispiel 1 einen Graphen vor, in dem die Knoten Individuen repräsentieren und die Kanten eine enge Beziehung zwischen den Individuen andeuten, so kann man einen vollständiger Graphen als Clique interpretieren. Ein Teilgraph eines Graphen, der vollständig ist, wird in der Graphentheorie daher als *Clique* bezeichnet. Das Auffinden von Cliquen ist aber nicht nur in den Sozialwissenschaften wichtig, sondern auch in vielen Optimierungsproblemen von Bedeutung, da Cliquen oft genutzt werden können, um z.B. Färbungsprobleme effizienter zu lösen.

Eine weitere wichtige Klasse von Graphen sind *bipartite* Graphen. Die Knotenmenge V eines *bipartiten* Graphen kann in zwei disjunkte Mengen A und B partioniert werden, so dass innerhalb der beiden Mengen keine Kanten verlaufen. Das heißt, die Kanten verbinden immer jeweils ein Element aus A mit einem Element aus B. Man zeichnet die beiden Mengen dann meistens so wie im linken Teil von Abbildung 3.4: die Elemente der Menge A auf der linken Seite übereinander und die Elemente der Menge B rechts, so dass man gleich sieht, dass die Kanten nur zwischen den Mengen verlaufen. Ein anderer bipartiter Graph ist im rechten Teil von Abbildung 3.4 dargestellt, allerdings nicht in der Standard-Form. Die Menge A enthält die weißen Knoten, die Menge B die schwarzen Knoten. Man sieht, dass keine zwei weißen und keine zwei schwarzen Knoten miteinander verbunden sind.

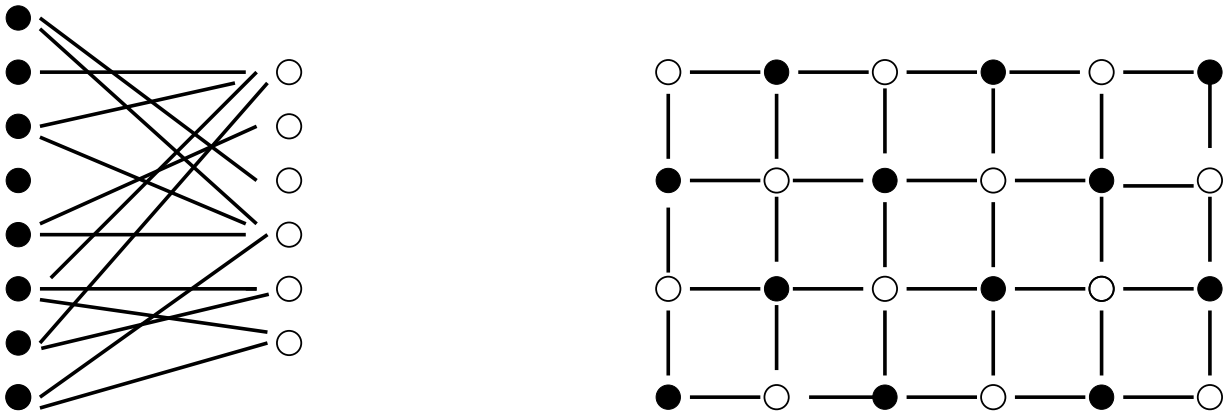


Abbildung 3.4: Zwei bipartite Graphen

Bipartite Graphen sind bei Zuordnungsproblemen wichtig. Man kann sich z.B. vorstellen, dass in der Menge A Mitarbeiter stehen und in der Menge B die auszuführenden Aufgaben. Die Kanten deuten an, ob ein Mitarbeiter für eine Aufgabe geeignet ist. Gesucht ist eine Zuordnung von Mitarbeitern zu Aufgaben, so dass alle Aufgaben erledigt werden. Ähnlich kann man Aufgaben Maschinen zuordnen oder in der Stundenplanerstellung Seminare und Vorlesungen auf Räume verteilen. Ein oft zitiertes Beispiel für einen bipartiten Graph ist der "Heiratsgraph" in dem die Menge A heiratsfähige Frauen enthält und die Menge B heiratsfähige Männer. (Sind gleichgeschlechtliche Ehen erlaubt, ist dieser Graph nicht mehr bipartit.) Zuordnungsprobleme haben wir in Kapitel 1 bereits kennengelernt und auf Seite 8 als ganzzahliges Programm formuliert.

Als letzte Klasse von besonderen Graphen sollen noch Bäume eingeführt werden. Ein *Baum* ist ein zusammenhängender, kreisfreier, ungerichteter Graph. Verschiedene Bäume sind in Abbildung 3.5 dargestellt. Dabei sind die beiden Bäume links und in der Mitte wiederum besondere Bäume: Ganz links ist ein so genannter *allgemeiner Stern* dargestellt, bei dem von einem zentralen Knoten aus verschiedene Wege abgehen, in der Mitte ein einfacher *Weg*.

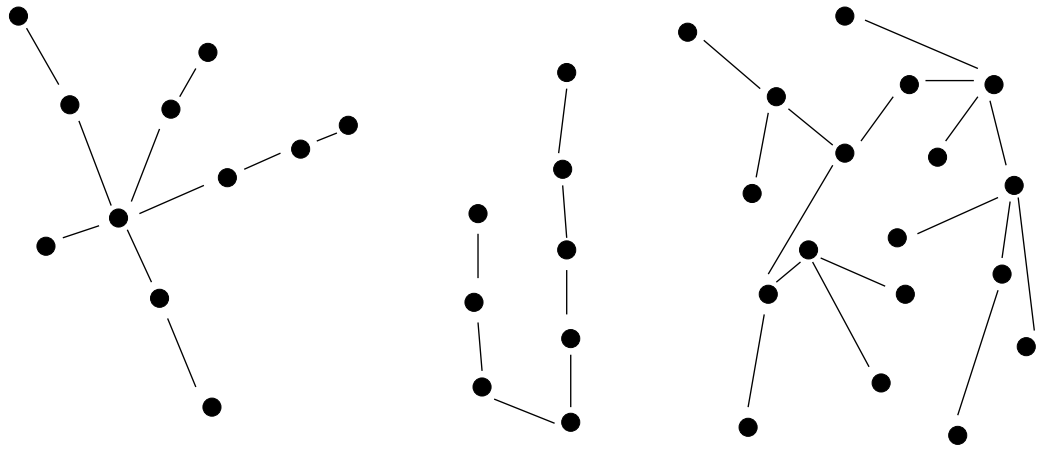


Abbildung 3.5: Drei Bäume

Die Anzahl der Kanten in einem Baum ist immer die Anzahl der Knoten - 1. Dieses Kriterium kann man auch nutzen, um Bäume zu charakterisieren: So lässt sich ein Baum mit n Knoten äquivalent definieren als

- ein kreisfreier Graph mit $n-1$ Kanten,
- ein zusammenhängender Graph mit $n-1$ Kanten,
- ein kreisfreier Graph mit größtmöglicher Anzahl von Kanten,
- ein zusammenhängender Graph mit kleinstmöglicher Anzahl von Kanten.

Zeichnet man zu einem Baum eine neue Kante hinzu, so entsteht genau ein Kreis. Diese Eigenschaft ist für viele Überlegungen nützlich. Bäume kommen zur Anwendung, wenn man die Knoten eines Graphen kostengünstig verbinden möchte. Baumstrukturen sind aber auch für viele andere Anwendungen wichtig, u.a. bei der Berechnung von periodischen Fahrplänen und in der Algorithmik.

3.3 Einige typische Fragestellungen in Graphen

Abschließend seien noch ein paar typische Fragestellungen aufgeführt, die man mit Hilfe von Graphen bearbeiten kann.

Hamiltonscher Pfad: Finde einen Weg, der jeden Knoten des Graphen genau einmal besucht.

Eulerscher Pfad: Finde einen Weg, der jede Kante des Graphen genau einmal besucht.

Clique: Finde die größte Clique in einem Graphen.

Färbungsproblem: Färbe die Knoten eines Graphen mit möglichst wenig Farben so, dass benachbarte Knoten immer unterschiedliche Farben haben.

Planarität: Finde eine planare Repräsentation eines Graphen oder erkenne, dass es keine gibt.

Weitere Fragestellungen benötigen neben den Knoten und den Kanten eines Graphen für jede Kante e noch ein *Kantengewicht* d_e, das man als Länge oder Kosten der Kante interpretieren kann. So ein Graph nennt man dann einen *gewichteten* Graph. Die folgenden Fragestellungen sind typisch für Graphen mit Kantengewichten:

Minimaler spannender Baum: Verbinde alle Knoten eines Graphen mit möglichst kostengünstigen Kanten.

Traveling Salesman Problem: Finde in einem gewichteten Graphen eine Rundtour minimaler Länge, die jeden Knoten genau einmal besucht.

Kürzester Weg: Finde in einem Graph mit gewichteten Kanten einen möglichst kurzen Weg von einem ausgezeichneten Knoten A zu einem anderen ausgezeichneten Knoten B.

Bipartite Zuordnung: Ordne in einem bipartiten Graphen mit den Mengen A und B jedem Element aus A ein Element aus B zu, so dass die Summe der Gewichte auf den Zuordnungskanten möglichst groß ist.

Interessanterweise sind die Verfahren zum Lösen der eben genannten Probleme sehr unterschiedlich. Manche der Probleme lassen sich auch in großen Graphen schnell lösen. Dazu gehören die Probleme Eulerscher Pfad, Planarität, minimaler spannender Baum, kürzester Weg und bipartite Zuordnung. Die anderen der genannten Probleme sind deutlich schwieriger; bei großen Graphen stößt man mit den bekannten Verfahren schnell an die Grenzen der Rechenleistungen. Man beurteilt dabei die Rechenzeit eines Verfahrens in Abhängigkeit der Anzahl der Knoten (oder Kanten) des gegebenen Graphen. Wächst das Verfahren polynomiell in der Anzahl der Knoten, so ist es effizient und lässt sich meist auch noch auf große Probleme anwenden. Ist die Laufzeit nicht durch ein Polynom beschränkt, so spricht man von einem Verfahren mit exponentieller Laufzeit. Für die letztgenannten Probleme sind bisher nur Verfahren bekannt, die im schlimmsten Fall exponentielle Laufzeit aufweisen. Ob es für diese Probleme überhaupt effiziente Verfahren gibt, ist bisher nicht geklärt — die meisten Wissenschaftler gehen aber davon aus, dass dem nicht so ist.

3.4 Ein algorithmisches Beispiel: Kürzeste Wege in einem gewichteten Graph

In diesem Abschnitt beschäftigen wir uns mit der Frage, kürzeste Wege in einem Graphen zu bestimmen. Wir betrachten dazu den folgenden gerichteten Graphen als Beispiel:

In diesem Graph gibt es verschiedene Wege vom Knoten 2 zum Knoten 1:

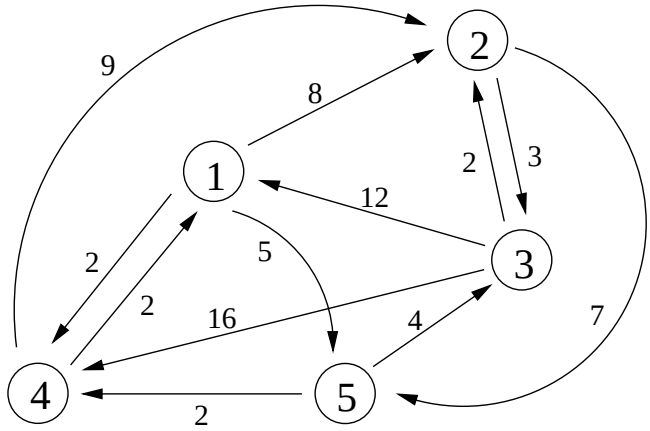


Abbildung 3.6: Der Beispielsgraph für das Verfahren von Floyd und Warshall

- Der Weg von Knoten 2 über Knoten 3 und dann zu Knoten 1 mit einer Länge von $3 + 12 = 15$,
- einen Weg von Knoten 2 über die Knoten 5,4 und 1 mit einer Länge von $7 + 2 + 2 = 11$
- und einen Weg von Knoten 2 über die Knoten 5,3,4 und 1 mit einer Länge von $7 + 4 + 16 + 2 = 29$.

Der kürzeste dieser möglichen Wege ist also der zweite über die Knoten 5,4 und 1. Da es (insbesondere in großen Netzwerken) mühsam ist, alle möglichen Wege zu finden und durchzuprobieren, wollen wir im folgenden ein Verfahren vorstellen, wie man das auf systematische Art und Weise machen kann.

Mit dem Verfahren, das nun vorgestellt werden soll, lassen sich alle kürzesten Wege in einem gegebenen Graphen berechnen. Es geht auf Floyd und Warshall zurück. Wir demonstrieren es zunächst an unserem Beispielsgraphen aus Abbildung 3.6.

Als erstes speichern wir die Entfernungen der vorhandenen Kanten in Form der hier dargestellten Matrix $D = (d_{ij}^0)$ ab.

von \ nach	1	2	3	4	5
1	0	8	–	2	5
2	–	0	3	–	7
3	12	2	0	16	–
4	2	9	–	0	–
5	–	–	4	2	0

Dabei bezeichnen wir den Eintrag in der Matrix aus der i.ten Zeile und der j.ten Spalte mit d_{ij}^0. Die hochgestellte Null deutet an, dass es sich um die gegebene Grundmatrix handelt, die die Länge der Kante von Knoten i nach Knoten j enthält. Gibt es keine Kante zwischen zwei Knoten, so deuten wir das mit einem $-$ an.

Um die Längen der kürzesten Wege zwischen verschiedenen Knotenpaaren zu finden, gehen wir iterativ vor. Die Idee ist, jeden Knoten einmal als so genannten *Umwegknoten* zu wählen und zu testen ob ein Weg über so einen Umwegknoten zu Ersparnissen führt.

Wir beginnen mit Knoten 1 als Umwegknoten und versuchen, die schon vorhanden Wege über den Knoten zu verbessern bzw. neue Wege zu erzeugen. Wir demonstrieren dieses Vorgehen an drei Beispielen:

- Wir vergleichen den bisherigen (aus einer Kante bestehenden) Weg $3 \to 4$ mit dem neuen Weg $3 \to 1 \to 4$. Der Weg $3 \to 1 \to 4$ existiert und hat die Länge $12 + 2 = 14$. Der beste bisher bekannte Weg hat die Länge 16, also ist der neue Weg kürzer als der bisherige Eintrag in der Matrix. Wir ersetzen daher die 16 in der Matrix durch eine 14.

- Nun vergleichen den besten bekannten "Weg" $3 \to 5$ (bisher gibt es hier noch keinen, daher ist das Zeichen $-$ in der Matrix eingetragen) mit einem möglichen Weg $3 \to 1 \to 4$ über den Knoten 1 als Umwegknoten. Der neue Weg $3 \to 1 \to 5$ existiert und hat die Länge $12+5 = 17$, das ist besser als der bisherige Eintrag in der Matrix, in dem überhaupt noch keine Verbindung zwischen den Knoten angegeben ist. Wir ersetzen daher den aktuellen Eintrag $-$ durch 17.

- Der Vergleich von $4 \to 2$ mit dem Weg $4 \to 1 \to 2$ zeigt, dass der Umweg über den Knoten 1 zu einer Länge von $2+8 = 10$ führt. Er ist somit länger als die ursprüngliche Länge des Weges $4 \to 2$. Die Länge der Strecke $4 \to 2$ bleibt daher 9.

- Der Vergleich von $2 \to 3$ mit dem Weg $2 \to 1 \to 3$ führt zu keinem Ergebnis, da bisher kein Weg von 2 nach 1 bekannt ist. Der Eintrag 3 in der Matrix bleibt folglich bestehen.

Geht man diese Überlegung für alle Knotenpaare durch, so erhält man die folgende Matrix $D^1 = (d^1_{ij})$. Die Einträge, die sich im Vergleich zur Ausgangsmatrix D^0 verändert haben, sind unterstrichen dargestellt.

$$D^1 = (d^1_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & - & 2 & 5 \\ 2 & - & 0 & 3 & - & 7 \\ 3 & 12 & 2 & 0 & \underline{14} & \underline{17} \\ 4 & 2 & 9 & - & 0 & \underline{7} \\ 5 & - & - & 4 & 2 & 0 \end{array}$$

Im nächsten Schritt wird nun der Knoten 2 als Umwegknoten gewählt und wieder werden alle Einträge der Matrix untersucht. (Man kann übrigens ein bisschen Arbeit sparen, weil man die Einträge in der zweiten Zeile und die in der zweiten Spalte nicht betrachten muss.) Die Einträge, in denen der Umweg über den Knoten 2 eine Verbesserung ergibt, sind in der sich ergebenden Matrix wieder unterstrichen.

$$D^2 = (d^2_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & \underline{11} & 2 & 5 \\ 2 & - & 0 & 3 & - & 7 \\ 3 & 12 & 2 & 0 & 14 & \underline{9} \\ 4 & 2 & 9 & \underline{12} & 0 & 7 \\ 5 & - & - & 4 & 2 & 0 \end{array}$$

Analog untersucht man weiter die Knoten 3 und 4 als Umwegknoten und erhält die folgenden Matrizen, in denen wieder die Einträge unterstrichen sind, bei denen der Weg über den jeweiligen Umwegknoten zu einer Verbesserung geführt hat.

$$D^3 = (d^3_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & 11 & 2 & 5 \\ 2 & \underline{15} & 0 & 3 & \underline{17} & 7 \\ 3 & 12 & 2 & 0 & 14 & 9 \\ 4 & 2 & 9 & 12 & 0 & 7 \\ 5 & \underline{16} & \underline{6} & 4 & 2 & 0 \end{array}, \quad D^4 = (d^4_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & 11 & 2 & 5 \\ 2 & 15 & 0 & 3 & 17 & 7 \\ 3 & 12 & 2 & 0 & 14 & 9 \\ 4 & 2 & 9 & 12 & 0 & 7 \\ 5 & \underline{4} & 6 & 4 & 2 & 0 \end{array}$$

Nimmt man zum Schluss noch den Knoten 5 in die Menge der erlaubten Umwegknoten, so erhält man eine Matrix, in der die Längen der kürzesten Wege der Knoten untereinander abgelesen werden können. Die Erstellung dieser letzten Matrix D^5 mit dem Knoten 5 als Umwegknoten führt zu vier Verbesserungen. Zwei davon sollen beispielhaft beschrieben werden:

- Für den Weg von Knoten 4 zu Knoten 3 liegt die bisher bester bekannter Länge bei 12. Wir untersuchen den neuen Weg über den Knoten 5. Die Länge des besten bekannte Weges von Knoten 4 nach Knoten 5 lässt sich in Matrix D^4 mit $d^4_{45} = 7$ ablesen; dazu addieren wir die Länge $d^4_{53} = 4$ des besten bekannten Weges von 5 nach 3. Wir erhalten also eine neue Länge von 11, die besser ist als der bisherige Weg mit einer Länge von 12. (Der neue Weg enthält dabei nicht nur den Knoten 5 sondern auch den Knoten 1 als Umwegknoten und lässt sich als $4 \to 1 \to 5 \to 3$ darstellen).
- Wir betrachten den Weg von 1 nach 3 und vergleichen dazu den bisher besten bekannten Weg mit Länge 11 mit dem neuen Weg $1 \to 5 \to 3$. Dessen Länge kann man in D^4 mit $5 + 4 = 9$ ablesen, er ist somit kürzer als der bisherige Weg.

Unsere endgültige Matrix sieht nun folgendermaßen aus:

$$D := D^5 = (d^5_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & \underline{9} & 2 & 5 \\ 2 & \underline{11} & 0 & 3 & \underline{9} & 7 \\ 3 & 12 & 2 & 0 & \underline{11} & 9 \\ 4 & 2 & 9 & \underline{11} & 0 & 7 \\ 5 & 4 & 6 & 4 & 2 & 0 \end{array}$$

Diese Matrix enthält die Längen der jeweils kürzesten Wege zwischen allen Knoten im Netzwerk. Möchte man nicht nur die Entfernungen bestimmen, sondern auch die kürzesten Wege selbst, so muss man sich in jeder Iteration die jeweiligen Vorgänger des Weges merken, anhand derer sich die kürzesten Wege dann später rekonstruieren lassen. Die Vorgänger werden in einer anderen Matrix V^k (Vorgänger bei der Erzeugung der Matrix zum k.ten Umwegknoten) gespeichert. Bei der Initialisierung in Schritt 1 setzt man $v^0_{ij} := i$ für alle Kanten $e = (i, j)$ und lässt die übrigen Einträge wie in der Matrix D^0. Währen der Iteration muss man dann für jeden verbesserten Eintrag auch den jeweiligen Vorgänger aktualisieren. Ersetzt man den aktuellen Weg von i nach j also durch einen Weg über den Knoten k, so wird der Vorgänger v^k_{ij} auf den Vorgänger des Weges von k nach j aus der Matrix V^{k-1} aktualisiert. Wir formulieren das Verfahren zuerst, dann verdeutlichen wir die Vorgängersuche an dem bisherigen Beispiel.

Algorithmus Floyd-Warshall (Längen und kürzeste Wege)

Input: Ein gerichteter Graph $G = (V, E)$ mit Knoten $\{1, 2, \ldots, n\}$ und Kanten $e \in E$. Jede Kante hat eine Länge d_e.

Schritt 1. Setze $d_{ij}^0 = d_e$ und $v_{ij}^0 = i$ für alle Kanten $e = (i, j) \in E$. Gibt es für $i \neq j$ keine Kante (i, j) so setzte $d_{ij}^0 := \infty$ und $v_{ij}^0 := \infty$. Setze weiterhin $d_{ii}^0 := 0$, $v_{ii}^0 := 0$.

Schritt 2.

For $k = 1, ..., n$

for $i = 1, ..., n$

for $j = 1, ..., n$

Falls $d_{ik}^{k-1} + d_{kj}^{k-1} < d_{ij}^{k-1}$

a) Setze $d_{ij}^k := d_{ik}^{k-1} + d_{kj}^{k-1}$

b) Setze $v_{ij}^k := v_{kj}^{k-1}$

Output: Matrix $D := D^n$, die als Einträge die kürzesten Entfernungen für jedes Knotenpaar enthält und Matrix $V := V^n$, die als Einträge die Vorgänger der kürzesten Wege enthält.

Man kann an der Darstellung des Verfahrens leicht ableiten, wie sich die Rechenzeit in Abhängigkeit der Knotenanzahl n verändert: Es werden drei ineinander geschachtelte Schleifen mit jeweils n Iterationen durchlaufen. In jedem Schritt wird einmal addiert und einmal das Minimum gebildet und noch zwei Variablen gesetzt. Das ergibt einen kubischen (also polynomiellen) Aufwand; man sagt das Verfahren hat eine Komplexität von $O(n^3)$.

Wir illustrieren noch das Berechnen der Vorgängermatrix. Dazu initialisieren wir das Verfahren mit der Start-Vorgängermatrix

von \ nach	1	2	3	4	5
1	0	1	–	1	1
2	–	0	2	–	2
3	3	3	0	3	–
4	4	4	–	0	–
5	–	–	5	5	0

Im ersten Schritt werden drei Einträge der Matrix D^0 verbessert, da die entsprechenden Wege über den Knoten 1 als Umwegknoten kürzer sind. Wir erhalten daher eine neue Vorgängermatrix, in der die drei Wege jetzt den Knoten 1 als direkten Vorgänger haben. Im nächsten Schritt wird der Knoten 2 als Umwegknoten gewählt. Die entsprechende Vorgängermatrizen sehen folgendermaßen aus:

$V^1 = (v_{ij}^1) =$

	1	2	3	4	5
1	0	1	–	1	1
2	–	0	2	–	2
3	3	3	0	$\underline{1}$	$\underline{1}$
4	4	4	–	0	$\underline{1}$
5	–	–	5	5	0

$V^2 = (v_{ij}^2) =$

	1	2	3	4	5
1	0	1	$\underline{2}$	1	1
2	–	0	2	–	2
3	3	3	0	1	$\underline{2}$
4	4	4	$\underline{2}$	0	1
5	–	–	5	5	0

Analog untersucht man weiter die Knoten 3 und 4 als Umwegknoten und erhält die folgenden Matrizen, in denen wieder die Einträge unterstrichen sind, bei denen der Weg über den jeweiligen Umwegknoten zu einer Verbesserung geführt hat.

$$V^3 = (v^3_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 2 & 1 & 1 \\ 2 & \underline{3} & 0 & 2 & \underline{3} & 2 \\ 3 & 3 & 3 & 0 & 1 & 2 \\ 4 & 4 & 4 & 2 & 0 & 1 \\ 5 & \underline{3} & \underline{3} & 5 & 5 & 0 \end{array}, \quad V^4 = (v^4_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 2 & 1 & 1 \\ 2 & 3 & 0 & 2 & 3 & 2 \\ 3 & 3 & 3 & 0 & 1 & 2 \\ 4 & 4 & 4 & 2 & 0 & 1 \\ 5 & \underline{4} & 3 & 5 & 5 & 0 \end{array}$$

Im letzten Schritt wird es nun noch einmal interessant: Betrachten wir den Weg von 2 nach 1. Wie wir schon wissen, kann er über den Umwegknoten 5 von seiner bisherigen Länge 15 auf eine Länge von 11 verbessert werden. Wir müssen nun den entsprechenden Vorgängerknoten anpassen. Was aber ist der Vorgänger von Knoten 1 auf diesem kürzesten Weg?
In der vorhergehenden Matrix D^4 haben wir die kürzesten Wege zusammengesetzt, und die Länge des neuen Weges mit $d^4_{25} + d^4_{51} = 7 + 4 = 11$ bestimmt. Die vorhergehende Matrix V^4 hilft uns nun beim Aktualisieren des Vorgängers: Da der Umwegknoten 5 ist, ist der neue Vorgänger für den Weg $2 \to 1$ also der bisherige Vorgänger auf dem Weg $5 \to 1$. Diesen kann man als $v^4_{51} = 4$ ablesen. Die restlichen Werte in V^5 sehen wie folgt aus:

$$V := V^5 = (v^5_{ij}) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & \underline{5} & 1 & 1 \\ 2 & \underline{4} & 0 & 2 & \underline{5} & 2 \\ 3 & 3 & 3 & 0 & \underline{5} & 2 \\ 4 & 4 & 4 & \underline{5} & 0 & 1 \\ 5 & 4 & 3 & 5 & 5 & 0 \end{array}$$

Das Ergebnis unseres Beispiels ist also die Matrix der kürzesten Entfernungen D und die Vorgängermatrix V:

$$D = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & 6 & 2 & 5 \\ 2 & 11 & 0 & 3 & 9 & 7 \\ 3 & 12 & 2 & 0 & 11 & 9 \\ 4 & 2 & 9 & 11 & 0 & 7 \\ 5 & 4 & 6 & 4 & 2 & 0 \end{array} \quad V = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 5 & 1 & 1 \\ 2 & 4 & 0 & 2 & 5 & 2 \\ 3 & 3 & 3 & 0 & 5 & 2 \\ 4 & 4 & 4 & 5 & 0 & 1 \\ 5 & 4 & 3 & 5 & 5 & 0 \end{array}$$

Die Matrix D enthält die Länge der kürzesten Wege in unserem Graphen. Wir demonstrieren abschließend, wie man kürzeste Wege anhand der Vorgängermatrix rekonstruieren kann. Dazu betrachten wir noch einmal den Weg von 2 nach 1. Aus dem Eintrag $d_{21} = 11$ entnehmen wir, dass der kürzeste Weg von 2 nach 1 die Länge 11 hat. Aber wie sieht er aus?
Wir können den Weg von hinten rekonstruieren. Als erstes lesen wir den Vorgänger des Weges von 2 nach 1 ab: $v_{21} = 4$. Der letzte Teil des Weges besteht also aus $\ldots \to 4 \to 1$. Den Knoten vor der 4 können wir erneut in der Vorgängermatrix V ablesen: Es ist der Vorgänger auf dem kürzesten Weg von 2 nach 4, also $v_{24} = 5$. Der Vorgänger von Knoten 5 auf unserem Weg findet sich im Eintrag v_{25} und lässt sich als 2 ablesen — unser Startknoten. Der gesuchte Weg ist also

$$2 \to 5 \to 4 \to 1.$$

Kapitel 4

Netzplantechnik

Netzplantechnik ist wichtig für die Planung von Projekten. Wir beginnen mit einem Beispiel.

Beispiel: Hausbau
Eine Baufirma hat für den Hausbau folgende Übersicht über auszuführende Arbeiten und dafür voraussichtlich benötigte Zeiten angefertigt:

Nummer	Aktivität	Dauer in Tagen	Aktivitäten, die vorher fertig sein müssen
A	Fundament legen	5	-
B	Wände bauen	5	A
C	Fenster und Außentüren einsetzen	3	A,B
D	Außenanstrich	3	A,B
E	Dach aufsetzen	2	A,B
F	Innentüren einbauen	5	A,B,C,E,G
G	Innenanstrich	3	A,B,C,E
H	Boden legen	2	A,B,E

Der Bauherr steht also vor dem Problem, diesen Projektplan zu realisieren. Welche Handwerker sollen wann kommen? Wann ist das Haus fertig?

In diesem Kapitel soll die Methode der Netzplantechnik vorgestellt werden.
Sie dient

- zur Darstellung der Zusammenhänge zwischen einzelnen Aktivitäten,
- zur Entwicklung eines Zeitplans,
- zum Auffinden der kritischen Stellen,
- zur laufenden Kontrolle und Terminüberwachung.

Die hier vorgestellte Methode ist unter dem Namen CPM (Critical Path Method) oder PERT (Program Evaluation and Review Technique) bekannt.

4.1 Aufstellen des Ereignis-Aktivitäts-Netzwerks

Es wird ein sogenanntes Ereignis-Aktivitäts-Netzwerk angefertigt, in dem Ereignisse als Knoten und Aktivitäten als Kanten dargestellt werden.

Abbildung 4.1: Ereignis-Aktivitätsnetzwerk für den Hausbau 1

beziehungsweise

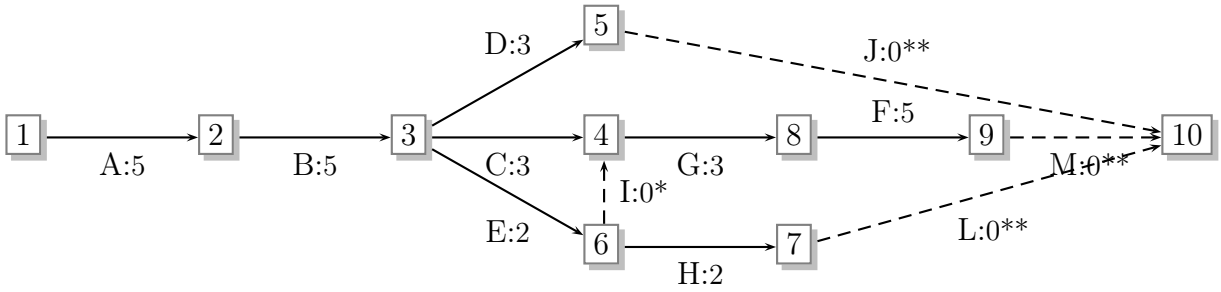


Abbildung 4.2: Ereignis-Aktivitätsnetzwerk für den Hausbau 2: gestrichelte Kanten bezeichnen Scheinaktivitäten

Bemerkungen:

- $*$:Der Pfeil I von 6 nach 4 wird gezeichnet, weil die Aktivität G="Innenanstrich" eine vorherige Beendigung von Aktivitäten C="Fenster und Außentüren einsetzen" und E="Dach aufsetzen" fordert. Da Pfeil G nicht gleichzeitig in Knoten 4 und Knoten 6 starten kann, werden diese durch eine Scheinaktivität I mit Länge 0 verbunden. So wird deutlich, dass G nicht starten kann, bevor C und E nicht beendet sind.

- $**$: Um darzustellen, dass vor Projektende alle Aktivitäten beendet sein müssen, können die Pfeile J von 5="Außenanstrich fertig", M von 9="Innentüren eingebaut" und 7="Boden gelegt" nach 10="Projektende" mit Länge 0 gezeichnet werden. Praktischer ist aber die Variante in Abbildung 4.3, bei der die Pfeile für die Aktivitäten D, F und H direkt im Knoten 10="Projektende" enden.

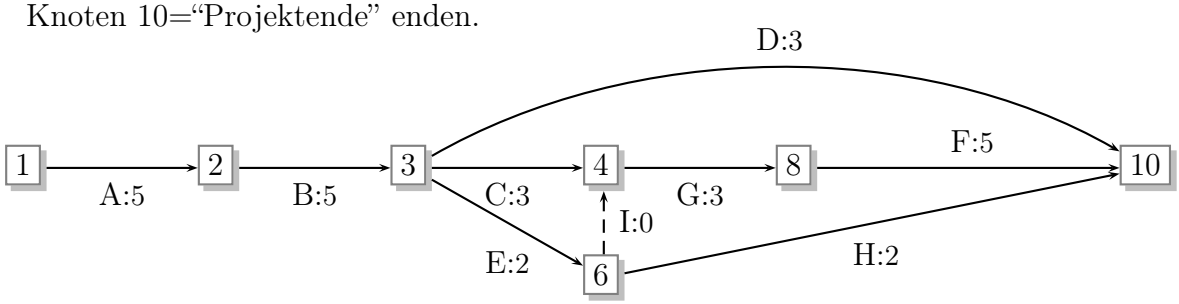


Abbildung 4.3: Ereignis-Aktivitätsnetzwerk für den Hausbau 3: gestrichelte Kanten bezeichnen Scheinaktivitäten

Die folgenden Regeln müssen beim Aufstellen des Ereignis-Aktivitätsnetzwerkes beachtet werden.

- Knoten stellen Ereignisse dar, Kanten Aktivitäten.
- Es gibt ein erstes Ereignis s, das den Projektstart repräsentiert und ein letztes Ereignis t für das Projektende. Für jedes andere Ereignis i gibt es einen gerichteten Weg von s nach i und einen gerichteten Weg von i nach t. Solche Wege können erreicht werden, indem man Scheinaktivitäten einfügt, die s mit i bzw. i mit t verbinden.
- Wenn erforderlich ist, dass zu Beginn der Aktivität B die Aktivität A abgeschlossen ist, muss es einen gerichteten Pfad vom Endpunkt der Kante "Aktivität A" zum Anfangspunkt der Kante "Aktivität B" geben.
  Beispiel: Bevor der Innenanstrich gemacht wird (Aktivität G) muss das Fundament gelegt sein (Aktivität A), die Wände müssen gebaut sein (Aktivität B), das Dach muss aufgesetzt sein (Aktivität E) und Fenster und Außentüren müssen eingesetzt sein (Aktivität C).
- Wenn eine Aktivität B nicht den vorherigen Abschluss einer anderen Aktivität A fordert, darf es auch keinen gerichteten Pfad von der Kante "Aktivität A" nach "Aktivität B" geben.
  Beispiel: Aktivität H baut auf den Aktivitäten A,B und E auf, nicht jedoch auf Aktivität C, also darf die Kante "Aktivität H" nicht in Knoten 4 starten.
  Notfalls muss ein zusätzlichen Knoten eingeführt werden.
  Beispiel: Angenommen es gäbe im obigen Ereignis-Aktivitätsnetzwerk eine weitere Aktivität N, die auf A,B und C aufbaut. Würde man einfach eine weitere Kante "Aktivität N" an 4 anfügen, würde durch die Kante "Scheinaktivität I" auch eine Abhängigkeit von Aktivität E dargestellt werden, die gar nicht vorhanden ist. Stattdessen geht man vor wie in Abbildung 4.4.

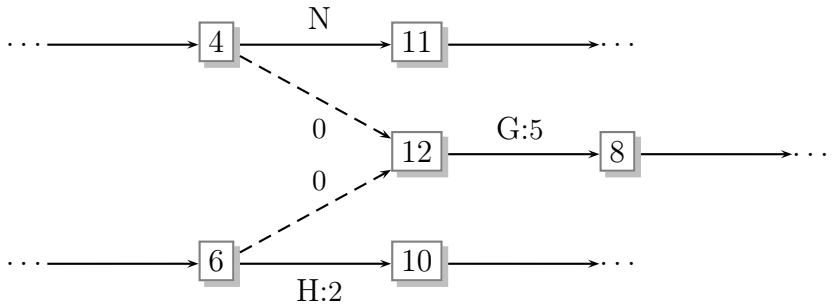


Abbildung 4.4: Einfügen von einem neuen Knoten und Scheinaktivitäten

4.2 Bestimmen des kritischen Pfades und Aufstellen des Zeitplans

Erstes Ziel der Netzplantechnik ist die Erstellung eines Zeitplans. Zur Berechnung der Start- und Endzeiten der Aktivitäten, teilt man sie in kritische und unkritische Aktivitäten auf.

Definition 4.1 *Eine Aktivität heißt* **kritisch**, *wenn jede Verzögerung ihres Ablaufs zu einer Verzögerung des Gesamtprojektes führt. Sonst heißt sie* **unkritisch**.
Ebenso heißt ein Ereignis **kritisch**, *wenn jede Verzögerung seines Eintretens zu einer Verzögerung des Gesamtprojektes führt. Sonst heißt es* **unkritisch**.

Eine Aktivitäts ist also kritisch, wenn selbst eine beliebig kleine Verzögerung von ihr zu einer Verzögerung des Gesamtprojektes führt.

Die Berechnung der Zeitpunkte der Ereignisse wird in zwei Schritten durchgeführt:

Vorwärtsrechnung: Für jedes Ereignis wird ein frühestmöglicher Zeitpunkt festgelegt.

Rückwärtsrechnung: Für jedes Ereignis wird ein letztmöglicher Zeitpunkt festgelegt, zu dem das Ereignis stattfinden kann, ohne das Gesamtprojekt zu verspäten.

Im folgenden werden diese beiden Schritte näher beschrieben.

Vorwärtsrechnung

Der frühestmögliche Zeitpunkt des ersten Ereignisses (Projektstart s) wird normalerweise auf 0 gesetzt, man kann aber auch jeden beliebigen anderen Zeitpunkt nehmen. Der frühestmögliche Zeitpunkt für das Eintreten eines Ereignisses ist der, zu dem ALLE vorher fertig zu stellenden Aktivitäten beendet sind, also die Länge eines längsten gerichteten Pfades vom Startknoten bis zum entsprechenden Knoten (wenn man von dem Zeitpunkt 0 für den Projektstart ausgeht).
Bezeichnet F_i für jeden Knoten i den frühestmöglichen Zeitpunkt und l_{ij} die Länge einer Kante (i, j) aus der Kantenmenge E so berechnet man die frühestmöglichen Zeitpunkte also nach folgender Formel:

$$F_j = \max_{(i,j)\in E} F_i + l_{ij}.$$

Im Beispiel ist also der früheste Zeitpunkt für Ereignis 4

$$13 = \text{“Länge des Pfades 1-A-2-B-3-C-4”}$$

und nicht

$$12 = \text{“Länge des Pfades 1-A-2-B-3-E-6-I-4”}.$$

In Abbildung 4.5 ist die Vorwärtsrechnung an unserem Beispiel dargestellt.
Wir fassen das Vorgehen bei der Vorwärtsrechnung als Algorithmus zusammen.

Algorithmus: Vorwärtsrechnung

Input: Ein Projektnetzwerk $G = (V, E)$ mit Knoten V und Kanten $e \in E$, mit Startknoten s (Projektstart) und Endknoten t (Projektende). Jede Kante $e = (i, j)$ hat eine Länge l_{ij}.

Schritt 1. $F_s := 0$

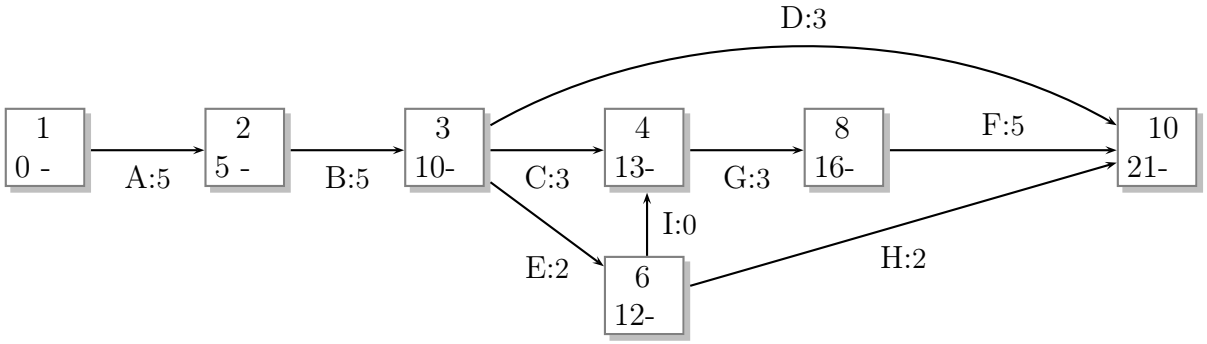


Abbildung 4.5: Vorwärtsrechnung: Die unten links in den Knoten eingetragenen Werte sind die frühestmöglichen Zeitpunkte der Ereignisse

Schritt 2. Wähle einen Knoten j, für den die Zeiten F_i für alle direkten Vorgänger von ihm schon bekannt sind.

Schritt 3. $F_j = \max_{(i,j)\in E} F_i + l_{ij}$

Output: Früheste Zeitpunkte F_j für alle Knoten j.

Man kann die Vorwärtsrechnung aus als lineares Programm formulieren. Dazu wählt man F_i als Variablen. Die Nebenbedingungen sorgen dafür, dass die Mindestdauern der Aktivitäten berücksichtigt werden, sie sind durch

$$F_j - F_i \geq l_{ij} \text{ für alle Kanten } (i,j) \in E$$

gegeben. Schließlich versuchen wir in der Zielfunktion, die Gesamtdauer des Projektes zu minimieren, also $F_t - F_s$. Das entsprechende lineare Programm hat also die folgende Form:

$$\begin{aligned} &\min F_t - F_s \\ \text{so dass} \quad &F_j - F_i \;\; \geq \;\; l_{ij} \text{ für alle Kanten } (i,j) \in E \end{aligned}$$

Mit der Vorwärtsrechnung erhält man also einen Projektplan, der zu der frühestmöglichen Fertigstellung des Gesamtprojektes führt. Er gibt an, wann jede einzelne Aktivität starten kann. Damit könnte man zufrieden sein. Allerdings sagt der Projektplan nichts darüber aus, wie viel Freiheit man bei der Planung der Aktivitäten hat, z.B. ob man einzelne Aktivitäten auch verschieben darf. Um das festzustellen und die oben definierten kritischen Aktivitäten zu finden, benötigt man noch einen weiteren Schritt, die Rückwärtsrechnung.

Rückwärtsrechnung

Bei der Rückwärtsrechnung startet man mit dem letzten Ereignis, also dem Projektende und trägt dort die in der Vorwärtsrechnung berechnete Zeit ein. Nun geht man den Graph von hinten nach vorne durch und berechnet die letztmöglichen Ereigniszeitpunkte durch Subtraktion der Kantenlängen der vom Ereignis ausgehenden Kanten von den vorherigen Ereigniszeitpunkten. Dabei wird immer das Minimum dieser Differenz gewählt.

Bezeichnen wir den spätestmöglichen Zeitpunkt eines Knotens i mit S_i und die Kantenlängen wie zuvor mit l_{ij} so berechnet man die spätestmöglichen Zeitpunkte durch

$$S_j = \min_{(j,i)\in E} S_i - l_{ji}.$$

Im Beispiel wird in Knoten 6

$$13 = 13 - 0 = S_4 - l_{64}$$

als spätester Zeitpunkt gewählt und nicht

$$19 = 21 - 2 = S_1 0 - l_{610}$$

Beispiel: Rückwärtsrechnung

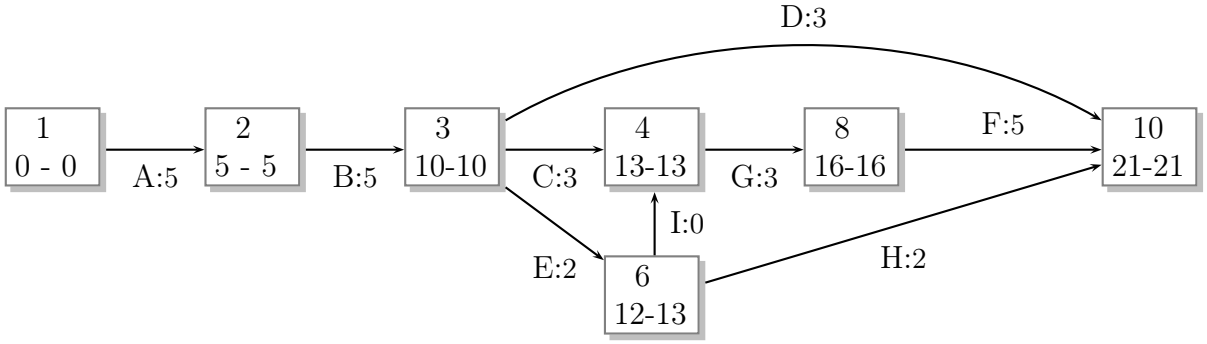


Abbildung 4.6: Rückwärtsrechnung: Die unten rechts in den Knoten eingetragenen Werte sind die spätestmöglichen Zeitpunkte der Ereignisse

Bevor wir die erhaltenen Ergebnisse interpretieren, fassen wir auch die Rückwärtsrechnung als Algorithmus zusammen.

Algorithmus: Rückwärtsrechnung

Input: Ein Projektnetzwerk $G = (V, E)$ mit Knoten V und Kanten $e \in E$, mit Startknoten s (Projektstart) und Endknoten t (Projektende). Jede Kante $e = (i, j)$ hat eine Länge l_{ij}.

Schritt 1. Bestimme den frühesten Zeitpunkt F_t für den Endknoten t mit dem Algorithmus Vorwärtsrechnung.

Schritt 2. $S_t := F_t$

Schritt 3. Wähle einen Knoten j, für den die Zeiten S_i für alle direkten Nachfolger von ihm schon bekannt sind.

Schritt 4. $S_j = \min_{(j,i)\in E} S_i - l_{ji}$

Output: Spätestmögliche Zeitpunkte S_j für alle Knoten j.

Wie kann man nach der Ausführung der beiden Schritte nun kritische Ereignisse und Aktivitäten erkennen?

In unserem Beispiel sieht man, dass bei Knoten 6 der Zeitpunkt der frühestmöglichen Eintretens und des letztmöglichen Eintretens des Ereignisses nicht übereinstimmen. Das Ereignis kann also um einen Tag verzögert werden, ohne dass sich das Projekt dadurch verlängert.
Die kritischen Ereignisse hingegen sind die, deren Verzögerung zu einer Verzögerung des Gesamtprojektes führt, für die also gilt "frühestmöglicher Zeitpunkt = letztmöglicher Zeitpunkt". In diesem Beispiel sind das alle Ereignisse außer Ereignis 6.
Die kritischen Aktivitäten erkennt man daran, dass sie zwischen kritischen Ereignissen verlaufen und dass ihre Länge genau der Differenz der Zeitpunkte von End- und Startknoten entspricht. So ist zum Beispiel G eine kritische Aktivität, D aber nicht, obwohl die zugehörige Kante zwischen den kritischen Ereignissen 3 und 10 verläuft.

Definition 4.2 *Ein* **kritischer Pfad** *ist ein Pfad vom Anfangs- zum Endknoten des Graphen (also von Projektanfang bis Projektende), auf dem nur kritische Ereignisse und kritische Aktivitäten liegen. Das heißt auf dem kritischen Pfad sind frühester und spätester Zeitpunkt der Ereignisse gleich und die Dauer der Aktivitäten entspricht genau der Differenz zwischen Start- und Endknoten.*

Bemerkung: Einen kritischen Pfad muss es immer geben, denn gäbe es keinen, könnten die Anfangszeitpunkte der Aktivitäten nach vorne verschoben und das Projekt früher beendet werden. Allerdings muss der kritische Pfad nicht eindeutig sein, das heißt es kann in einem Netzplan mehrere kritische Pfade geben.

Den kritischen Pfad für das Beispiel "Hausbau" sieht man in Abbildung 4.7.

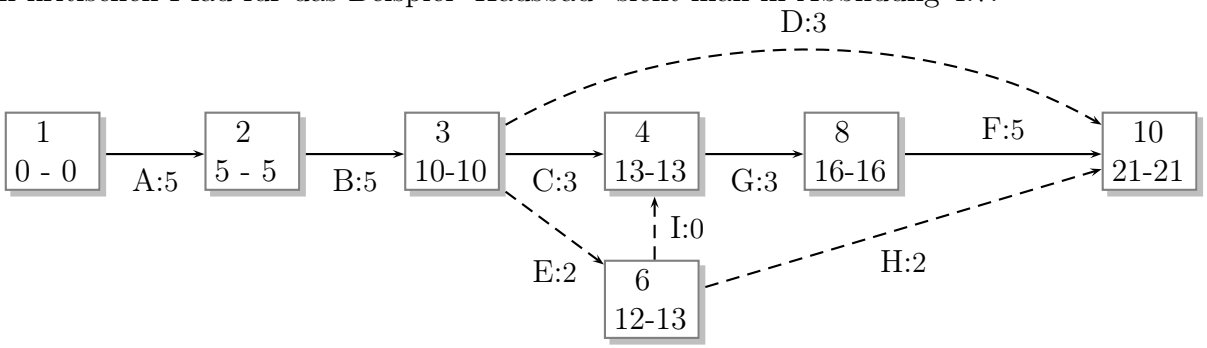


Abbildung 4.7: Die durchgezogene Linie stellt den kritischen Pfad dar.

Neben den frühestmöglichen und spätestmöglichen Zeitpunkten der Ereignisse kann man jetzt auch frühestmögliche und spätestmögliche Zeitpunkte für die Aktivitäten berechnen.
Aktivität D="Außenanstrich" kann zum Beispiel frühestens zum Zeitpunkt 10 (Zeitpunkt des Ereignisses 3) starten. Da das Streichen der Außenwände jedoch nur 3 Tage dauert und erst zu Projektende, also zum Zeitpunkt 21 von Ereignis 10 fertig sein muss, ist der späteste Startzeitpunkt für Aktivität D der 18. Tag.

Man bezeichnet den frühestmöglichen Anfangszeitpunkt der Aktivität a mit FAZ_a, ihren frühestmöglichen Endzeitpunkt mit FEZ_a, ihren spätestmöglichen Anfangszeitpunkt mit SAZ_a

und schließlich den spätestmöglicher Endzeitpunkt von Aktivität a mit SEZ_a. Man kann diese Werte für eine Aktivität $a = (i, j)$ folgendermaßen bestimmen.

- Frühester Anfangszeitpunkt von $a = FAZ_a = F_i$
- Frühester Endzeitpunkt von $a = FEZ_a = F_i + l_{ij}$
- Spätester Endzeitpunkt von $a = SEZ_a = S_j$
- Spätester Anfangszeitpunkt von $a = SAZ_a = S_j - l_{ij}$

Offensichtlich sind für die Aktivitäten auf dem kritischen Pfad frühester und spätester Start ebenso wie frühestes und spätestes Ende gleich. Für die Aktivitäten, die nicht auf dem kritischen Pfad liegen, entstehen Pufferzeiten. Hier gibt es verschiedene Definitionen, am wichtigsten ist die Gesamtpufferzeit.

Definition 4.3 *Die* **Gesamtpufferzeit** *gibt an, wie lange eine Aktivität verlängert oder verzögert werden kann, ohne dass der Endtermin des Projekts hinausgezögert wird. Sie errechnet sich als Differenz des spätestmöglichen und des frühestmöglichen Endzeitpunkts einer Aktivität oder gleichwertig als Differenz des spätestmöglichen und des frühestmöglichen Anfangszeitpunkts einer Aktivität.*

Man erhält die Gesamtpufferzeit GP_a für Aktivität a also durch

$$GP_a = SEZ_a - FEZ_a = SAZ_a - FAZ_a$$

Aktivität	Dauer	frühester Start	frühestes Ende	spätester Start	spätestes Ende	Gesamt-Pufferzeit
A*	5	0	5	0	5	0
B*	5	5	10	5	10	0
C*	3	13	16	13	16	0
D	3	10	13	18	21	8
E	2	10	12	11	13	1
F*	5	16	21	16	21	0
G*	3	13	16	13	16	0
H	2	12	14	19	21	7

Tabelle 4.1: Früheste und späteste Start- und Endzeitpunkte der Ereignisse. * kennzeichnet Aktivitäten auf dem kritischen Pfad.

4.3 Weitere Beispiele

Welche Projekte eignen sich für die Netzplantechnik?

- Das Projekt ist in mehrere Aktivitäten zerlegbar.

- Die Aktivitäten bauen (teilweise) aufeinander auf.
- Der Zeitbedarf für die Aktivitäten kann berechnet oder geschätzt werden.
- Das Projekt kann unabhängig von anderen Vorgängen im Unternehmen betrachtet werden.

Einige Beispiele:

- Hausbau
- Kochen (siehe Abbildung 4.8)

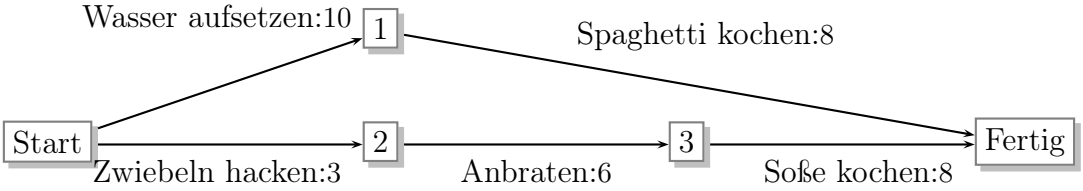


Abbildung 4.8: Netzplan Spaghetti Bolognese

- Anfertigen einer Seminararbeit (siehe Abbildung 4.9)

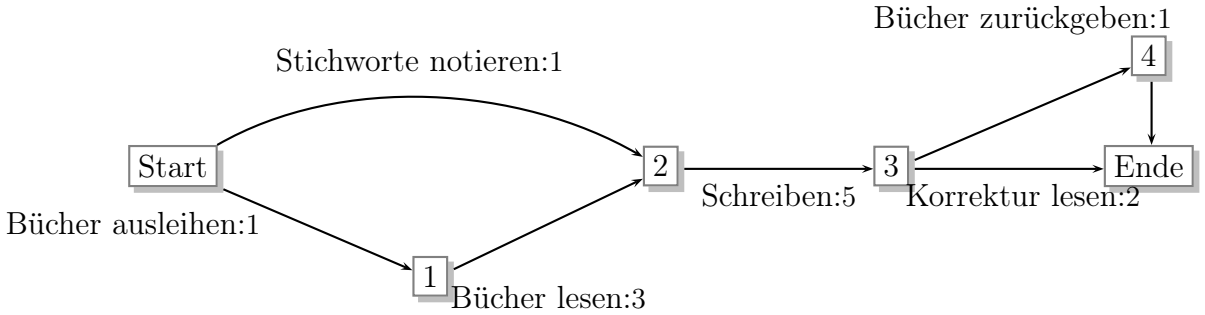


Abbildung 4.9: Netzplan Seminararbeit

- Umstellung des Produktionsablaufes eines Produkts.
- Werbekampagne

In Firmen werden oft sehr große und unübersichtliche Projekte definiert, die Analysetools der Netzplantechnik sind dafür besonders wichtig.

4.4 Erweiterungen

Anpassen des Zeitplans

Angenommen eine Aktivität verzögert sich.

- Wann können die anderen Aktivitäten beginnen?

- Wie lange ist die Gesamtprojektdauer in diesem Fall?
- Ändert sich der kritische Pfad?

Beispiel: Angenommen Aktivität E, das Aufsetzen des Daches verzögert sich um 5 Tage. Bei der zeitlichen Umplanung aller weiteren Aktivitäten kann der Netzplan weiterverwendet werden, indem die betreffende Kante verändert wird. Durch Vorwärts- und Rückwärtsrechnung erhält man wieder früheste und späteste Zeitpunkte.
Achtung: In diesem Fall ändert sich der kritische Pfad. Anhand des veränderten Netzplans lässt sich dann ein neuer Zeitplan aufstellen. Vorwärts- und Rückwärtsrechnung für dieses Beispiel sind in den Abbildungen 4.10 und 4.11 dargestellt.

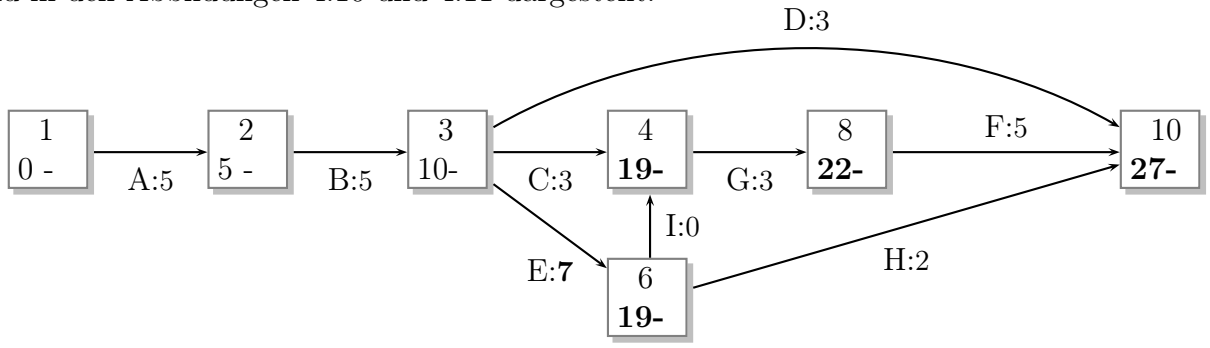


Abbildung 4.10: Aktivität E verzögert sich um 5 Tage: Vorwärtsrechnung

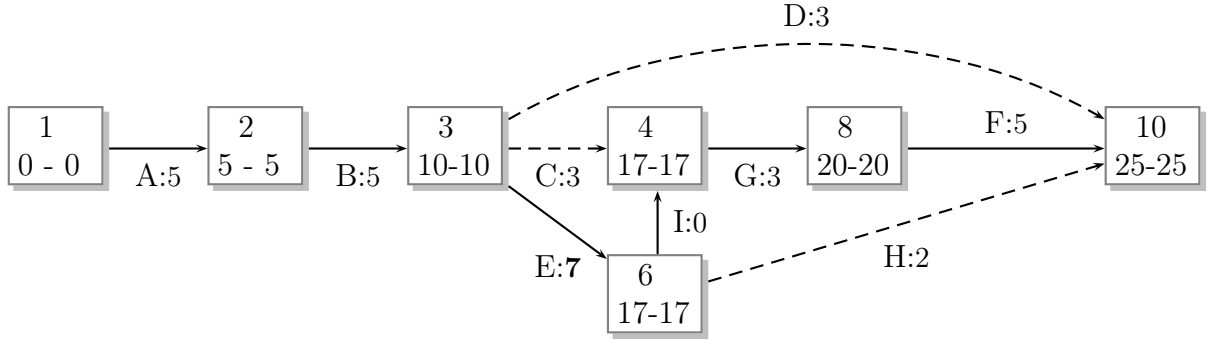


Abbildung 4.11: Aktivität E verzögert sich um 5 Tage: Rückwärtsrechnung und kritischer Pfad

Stochastische Aktivitätsdauern

Ein Bauunternehmer mag die Dauer der zum Hausbau durchzuführenden Projekte vielleicht noch präzise abschätzen können. Ist das geplante Projekt aber zum Beispiel die Durchführung einer Werbekampagne, so ist schwer vorhersehbar, wie lange zum Beispiel die Aktivität "Entwurf eines griffigen Slogans" dauert. Meistens liegen aber immerhin Erfahrungswerte vor, wie lange eine Aktivität normalerweise dauert und wie viel Zeit sie minimal oder maximal in Anspruch nimmt.
Unter der Annahme die Dauern seien Beta-verteilt, kann man dann nach einer einfachen Formel die durchschnittliche Dauer berechnen und im Netzwerk eintragen. Außerdem kann man die Varianz jeder Aktivitätsdauer bestimmen. Nun erhält man für jedes Ereignis den erwarteten Eintrittszeitpunkt und seine Varianz.

Einbeziehen der Ressourcen

Oft sind Ressourcen (wie zum Beispiel Arbeitskräfte) nicht unbegrenzt vorhanden. Nach Erstellung des Zeitplans mit Pufferzeiten versucht man deshalb, die unkritischen Aktivitäten innerhalb ihres Zeitrahmens so zu verschieben, dass die Ressourcen ausreichen oder festzustellen, dass das nicht möglich ist, ohne die Projektdauer zu verlängern. Auch wenn genügend Ressourcen vorhanden sind, versucht man oft, die unkritischen Aktivitäten so zu verschieben, dass der Ressourcenverbrauch möglichst gleichmäßig ist. Andererseits kann es auch von Vorteil sein, mit den unkritischen Aktivitäten so früh wie möglich zu beginnen, damit eventuelle Verzögerungen dieser oder nachfolgender unkritischer Aktivitäten weniger schnell zur Verzögerung des Gesamtprojekts führen. Beide Überlegungen sind gegeneinander abzuwägen.

Einbeziehen der Kosten

Es ist vorstellbar, dass die Kosten für eine Aktivität nicht fest sind, sondern von der Aktivitätsdauer abhängen, das heißt steigen, wenn die Aktivitätsdauer sinkt. Ein Beispiel hierfür wäre der Kauf einer effizienteren Anlage, der zwar Kosten verursacht, aber auch die Arbeitszeit reduziert oder das Einstellen zusätzlicher Mitarbeiter. Unter diesen Voraussetzungen ließe sich eine Aktivität also beschleunigen, wenn man bereit wäre, dafür höhere Kosten in Anspruch zu nehmen. Sinn macht das offensichtlich aber nur für Aktivitäten auf dem kritischen Pfad, da die Dauer anderer Aktivitäten die Projektdauer insgesamt nicht beeinflusst. Auch die Verkürzung der Dauer der kritischen Aktivitäten macht nur solange Sinn, wie die Verkürzung der Kantenlänge im Ereignis-Aktivitätsnetzwerk nicht dazu führt, dass ein anderer Pfad zwischen Projektanfang und Projektende zum kritischen Pfad wird. In so einem Fall müsste dann der neue kritische Pfad betrachtet werden.

Kapitel 5

Knotenfärbungsprobleme

5.1 Einführung

Beispiel: Raumvergabe in einem Tagungscenter
In einem Tagungscenter stehen fünf Räume zur Verfügung, die für Veranstaltungen vermietet werden und mit 1 bis 5 durchnummeriert sind. Für die nächste Woche haben sich sieben Gruppen angekündigt:

Gruppe	Tagungsdauer
A	Montag-Mittwoch
B	Mittwoch-Samstag
C	Dienstag und Mittwoch
D	Donnerstag-Sonntag
E	Mittwoch-Freitag
F	Montag-Mittwoch
G	Samstag und Sonntag

Die einzelnen Tagungen sollen immer im gleichen Raum stattfinden.

- Reichen die Räume des Tagungszentrums aus?
- Es gibt eine weitere Anfrage für eine die ganze Woche dauernde Tagung. Kann diese auch im Tagungscenter stattfinden?

Um das Problem zu lösen, zeichnet man einen Graph, dessen Knoten die einzelnen Veranstaltungen darstellen. Man verbindet alle Veranstaltungsknoten, die sich zeitlich überschneiden, mit einer (ungerichteten) Kante. Der Graph ist in Abbildung 5.1 dargestellt.
Nun ordnet man jedem Knoten, also jeder Veranstaltung, eine Raumnummer zu. Knoten, die durch eine Kante verbunden sind, dürfen nicht die gleiche Raumnummer erhalten. Natürlich gibt es viele Möglichkeiten, die Raumnummern zu verteilen. Um eine Antwort auf die Frage zu finden, ob die fünf Räume des Tagungscenters ausreichen, sucht man eine Zuordnung der Raumnummern bei der höchstens fünf unterschiedliche Räume benutzt werden.
Um während der ganzen Woche einen Raum an eine weitere Gruppe vergeben zu können, müsste man einen Belegungsplan finden, der nur vier Räume benutzt, um den fünften Raum

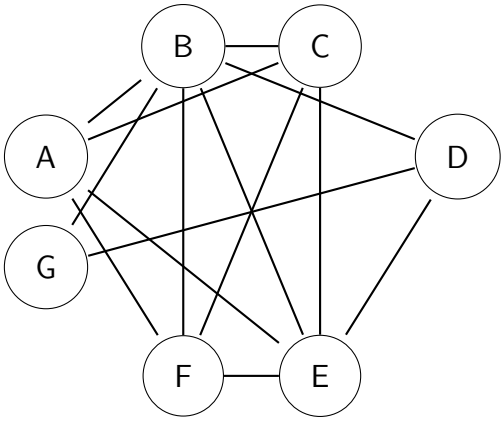


Abbildung 5.1: Graph für das Tagungsproblem

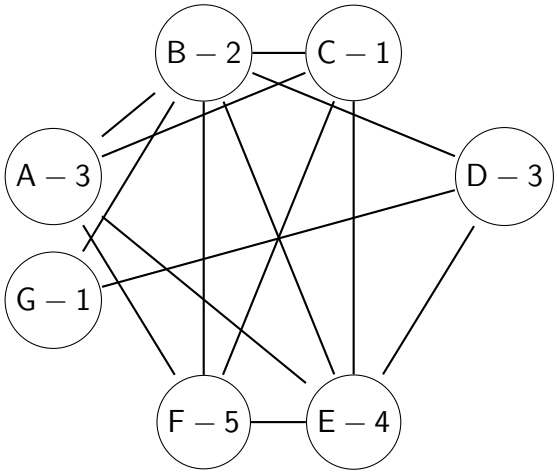


Abbildung 5.2: Raumzuordnung für das Tagungsproblem

für die zusätzliche Gruppe freizuhalten. Betrachtet man die Tabelle sieht man jedoch, dass am Mittwoch fünf verschiedene Veranstaltungen stattfinden, an diesem Tag also kein Raum an eine weitere Gruppe vergeben werden kann. Im Graph erkennt man dies daran, dass alle am Mittwoch stattfinden Veranstaltungen (A,B,C,E und F) einen vollständigen Untergraphen bilden, das heißt paarweise durch Kanten verbunden sind. Jeder dieser Knoten muss also eine andere Nummer bekommen.

Statt die Räume durchzunummerieren könnte man auch für jeden Raum eine andere Farbe wählen und die Knoten einfärben. Benachbarte (das heißt durch eine Kante verbundene) Knoten dürfen dann nicht die gleiche Farbe bekommen. Probleme dieser Art nennt man deshalb auch *Knotenfärbungsprobleme*. In der Graphentheorie werden auch *Kantenfärbungsprobleme* betrachtet, bei denen nicht die Knoten sondern die Kanten eingefärbt werden, wir beschränken uns hier aber auf Knotenfärbungsprobleme und bezeichnen diese im Weiteren auch kurz als Färbungsprobleme.

Definition 5.1 *Sei G ein einfacher ungerichteter Graph. Eine* **k-Färbung** *von G ist eine Zuordnung von höchstens k Farben (oder Zahlen, Buchstaben, Symbolen,...) zu den Knoten von G, so dass benachbarte Knoten unterschiedliche Farben (Zahlen, Buchstaben, Symbolen,...)*

bekommen.
Gibt es eine k-Färbung für den Graph G, so heißt der Graph **k-färbbar**.
Die kleinste Zahl k, für die ein Graph G k-färbbar ist, heißt **chromatische Zahl** *von G und wird mit $\chi(G)$ abgekürzt.*
Das **Färbungsproblem** *ist das Problem, für einen Graph G eine k-Färbung mit möglichst kleinem k zu finden.*

Der Graph G aus dem Beispiel ist also 5-färbbar, aber auch 6-färbbar, 7-färbbar,..., wie man sich leicht überlegen kann. Aber er ist, wie wir gesehen haben, nicht 4-färbbar, also ist die chromatische Zahl $\chi(G) = 5$.

5.2 Einfache Färbungsprobleme

Beispiel: Färbung von vollständigen Graphen
Wir betrachten zunächst einen vollständigen Graph G, d.h. ein Graph, in dem alle n Knoten paarweise durch Kanten miteinander verbunden sind. Da jedes Knotenpaar durch eine Kante verbunden ist, braucht man für jeden Knoten eine andere Farbe. Die chromatische Zahl ist also $\chi(G) = n$.

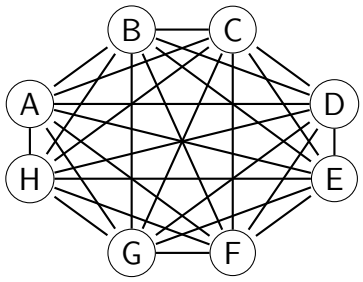


Abbildung 5.3: Vollständiger Graph mit acht Knoten

Beispiel: Färbung von bipartiten Graphen
Wie auf S. 22 bereits eingeführt, heisst ein einfacher Graph in der Graphentheorie bipartit, falls sich seine Knoten in zwei disjunkte Teilmengen A und B aufteilen lassen, so dass zwischen den Knoten innerhalb beider Teilmengen keine Kanten verlaufen.
Für die Färbung eines bipartiten Graphen braucht man nur zwei Farben, eine für die Knotenmenge A und eine für die Knotenmenge B, die chromatische Zahl ist also für jeden bipartiten Graph G $\chi(G) = 2$.

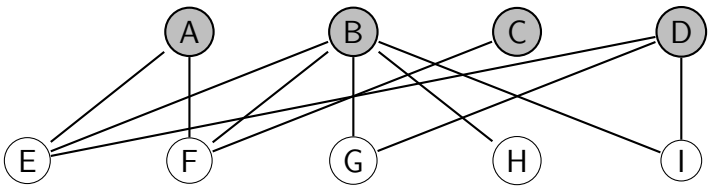


Abbildung 5.4: Färbung eines bipartiten Graphen

Beispiel: Färbung von Kreisen
Ein Kreis ist ein zusammenhängender einfacher Graph, in dem jeder Knoten mit genau zwei

weiteren Knoten verbunden ist. Zur Färbung eines Kreises kann man an ihm entlanggehen und die Knoten abwechselnd in zwei Farben einfärben. Ist die Knotenanzahl gerade, geht das gut und $\chi(G) = 2$. Ist die Knotenanzahl ungerade, braucht man noch eine weitere Farbe für den letzten einzufärbenden Knoten.

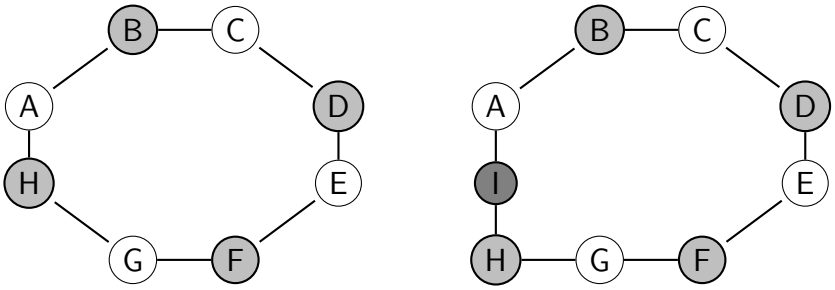


Abbildung 5.5: Färbung von Kreisen

Beispiel: Färbung von Bäumen
Ein Baum G ist ein zusammenhängender, kreisfreier Graph. Auch im Baum kann man von einem beliebigen Knoten ausgehend die Knoten abwechselnd einfärben. Die chromatische Zahl $\chi(G)$ ist also immer 2. das liegt dran, dass Bäume immer bipartit sind.

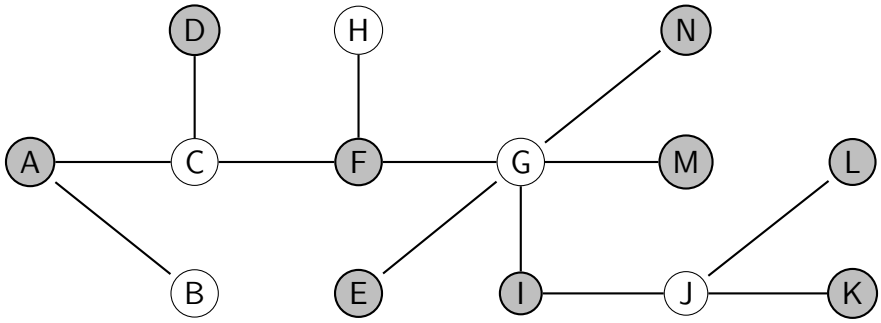


Abbildung 5.6: Färbung eines Baums

5.3 Formulierung als ganzzahliges Programm

Das Färbungsproblem kann man auch als boolesches Programm formulieren.

Variablen: Wir gehen davon aus, dass genug verschiedene Farben zum Einfärben der Knoten zur Verfügung stehen, zum Beispiel n, für jeden Knoten eine. Nun definieren wir:

$$y_r = \begin{cases} 1 & \text{falls Farbe } r \text{ verwendet wird} \\ 0 & \text{sonst} \end{cases}$$

und

$$x_{ir} = \begin{cases} 1 & \text{falls Knoten } i \text{ mit Farbe } r \text{ eingefärbt wird} \\ 0 & \text{sonst.} \end{cases}$$

Nebenbedingungen: Jeder Knoten muss in einer Farbe gefärbt werden:

$$\sum_r x_{ir} = 1 \quad \forall \text{ Knoten } i \ .$$

Wird ein Knoten in einer Farbe eingefärbt, muss diese Farbe verwendet werden, also gilt:

$$x_{ir} \leq y_r \quad \forall \text{ Knoten } i \ \forall \text{ Farben } r.$$

Zwei Knoten i und j, die durch eine Kante $\{i, j\}$ verbunden werden, dürfen nicht beide die gleiche Farbe r haben:

$$x_{ir} + x_{jr} \leq 1 \quad \forall \text{ Kanten } \{i, j\} \ \forall \text{ Farben } r.$$

Und die 0-1-Bedingungen:

$$\begin{aligned} y_r \in \{0, 1\} \qquad & \forall \text{ Farben } r \\ x_{ir} \in \{0, 1\} \qquad & \forall \text{ Knoten } i \ \forall \text{ Farben } r. \end{aligned}$$

Zielfunktion: Zu minimieren ist die Anzahl der benutzten Farben, also:

$$\min \sum_{r=1}^{n} y_r.$$

Das Gesamtmodell ergibt sich damit als

$$\begin{aligned} & \min \sum_{r=1}^{n} y_r \\ \text{so dass } & \sum_r x_{ir} = 1 && \forall \text{ Knoten } i \\ & x_{ir} \leq y_r && \forall \text{ Knoten } i \ \forall \text{ Farben } r \\ & x_{ir} + x_{jr} \leq 1 && \forall \text{ Kanten } \{i, j\} \ \forall \text{ Farben } r \\ & y_r \in \{0, 1\} && \forall \text{ Farben } r \\ & x_{ir} \in \{0, 1\} && \forall \text{ Knoten } i \ \forall \text{ Farben } r \end{aligned}$$

5.4 Weitere Beispiele

- Gleisbelegung für die Bahn
  Nach der Fahrplanerstellung, bei der die Ankunfts- und Abfahrtszeiten aller Züge an allen Bahnhöfen festgelegt werden, beginnt die Bahn mit der Detailplanung:
  Welcher Zug soll auf welches Gleis?
  Für einen festen Bahnhof zeichnet man die durchfahrenden Züge als Knoten. Zwei Knoten bekommen eine Kante, wenn sich die Zeiten der enstprechenden Züge überlappen; sie also nicht auf das gleiche Gleis dürfen. Eine Färbung des Graphen gibt an, wie viele Gleise man braucht; also ob der Fahrplan für den Bahnhof realisierbar ist. Ist er das nicht, müssen Zeiten verschoben werden.

- Verteilung von Tieren in einem Zoo
  Ein neuer Zoo wird gebaut. Dabei soll nicht jede Tierart ein einzelnes Gehege bekommen, stattdessen sollen mehrere Arten zusammen leben. Aus Sicherheitsgründen können manche Tiere sich kein Gehege teilen.
  Wie viele Gehege braucht der Zoo? Wie sollen die Tiere darauf verteilt werden?
  Die Tierarten zeichnet man als Knoten, für Tiere die sich nicht vertragen fügt man eine Kante ein. Eine zulässige Knotenfärbung gibt nun eine "sichere" Verteilung der Tiere auf die Gehege an.

- Lagerung von Chemikalien
  Eine Chemiefirma steht vor dem Problem, dass einige ihrer Chemikalien bei räumlicher Nähe miteinander reagieren könnten und will diese Chemikalien deshalb getrennt aufbewahren.
  Wie viele Räume werden benötigt und wo soll welche Chemikalie gelagert werden?
  Für die Lösung als Knotenfärbungsproblem stellt man die Chemikalien als Knoten dar, zwischen gefährlichen Chemikalienpaaren zeichnet man eine Kante. Durch eine Knotenfärbung erhält man eine Aufteilung der Chemikalien auf verschiedene Lagerräume.

5.5 Schranken

Wie wir in Abschnitt 5.2 gesehen haben, ist es für einige Graphentypen einfach, die chromatische Zahl zu bestimmen. Im Allgemeinen ist das Knotenfärbungsproblem aber NP-schwer. Deswegen ist es wichtig, zumindest Schranken für die chromatische Zahl bestimmen zu können.

Definition 5.2 *Als* **obere Schranke** s_o *für ein Problem bezeichnet man eine Zahl, von der man weiß, dass sie größer oder gleich dem gesuchten Zielfunktionswert ist.* **Untere Schranke** s_u *nennt man eine Zahl, die in jedem Fall kleiner oder gleich dem Zielfunktionswert ist.*
Im Fall des Färbungsproblems gilt also

$$s_u \leq \chi(G) \leq s_o.$$

Hat man eine obere und eine untere Schranke gefunden, liegt der Zielfunktionswert daher immer dazwischen. Um den Bereich, in dem der Zielfunktionswert liegen könnte, möglichst stark einzuschränken, versucht man also, eine möglichst kleine obere Schranke und eine möglichst große untere Schranke zu finden. Findet man sogar gleiche obere und untere Schranken, hat man den Zielfunktionswert gefunden, denn dann gilt $s_u = \chi(G) = s_o$.

Obere Schranken

- Färbt man jeden Knoten eines Graphen G in einer unterschiedlichen Farbe, erhält man immer eine zulässige Färbung. Die Anzahl der Knoten ist also eine obere Schranke für $\chi(G)$.

- Sei G ein einfacher Graph mit maximalem Knotengrad d, das heißt in jedem Knoten gehen höchstens d Kanten ab. Dann ist G $(d+1)$-färbbar, denn für jeden Knoten gibt es höchstens d verbotene Farben, nämlich die Farben der höchstens d Nachbarn.

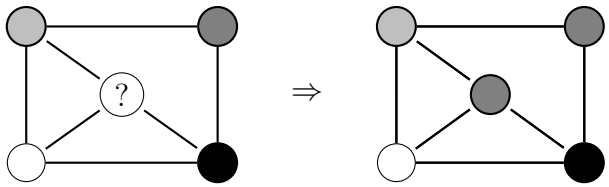


Abbildung 5.7: Der mittlere Knoten hat 3 Nachbarn, also Knotengrad 3.

Ist der Graph G kein vollständiger Graph und auch kein Kreis mit einer ungeraden Kantenzahl, dann gilt sogar $\chi(G) \leq d$. Nicht nur $d+1$ sondern auch d ist also eine obere Schranke für die chromatische Zahl. Diese Aussage ist unter dem Namen Satz von Brooks bekannt.

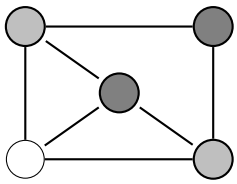


Abbildung 5.8: Weitere Möglichkeit der Knotenfärbung mit nur $d = 3$ Farben.

- Jede gefundene Knotenfärbung mit k-Farben liefert eine obere Schranke für die Anzahl der zur Knotenfärbung benötigten Farben. Im Anfangsbeispiel etwa wurde eine Verteilung der Gruppen auf fünf Räume angegeben, dies bedeutet natürlich dass man höchstens fünf Räume zur Verteilung der Gruppen braucht, also ist $\chi(G) \leq 5$.

Untere Schranken

- In Abschnitt 5.2 haben wir schon gesehen, dass die chromatische Zahl eines vollständigen Graphen immer gleich der Anzahl der Knoten des Graphen ist. Enthält ein Graph G mit n Knoten als Untergraph den vollständigen Graph g mit m Knoten muss also gelten $\chi(G) \geq m$ und wir haben eine untere Schranke m gefunden. Die chromatische Zahl ist also immer mindestens so groß wie die größte Clique des Graphen.

Diese Taktik haben wir auch schon im Anfangsbeispiel in Abschnitt 5.1 angewendet um zu zeigen, dass im Tagungscenter mindestens fünf Räume verwendet werden.
Wie oben erklärt konnten wir in diesem Beispiel die chromatische Zahl bestimmen, weil wir eine zulässige Lösung (also eine obere Schranke) gefunden haben, die gleich der unteren Schranke war. Stimmen obere und untere Schranke überein, so ist die chromatische Zahl gleich diesem Wert.

5.6 Das Landkartenfärbungsproblem

Beim Landkartenfärbungsproblem versucht man die einzelnen Länder eine Landkarte so zu färben, dass keine zwei benachbarten Länder die gleiche Farbe haben. Dieses Problem lässt sich als Knotenfärbungsproblem umschreiben. Dazu zeichnet man für jedes Land einen Knoten, benachbarte Länder werden durch Kanten verbunden (siehe Abbildungen 5.10 und 5.11).

Definition 5.3 *Ein Graph heißt* **planar**, *wenn er gezeichnet werden kann, ohne dass sich seine Kanten schneiden.*

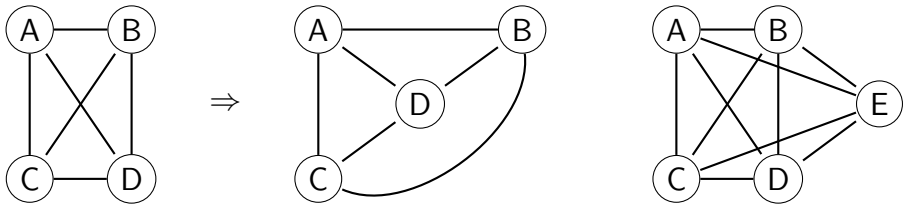


Abbildung 5.9: Der erste Graph ist planar, denn er lässt sich ohne sich schneidende Kanten zeichnen. Beim zweiten Graph ist dies nicht möglich, er ist also nicht planar.

Satz 5.4 *Planare Graphen sind* 4*-färbbar.*

Da Landkarten als Graphen dargestellt immer planar sind, gilt also auch:

Korollar 5.5 *Man kann jede Landkarte mit nur* 4 *(oder weniger) Farben so färben, dass keine benachbarten Länder die gleiche Farbe haben.*

Diese Vermutung wurde schon im Jahre 1852 aufgestellt. Aber erst mehr als 100 Jahre später, im Jahr 1976 gelang es, sie unter Zuhilfenahme von Computern zu beweisen.
Beispiel:

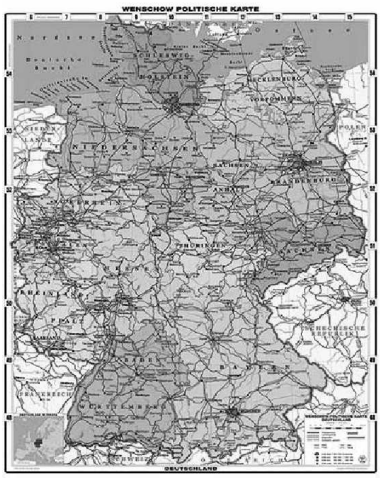


Abbildung 5.10: Deutschland politisch . Der entsprechende Graph ist in Abbildung 5.11 dargestellt.

5.7 Der Greedy-Algorithmus zum Lösen des Färbungsproblems

Wie schon in Abschnitt 5.5 erwähnt ist das Knotenfärbungsproblem NP-schwer. Exakte Lösungsverfahren würden also im Allgemeinen viel zu lange dauern. Deshalb wird im Folgenden

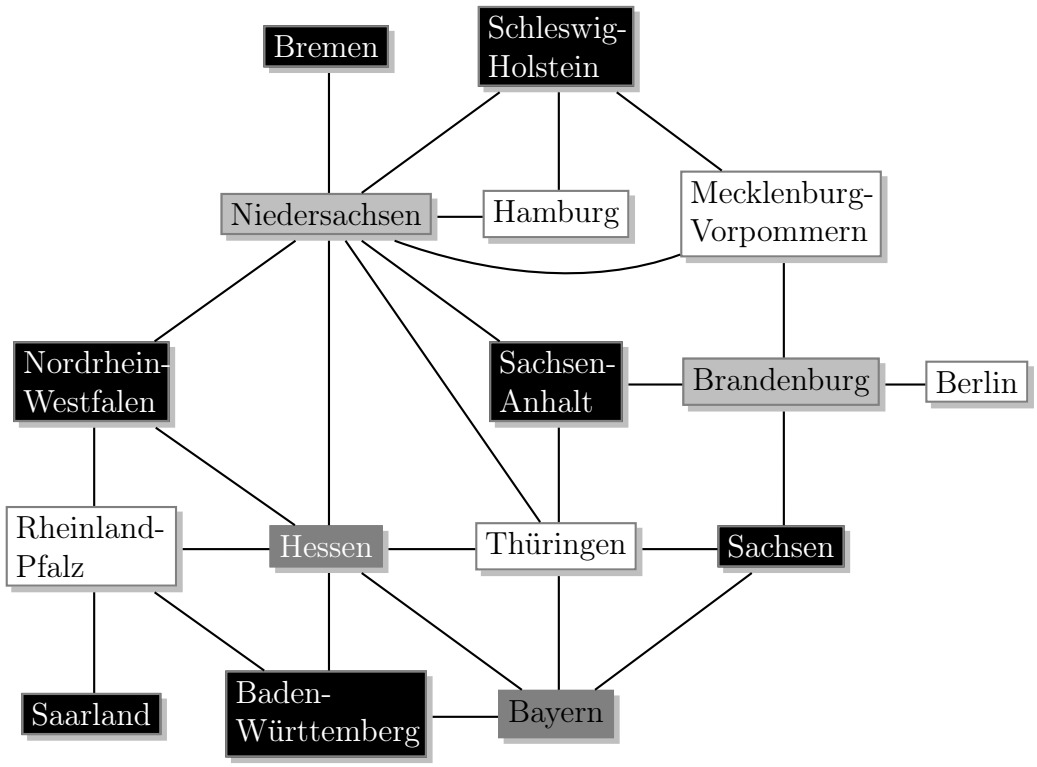


Abbildung 5.11: Knotenfärbung für Deutschland

eine Heuristik für das Knotenfärbungsproblem vorgestellt. Heuristiken sind Lösungsverfahren, bei denen man nicht weiß, ob die ausgegebene Lösung optimal ist. Im Allgemeinen muss die Ausgabe einer Heuristik noch nicht einmal zulässig sein. Heuristiken sind aber normalerweise so konstruiert, dass sie in den meisten Fällen gute Lösungen erzeugen.

Algorithmus: Knotenfärbung

Input: Ein Graph $G = (V, E)$ mit Knoten V und Kanten $e \in E$, eine Menge von Farben $R = \{r_1, \ldots, r_n\}$.

Schritt 1. Nummeriere die Knoten in beliebiger Reihenfolge.

Schritt 2. Wähle den Knoten $i \in V$ mit kleinster Nummer, dem noch keine Farbe zugeordnet wurde. Färbe ihn mit der Farbe $r_j \in R$ mit niedrigstem Index, mit der noch kein benachbarter Knoten gefärbt worden ist. Wiederhole diesen Schritt, bis kein Knoten mehr übrig ist.

Output: Eine Knotenfärbung für den Graph G.

Dieser Algorithmus ist ein sogenannter Greedy-Algorithmus. "Greedy" ist Englisch für "gierig" und steht dafür, dass der Algorithmus in jedem Schritt das tut, was lokal am besten erscheint. Ein weiterer Greedy-Algorithmus ist der Algorithmus von Prim zur Bestimmung eines minimal spannenden Baumes aus Kapitel 10. Der Greedy-Algorithmus für das Knotenfärbungsproblem

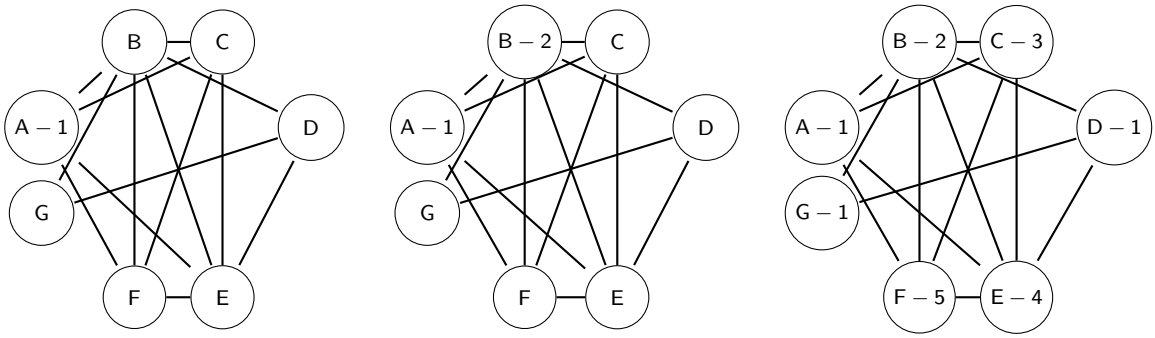


Abbildung 5.12: Greedy-Algorithmus für das Tagungsproblem

findet zwar, anders als der Algorithmus von Prim, nicht immer eine Optimallösung, aber meistens eine gute Lösung. Im in Abbildung 5.12 gezeigten Beispiel ist die gefundene Lösung sogar optimal.

Kapitel 6

Das Transportproblem

6.1 Problembeschreibung und Modellierung

Das Transportproblem verdeutlicht man sich am Besten an einem Beispiel:

Beispiel:
Eine Firma produziert ihr Produkt an drei verschiedenen Produktionsstätten. Danach soll es zu vier Empfängern (z.B. weiterverarbeitende Fabriken, Verkaufsstätten) transportiert werden. Die in den Produktionsstätten maximal produzierbaren Mengen sind

1	2	3
15	25	5

Die von den Empfängern benötigten Mengen sind

1	2	3	4
5	15	15	10

Die Transportkosten zwischen Produktionsstätten und Empfängern sind in folgender Tabelle dargestellt:

	1	2	3	4
1	10	0	20	11
2	12	7	9	20
3	0	14	16	18

Allgemein kann das Problem so formuliert werden:

Es sei n die Anzahl der Produktionsstätten und m die Anzahl der Empfänger. Jeder Produktionsstandort i kann bis zu a_i Einheiten pro Tag produzieren. Jeder Empfänger j benötigt pro Tag b_j Einheiten.
Die Produkte können von jedem Standort zu jedem Empfänger transportiert werden, allerdings fallen abhängig von Standort i und Empfänger j die Lieferkosten w_{ij} pro Einheit an.

- Wie kann das Produkt möglichst kostengünstig an die Empfänger verteilt werden?

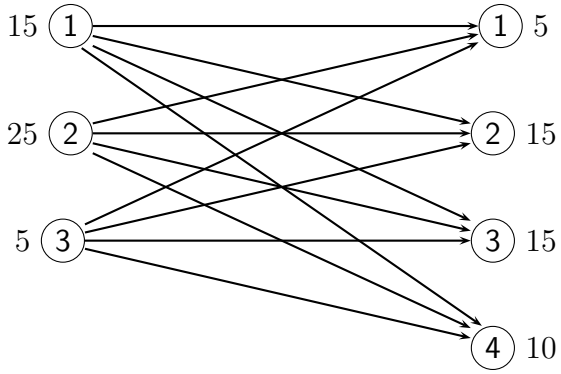


Abbildung 6.1: Das Transportproblem als Graph

Als Graph kann das Problem wie in Abbildung 6.1 dargestellt werden:
Damit der Bedarf an Einheiten in allen Produktionsstätten gedeckt werden kann, muss die insgesamt produzierte Menge mindestens so groß sein wie die von den Empfängern benötigte Menge. Wir haben also folgende Zulässigkeitsbedingung:

$$\sum_{i=1}^{n} a_i \geq \sum_{j=1}^{m} b_j$$

Diese Bedingung lässt sich vereinfachen: Gilt $\sum_{i_1}^{n} a_i > \sum_{j=1}^{m} b_j$ führen wir einen zusätzlichen Empfänger $m+1$, auch "Dummy" genannt, mit Bedarf $d_{m+1} = \sum_{i_1}^{n} a_i - \sum_{j=1}^{m} b_j$ und Transportkosten $w_{i,m+1} = 0$ ein, wie in Abbildung 6.2 dargestellt. Dadurch ergibt sich unter Berücksichtigung des neuen Dummy-Empfängers:

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m+1} b_j.$$

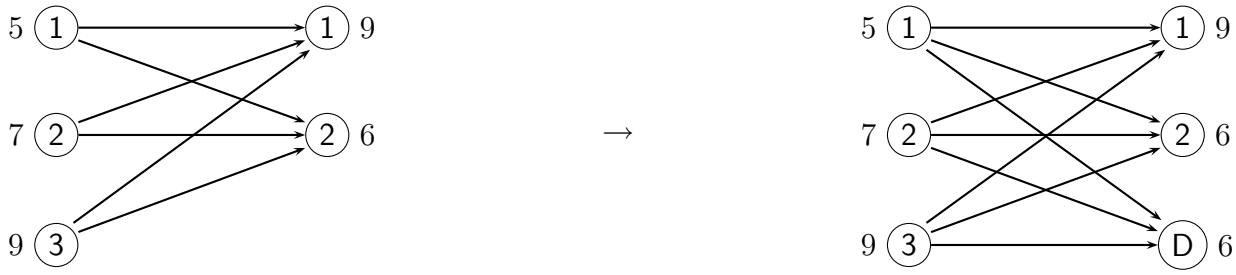


Abbildung 6.2: Einführung eines Dummy-Empfängers

Da die Einführung eines solchen Dummy-Knotens für jedes zulässige Transportproblem möglich ist, gehen wir im Folgenden von der Gleichheit des Gesamtbedarfs und der gesamten produzierten Menge aus, es gilt also immer

$$\sum_{i} a_i = \sum_{j} b_j.$$

Es ist auch eine Formulierung als lineares Programm möglich:

Variablen: Wir definieren x_{ij} als die Menge der Einheiten, die von Produktionsstandort i zu Empfänger j transportiert werden.

Nebenbedingungen: Der Bedarf jedes Empfängers muss gedeckt werden, also:

$$\sum_{i=1}^{n} x_{ij} \geq b_j \quad \forall \text{ Empfänger } j.$$

Jede Produktionsstätte kann höchstens soviel liefern, wie sie produziert hat:

$$\sum_{j=1}^{m} x_{ij} \leq a_i \quad \forall \text{ Produktionsstätten } i.$$

Unter Beachtung der Vereinfachung $\sum_{i_1}^{n} a_i = \sum_{j=1}^{m+1} b_j$ nach Einführung einer Dummy-Senke ergibt sich, dass keine Produktionsstätte i weniger als a_i Einheiten produzieren darf und kein Empfänger j mehr als b_j empfangen kann. Also lassen sich obige Bedingungen umformulieren zu:

$$\sum_{i=1}^{n} x_{ij} = b_j \qquad \forall \text{ Empfänger } j,$$

$$\sum_{j=1}^{m} x_{ij} = a_i \qquad \forall \text{ Produktionsstätten } i.$$

Außerdem kann die gelieferte Menge nicht negativ sein:

$$x_{ij} \geq 0 \quad \forall \text{ Produktionsstätten } i, \ \forall \text{ Empfänger } j.$$

Zielfunktion: Zu minimieren sind die Transportkosten, also:

$$\min \sum_{n} \sum_{m} w_{ij} x_{ij}$$

Das Gesamtmodell ergibt sich damit als

$$\begin{aligned}
&\min \sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} x_{ij} \\
&\text{s.d.} \sum_{j=1}^{m} x_{ij} = a_i \quad \forall \text{ Produktionsstätten } i \\
&\quad \sum_{i=1}^{n} x_{ij} = b_j \quad \forall \text{ Empfänger } j \\
&\quad x_{ij} \geq 0 \quad \forall \text{ Produktionsstätten } i, \ \forall \text{ Empfänger } j.
\end{aligned}$$

6.2 Weitere Beispiele

Beispiel: **Zuordnungsproblem**

In einem Unternehmen fallen verschiedene Aufgaben an. Jeder Angestellte hat andere Stärken und braucht deshalb unterschiedlich lange für die Aufgaben. Allerdings soll jedem höchstens eine Aufgabe zugeteilt werden. Es sind 3 Aufgaben auf 5 Personen zu verteilen. Der Zeitaufwand in Minuten ist dabei folgender:

	1	2	3
1	12	15	20
2	17	30	20
3	30	40	50
4	20	15	10
5	10	15	10

- Welche Aufgabenverteilung ist optimal?

Auch dieses Problem lässt sich als Transportproblem modellieren. Dabei entsprechen die Angestellten den Produktionsstätten und die Aufgaben den Empfängern:

Im Vergleich zu obiger Formulierung gibt es allerdings drei Besonderheiten:

- Jeder Angestellte soll höchstens eine Aufgabe erledigen: $a_i = 1 \; \forall$ Angestellten i
- Jede Aufgabe muss nur einmal erledigt werden: $b_j = 1 \; \forall$ Aufgaben j
- Es gibt mehr Angestellte als Aufgaben, also gilt hier $\sum a_i > \sum b_j$ und wir brauchen zwei Dummy-Aufgaben.

Beispiel: **Catering-Problem**

Ein Catering-Unternehmen benötig frische Servietten für Aufträge in den nächsten 5 Tagen. Die benötigten Stückzahlen sind in folgender Tabelle aufgeführt:

Tag 1	Tag 2	Tag 3	Tag 4	Tag 5
110	210	190	120	110

Tabelle 6.1: benötigte Serviettenstückzahlen

Die Servietten können gekauft werden, das Stück kostet 50 Cent. Schmutzige Servietten können außerdem am Abend in die Reinigung gebracht werden. Der 24h-Service der Reinigung kostet 5 Cent pro Serviette, der 48h-Service ist günstiger und kostet nur 3 Cent. Erst am Ende des Tages können die Servietten zum Waschen gebracht werden, deswegen ist eine erneute Benutzung erst 2 bzw. 3 Tage später wieder möglich. Zusammengefasst:

Kaufen:	$0,10 €$
Waschen 24 h:	$0,05 €$
Waschen 48 h:	$0,03 €$

Tabelle 6.2: Preise der Serviettenbeschaffung

- Wie kann das Catering-Unternehmen vorgehen, um möglichst preisgünstig seinen Bedarf an Servietten zu decken?

Auch dieses Problem lässt sich als Transportproblem modellieren. Die Produktionsstätten entsprechen dabei den Serviettenbereitstellungsmöglichkeiten. Gezeichnet werden also die Knoten "gekaufte Servietten", "schmutzige Servietten vom 1.Tag", ..., "schmutzige Servietten vom 5.Tag". Die Anlässe, zu denen die Servietten gebraucht werden, also die Veranstaltungen an Tag 1-5 entsprechen den Empfängern.
Den täglichen Serviettenbedarf und die Stückzahl der abends in die Reinigung gebrachten Servietten kann man an Tabelle 6.1 ablesen und als "Produktionsmenge" bzw "Bedarf" an den Graph schreiben. Bei der Anzahl der gekauften Servietten scheint es keine Beschränkung zu geben. Um diesen Punkt trotzdem ins Modell aufnehmen zu können, beschränken wir die Anzahl der gekauften Servietten auf maximal 740. Das ist die Anzahl der insgesamt benötigten Servietten; mehr Servietten als insgesamt benötigt zu kaufen wäre in jedem Fall unsinnig. Damit die Summe des Bedarfs der Summe der Produktionsmengen entspricht, führen wir einen Dummy-Knoten mit Bedarf 740 ein.
Den entstehenden Graph sieht man in Abbildung 6.3. Die unterschiedlichen Transportkosten werden hierbei durch unterschiedliche Linientypen repräsentiert. Zwischen einigen Knotenpaaren sind gar keine Linien eingezeichnet, da eine Bereitstellung von Servietten auf diesem Wege nicht möglich ist. Die gestrichelte Linien stehen für gekaufte Servietten, also einen Preis von $0,10 €$ pro Serviette. Die grauen Linien repräsentieren die 24h-Reinigung mit einem Stückpreis von $0,05 €$ und die schwarzen Linien die 48h-Reinigung mit Preis $0,03 €$ pro Serviette. Die gepunkteten Linien stehen für an den Dummy-Knoten abgeführte, das heißt nicht verwendete Servietten, die Kosten hierfür sind $0,00 €$.

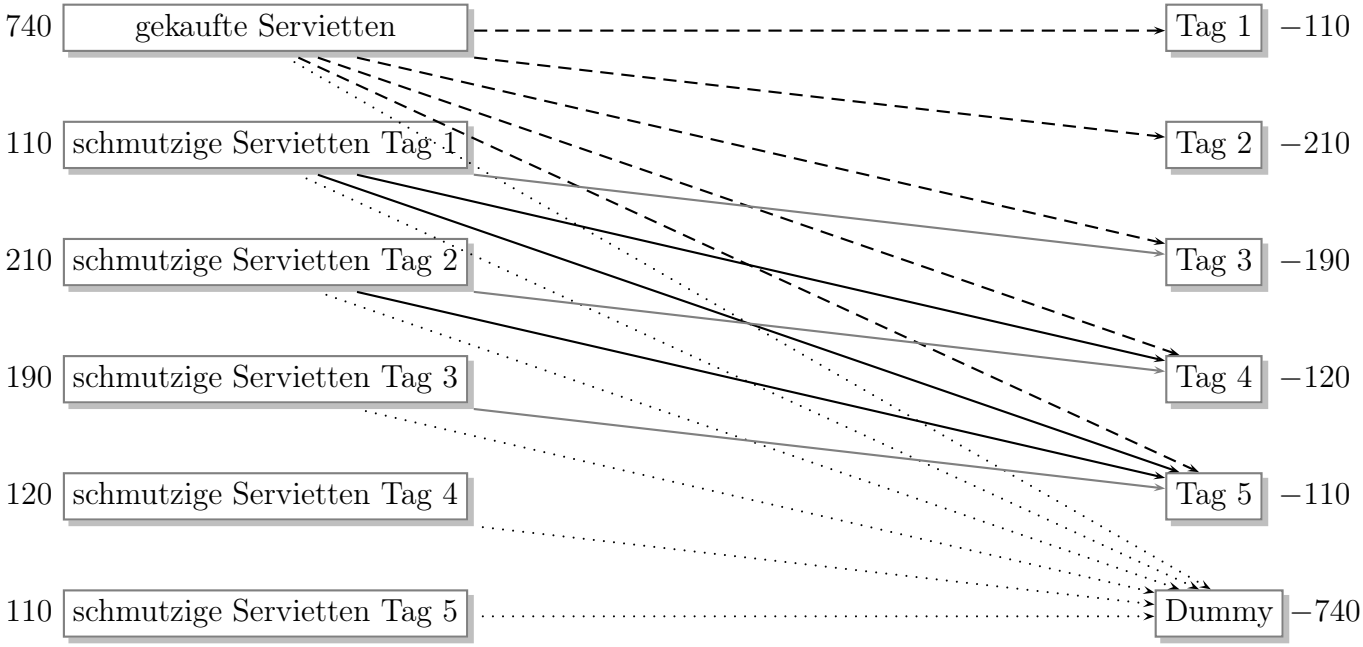


Abbildung 6.3: Graph für das Catering-Problem

6.3 Erstellen einer zulässigen Lösung

Eine zulässige Lösung des Transportproblems ist eine Zuordnung von Transportmengen zu jedem Paar aus Produktionsstätte und Empfänger, so dass der Bedarf des Empfängers gedeckt wird und die maximal in der Produktionsstätte produzierbare Menge nicht überschritten wird. Eine zulässige Lösung lässt sich am besten mit Hilfe des Tableaus in Abbildung 6.4 erstellen, in dem die Transportkosten w_{ij} zwischen jeder Produktionsstätte i und jedem Kunden j, sowie die Menge der produzierten Einheiten a_i und die Höhe des Bedarfs b_j noch einmal aufgeführt sind.

	1	2	3	4	a_i
1	10	0	20	11	15
2	12	7	9	20	25
3	0	14	16	18	5
b_j	5	15	15	10	

Abbildung 6.4: Kostentableau

Dabei wird im zu Produktionsstätte i und Empfänger j gehörigen Feld die Liefermenge von i nach j eingetragen. Ist die Liefermenge von Produktionsstätte i zu Empfänger j 0, wird dieser Eintrag normalerweise weggelassen. Damit der Bedarf von Empfänger j gedeckt ist, muss die Summe der Einträge in jeder Spalte dem b_j-Eintrag in dieser Spalte entsprechen. Ebenso

muss, damit jede Produktionsstätte i die richtige Menge produziert, die Summe der Einträge in jeder Zeile a_i entsprechen. Im Folgenden werden zwei Vorgehensweisen zum Bestimmen einer zulässigen Lösung vorgestellt.

Die Nord-West-Regel

Eine einfache Regel, um schnell eine zulässige Lösung zu erhalten, ist die Nord-West-Regel. Das Ausfüllen der Tableaufelder erfolgt von oben links ("Nord-Westen") nach unten rechts. Da hierbei die Transportkosten nicht beachtet werden, erzeugt die Nord-West-Regel im Allgemeinen Lösungen mit schlechtem Zielfunktionswert.

Algorithmus: Nord-West-Regel

Input: Tableau mit Transportkosten w_{ij}, Menge der produzierten Einheiten a_i für jede Produktionsstätte i , Höhe des Bedarfs b_j für jeden Empfänger j wie in Abbildung 6.4.

Schritt 1. Schreibe in Feld oben links maximal mögliche Liefermenge und streiche volle Zeile/Spalte.

Schritt 2. Erhalte kleineres Tableau. Mach mit kleinerem Tableau in Schritt 1 weiter, bis alle Zeilen und Spalten gestrichen sind.

Output: Zulässige Lösung des Transportproblems.

Das Vorgehen der Nord-West-Regel ist in den Abbildungen 6.5(a)-6.7(b) für das Anfangsbeispiel veranschaulicht.

	1	2	3	4	
1	**5** 10	0	20	11	15
2	12	7	9	20	25
3	0	14	16	18	5
	5	15	15	10	

(a) In das Feld ganz oben rechts wird 5 eingetragen. Ein höherer Wert ist nicht möglich, da $b_1 = 5$. Dann wird die erste Spalte gestrichen, da der Bedarf von Empfänger 1 gedeckt ist.

	1	2	3	4	
1	**5** 10	**10** 0	20	11	15
2	12	7	9	20	25
3	0	14	16	18	5
	5	15	15	10	

(b) In der neuen Tabelle, die durch Streichen der ersten Spalte entsteht, wird ganz oben wieder der maximale Wert, nämlich 10 eingetragen. Ein größerer Wert kann nicht eingetragen werden, da im vorigen Schritt in der selben Zeile schon 5 eingetragen wurden und $a_1 = 15$.

Abbildung 6.5: 1. und 2. Durchgang der Nord-West-Regel für das Anfangsbeispiel

	1	2	3	4	
1	**5** 10	**10** 0	20	11	15
2	12	**5** 7	9	20	25
3	0	14	16	18	5
	5	15	15	10	

	1	2	3	4	
1	**5** 10	**10** 0	20	11	15
2	12	**5** 7	**15** 9	20	25
3	0	14	16	18	5
	5	15	15	10	

Abbildung 6.6: 3. und 4. Durchgang der Nord-West-Regel für das Anfangsbeispiel

	1	2	3	4	
1	**5** 10	**10** 0	20	11	15
2	12	**5** 7	**15** 9	**5** 20	25
3	0	14	16	18	5
	5	15	15	10	

	1	2	3	4	
1	**5** 10	**10** 0	20	11	15
2	12	**5** 7	**15** 9	**5** 20	25
3	0	14	16	**5** 18	5
	5	15	15	10	

Abbildung 6.7: 5. und 6. Durchgang der Nord-West-Regel für das Anfangsbeispiel. Die Kosten dieser Lösung berechnet man als $5 \cdot w_{11} + 10 \cdot w_{12} + 5 \cdot w_{22} + 15 \cdot w_{23} + 5 \cdot w_{24} + 5 \cdot w_{34} = 410$.

Approximationsregel von Vogel

Im Folgenden wird eine weitere Regel zur Berechnung einer zulässigen Lösung vorgestellt: die Approximationsregel von Vogel. Zum Eintragen der Liefermengen wird berechnet, wieviel zusätzliche Kosten entstehen, wenn ein Empfänger nicht von der für ihn günstigsten Produktionsstätte beliefert wird, sondern nur von der zweitgünstigsten. Durch diese zusätzliche Berechnung ist die Approximationsmethode von Vogel aufwändiger als die Nord-West-Regel, liefert aber meistens auch wesentlich bessere Ergebnisse, oft sogar die Optimallösung.

Algorithmus: Approximationsregel von Vogel

Input: Tableau mit Transportkosten w_{ij}, Menge der produzierten Einheiten a_i für jede Produktionsstätte i , Höhe des Bedarfs b_j für jeden Empfänger j.

Schritt 1. Berechne für jede Zeile und jede Spalte die Strafterme:
s_i = zweitniedrigste Kosten in Zeile i - niedrigste Kosten in Zeile i
t_j =zweitniedrigste Kosten in Spalte j - niedrigste Kosten in Spalte j

Schritt 2. Bestimme Zeile oder Spalte mit höchstem Strafterm.

Schritt 3. Wähle dort das Kästchen mit geringsten Kosten und trage die maximale Liefermenge ein.

Schritt 4. Streiche die volle Zeile oder Spalte.

Schritt 5. Passe den Bedarf der noch nicht vollen Spalte oder Zeile an.

Schritt 6. Falls noch nicht alle Zeilen und Spalten gestrichen: Weiter bei Schritt 1.

Output: Zulässige Lösung des Transportproblems.

Das Vorgehen ist in den Abbildungen 6.8(a)-6.11 veranschaulicht.

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	10
2	12	7	9	20	25	2
3	0	14	16	18	5	**14**
b_j	5	15	15	10		
s_j	10	7	7	7		

(a) Schritt 1: Berechnen der Strafterme durch Bilden der Differenz von niedrigsten und zweitniedrigsten Kosten in der entsprechenden Zeile/Spalte. Schritt 2: 14 ist der höchste Strafterm, wähle also Zeile 3.

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	10
2	12	7	9	20	25	2
3	**5** 0	14	16	18	5	14
b_j	5	15	15	10		
s_j	10	7	7	7		

(b) Schritt 3: In der 3. Zeile ist das erste Kästchen das mit den geringsten Kosten. Trage dort die maximale Liefermenge $\min\{b_1, a_3\} = 5$ ein. Schritt 4: Streiche Zeile 3 und Spalte 1, weil $a_3 = b_1 = 5$.

Abbildung 6.8: Approximationsregel von Vogel für das Anfangsbeispiel - Erster Durchgang

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	11
2	12	7	9	20	25	2
3	5 0	14	16	18		
b_j		15	15	10		
s_j		7	**11**	9		

(a) Schritt 1: Berechne in der neuen, durch Streichen entstandenen Tabelle die Strafterme. Schritt 2: Der höchste Strafterm ist $t_3 = 11$, wähle also die dritte Spalte.

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	11
2	12	7	**15** 9	20	10	2
3	5 0	14	16	18		
b_j		15	15	10		
s_j		7	11	9		

(b) Schritt 3: Wähle das Kästchen mit Kosten 9 und trage dort die maximale Liefermenge 15 ein. Schritt 4: Streiche die dritte Spalte. Schritt 5: Setze den Bedarf in Zeile 2 auf $25 - 15 = 10$.

Abbildung 6.9: Approximationsregel von Vogel für das Anfangsbeispiel - Zweiter Durchgang

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	11
2	12	7	15 9	20	10	**13**
3	5 0	14	16	18		
b_j		15		10		
s_j		7		9		

(a) Schritt 1: Berechne in der neuen, durch Streichen entstandenen Tabelle die Strafterme. Schritt 2: Der höchste Strafterm ist $s_2 = 13$, wähle also die zweite Zeile.

	1	2	3	4	a_i	s_i
1	10	0	20	11	15	11
2	12	**10** 7	15 9	20	10	13
3	5 0	14	16	18		
b_j		5		10		
s_j		0		0		

(b) Schritt 3: Wähle das Kästchen mit Kosten 7 und trage dort die maximale Liefermenge 10 ein. Schritt 4: Streiche die zweite Zeile. Schritt 5: Setze den Bedarf in Spalte 2 auf $15 - 10 = 5$.

Abbildung 6.10: Approximationsregel von Vogel für das Anfangsbeispiel - Dritter Durchgang

	1	2	3	4	a_i	Strafe
1	10	**5** 0	20	**10** 11		
2	12	10 7	15 9	20		
3	5 0	14	16	18		
b_j						
Strafe						

Abbildung 6.11: Approximationsregel von Vogel für das Anfangsbeispiel - Trage in die letzten verbleibenden Felder die fehlenden Liefermenge ein

Für diese Lösung betragen die Kosten $5 \cdot 0 + 10 \cdot 11 + 10 \cdot 7 + 15 \cdot 9 = 315$. Wie wir später sehen werden, ist das die Optimallösung für dieses Beispiel. Leider trifft das nicht immer zu, wenn man die Approximationsmethode von Vogel anwendet, sie liefert aber ziemlich gute Lösungen, ist also eine gute Heuristik für das Transportproblem.

6.4 Überprüfung der Optimalität

Wie können wir nun überprüfen, ob eine gefundene Lösung des Transportproblems optimal ist? Dazu teilen wir für die Felder, in denen Liefermengen $\neq 0$ eingetragen wurden, die Transportkosten w_{ij} auf die Produktionsstätte i und den Empfänger j auf, so dass für diese Felder gilt:

$$w_{ij} = u_i + v_j.$$

Dabei kann ein u_i oder v_j frei gewählt werden. Normalerweise setzt man $u_1 = 0$. In Abbildung 6.12 ist diese Aufteilung für die mit Hilfe der Nord-West-Regel aufgestellte zulässige Lösung des Anfangsproblems veranschaulicht.

	v_j	10		0		2		13		
u_i		1		2		3		4		a_i
0	1	5	10	10	0		20		11	15
7	2		12	5	7	15	9	5	20	25
5	3		0		14		16	5	18	5
	b_j	5		15		15		10		

Abbildung 6.12: Berechnung von u_i = Transportkosten von Produktionsstätte i weg und v_j = Transportkosten zu Empfänger j hin für die Lösung nach der Nord-West-Regel

Nun fragen wir uns: Ist es unter Voraussetzung dieser Transportkosten günstiger, eine andere Kombination aus Produktionsstätten und Empfängern zu wählen?
Das heißt wir berechnen die Differenz zwischen den Kosten $c_{ij} = u_i + v_j$ die entstehen würden, wenn wir unter der gerade berechneten Kostenaufteilung in u_i und v_j von Produktionsstätte i zu Empfänger j liefern würden, und den tatsächlichen Transportkosten von Produktionsstätte i zu Empfänger j w_{ij}.
Die Kostenersparnis pro Einheit für die Lieferung von Produnktionsstätte i zu Empfänger j wäre also

$$k_{ij} = c_{ij} - w_{ij} = u_i + v_j - w_{ij}.$$

Diese Kostenersparnisse sind für die Lösung nach der Nord-West-Regel in Abbildung 6.4 in den Feldern unten links dargestellt. Für Kombinationen aus Produktionsstätte i und Empfänger j, bei denen Liefermengen $\neq 0$ eingetragen sind, gilt immer $c_{ij} = w_{ij}$, also ist die Kostenersparnis $k_{ij} = 0$ und wird nicht eingetragen.

	v_j	10		0		2		13		
u_i		1		2		3		4		a_i
0	1	5	10	10	0		20		11	15
						-18		**2**		
7	2		12	5	7	15	9	5	20	25
		5								
5	3		0		14		16	5	18	5
		15		-9		-9				
	b_j	5		15		15		10		

Abbildung 6.13: Kostenersparnisse $k_{ij} = c_{ij} - w_{ij} = u_i + v_j - w_{ij}$ für die Lösung nach der Nord-West-Regel

Im in Abbildung 6.13 dargestellten Fall ist zum Beispiel $u_1 + v_4 - w_{14} = 2$, das heißt würde man die Bestellmengen so ändern, dass von Produktionsstätte 1 zu Empfänger 4 geliefert wird, würde man seinen Gewinn um 2 pro von Produktionsstätte 1 zu Empfänger 4 gelieferter Einheit vergrößern. Wie man die Bestellmengen am geschicktesten ändert, wird in Abschnitt 6.5 erklärt.

Als weiteres Beispiel wird in die Kostenersparnis bei der durch die Approximationsmethode von Vogel erstellten Lösung betrachtet.
Beim Ausrechnen der Transportkosten u_i von Produktionsstätte i weg und v_j zu Empfänger j hin tritt das in Abbildung 6.14 dargestellte Problem auf: Bei Vorgabe nur eines Wertes, nämlich $u_1 = 0$ lassen sich nicht alle weiteren Transportkosten berechnen.

	v_j	?	0	2	11	
u_i		1	2	3	4	a_i
0	1	10	5 0	20	10 11	15
7	2	12	10 7	15 9	20	25
?	3	5 0	14	16	18	5
	b_j	5	15	15	10	

Abbildung 6.14: Berechnung der Kostenersparnis bei der durch die Approximationsmethode von Vogel erstellten Lösung

In so einem Fall wenden wir folgenden Trick an: Wir setzen einfach eine Liefermenge (hier die von s_2 nach e_1) von 0 auf eine sehr kleine Zahl ϵ und erhöhen die entsprechenden Liefermengen a_2 und b_1. So können wir die restlichen Transportkosten und die Kostenersparnis berechnen. Dieses Vorgehen ist in Abbildung 6.15 dargestellt.
Wie man in Abbildung 6.15(b) sieht ist in diesem Beispiel die Kostenersparnis für alle alternativen Kombinationen aus Produktionsstätte und Empfänger negativ. Das heißt es können durch Verändern der Liefermengen keine Kosten mehr eingespart werden. Wir haben also gezeigt, dass in diesem Fall die Approximationsmethode von Vogel eine Optimallösung geliefert hat wie in Abschnitt 6.3 behauptet.

Das Vorgehen wird nochmal in folgendem Algorithmus zusammengefasst:

Algorithmus: Überprüfung der Optimalität

Input: Zulässige Lösung des Transportproblems, eingetragen in ein Tableau mit Transportkosten w_{ij}, Menge der produzierten Einheiten a_i für jede Produktionsstätte i , Höhe des Bedarfs b_j für jeden Empfänger j.

Schritt 1. Teile die Kosten w_{ij} für die Einträge von Liefermengen $\neq 0$ auf Produktionsstätte i und Empfänger j auf: $w_{ij} = u_i + v_j$.

Schritt 2. Berechne für alle Felder, in denen keine Liefermengen $\neq 0$ eingetragen wurden, die Kostenersparnis $k_{ij} = u_i + v_j - w_{ij}$.

Schritt 3. Ist die Kostenersparnis in jedem Feld nichtpositiv, ist die Lösung optimal. Gibt es einen positiven Eintrag, so ist sie nicht optimal.

Output: Entscheidung, ob die gefundene Lösung des Transportproblems optimal ist.

	v_j	5		0		2		11		
u_i		1		2		3		4		a_i
0	1		10	5	0		20	10	11	15
7	2	ϵ	12	10	7	15	9		20	25+ϵ
0	3	5	0		14		16		18	5
	b_j	5+ϵ		15		15		10		

(a) Füge eine zusätzliche Liefermenge ϵ für die Lieferung von Produktionsstätte 2 zu Empfänger 1 ein und erhöhe die entsprechenden Liefermengen a_2 und b_1.

	v_j	5		0		2		11		
u_i		1		2		3		4		a_i
0	1		10	5	0		20	10	11	15
		-5				-18				
7	2	ϵ	12	10	7	15	9		20	25+ϵ
								-2		
-5	3	5	0		14		16		18	5
				-19		-19		-12		
	b_j	5+ϵ		15		15		10		

(b) Nun kann ganz normal die Kostenersparnis berechnet werden.

Abbildung 6.15: Aufteilen der Transportkosten mit Hilfe einer zusätzlichen Liefermenge ϵ

6.5 Finden einer besseren Lösung

Ist die von uns heuristisch gefundene Lösung nicht optimal, möchten wir eine bessere Lösung finden. In diesem Abschnitt soll ein Verfahren vorgestellt werden, dass uns aus jeder nicht-optimalen Lösung eine bessere Lösung erstellt. Es beruht auf folgender Beobachtung:

Beobachtung:
Die Zulässigkeit einer Lösung des Transportproblems verändert sich nicht, wenn man im Tableau auf geschlossenen Wegen aus senkrechten und waagerechten Teilstücken Liefermengen herumschiebt, solange die eingetragenen Liefermengen nicht kleiner als 0 werden.

In Abbildung 6.16 wird eine Liefermenge von 5 Einheiten verschoben. Dazu wird zunächst eine geschlossener Weg aus senkrechten und waagerechten Teilstücken eingezeichnet. Dann werden die Eckpunkte des Weges abwechselnd mit + und − markiert um anzudeuten, wo die Liefermenge erhöht und wo herabgesetzt werden soll. In jeder Zeile und jeder Spalte stehen gleich viele + wie − Zeichen. Somit ist sichergestellt, dass die Gesamtliefermenge a_i von einer Produktionsstätte i sich nicht verändert und auch die Menge b_j, die ein Empfänger j erhält, gleich bleibt. Nun wird eine Liefermenge von 5 Einheiten verschoben, das heißt auf die Liefermenge in jedem mit + markierten Feld werden 5 Einheiten addiert, von der Liefermenge in jedem mit

− markierten Feld werden 5 Einheiten abgezogen. Da die Liefermenge niemals negativ werden darf, dürfen auf diesem Weg nicht mehr als 5 Einheiten verschoben werden, denn sonst wäre die Liefermenge von Produktionsstätte 1 zu Empfänger 1 negativ. Der kleinste Wert in den mit − markierten Feldern bestimmt also die maximal zu verschiebene Menge.
Ist die in Abbildung 6.16 erstellte neue Lösung nun besser als die mit der Nord-West-Regel erstellte?
Ein Vergleich des neuen Zielfunktionswerts $0 \cdot 10 + 15 \cdot 0 + 0 \cdot 7 + 15 \cdot 9 + 10 \cdot 20 + 5 \cdot 0 + 0 \cdot 18 = 335$ mit dem alten Zielfunktionswert von 410 ergibt eine Verbesserung um 75. Diese kommt dadurch zustande, dass die Liefermenge von Produktionsstätte 3 zu Empfänger 1 um 5 erhöht wurde und so die Kostenersparnis von 15 pro Einheit fünfmal anfiel. Da auf allen anderen Felder, auf denen die Liefermenge geändert wurde, eine Kostenersparnis von 0 eingetragen war, ist die gesamte Kostenersparnis durch verschieben der 5 Einheiten $5 \cdot 15 = 75$ und der Zielfunktionswert verringert sich von 410 um 75 auf 335.
Um zu überprüfen, ob die neue Lösung optimal ist, kann jetzt wieder die Kostenersparnis für jedes Tableaufeld ausgerechnet werden, wie in Abschnitt 6.4 erklärt. Leider ist auch diese Lösung noch nicht optimal, da zum Beispiel im Feld für Produktionsstätte 2 und Empfänger 1 noch eine Kostenersparnis von 5 eingetragen ist, wie man Abbildung 6.17 sieht. Würden wir das eben erklärte Verfahren wiederholen, könnten wir die Lösung also noch verbessern.
Wir fassen unser Vorgehen zum Erstellen einer besseren Lösung noch einmal in einem Algorithmus zusammen:

Algorithmus: Finden einer besseren Lösung

Input: Zulässige aber nicht optimale Lösung des Transportproblems, eingetragen in ein Tableau mit Transportkosten w_{ij}, Menge der produzierten Einheiten a_i für jede Produktionsstätte i , Höhe des Bedarfs b_j für jeden Empfänger j, Kosten u_i und v_j und Kostenersparnis k_{ij}.

Schritt 1. Wähle Feld F mit größter Kostenersparnis.

Schritt 2. Konstruiere einen Weg, so dass F eine Ecke ist und alle weiteren Ecken des Weges eine Liefermenge > 0 haben.

Schritt 3. Markiere die Eckfelder des Weges abwechselnd mit + oder −, Feld F bekommt ein +

Schritt 4. Verschiebe die maximale Liefermenge, so dass die Lieferung nirgendwo negativ wird, entlang des Weges.

Output: Zulässige Lösung des Transportproblems mit besserem Zielfunktionswert als die eingegebene Lösung.

Zu Schritt 2: Bei der Konstruktion des Weges im Tableau sollte man darauf achten, dass in allen Ecken des Weges außer F eine Liefermenge > 0 eingetragen ist, alle Ecken außer F also eine Kostenersparnis von 0 haben. Sonst ist es möglich, dass die durch Einfügen einer positiven Liefermenge in Ecke F entstehende Kostenersparnis durch eine negative Kostenersparnis aufgehoben oder geschmälert wird.

Die Kosten der neuen Lösung kann man dann entweder aus dem Tableau berechnen, oder man erhält sie durch die Formel:
Kosten neue Lösung = Kosten alte Lösung - Liefermenge · Kostenersparnis

Indem wir den Algorithmus zum Finden einer besseren Lösung wiederholt ausführen, erhalten wir schließlich eine Optimallösung. Im Folgenden wird das Verfahren zum Lösen des Transportproblems noch einmal als Algorithmus zusammengestellt:

Algorithmus: Lösen des Transportproblems

Input: Ein Transportproblem

Schritt 1. Erstelle das Tableau für das gegebene Transportproblem mit Transportkosten w_{ij}, Menge der produzierten Einheiten a_i für jede Produktionsstätte i , Höhe des Bedarfs b_j für jeden Empfänger j.

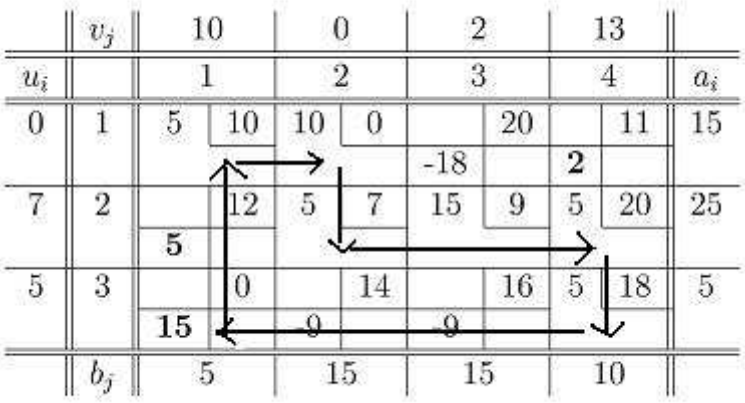
Schritt 2. Finde eine zulässige Lösung.

Schritt 3. Verteile die Kosten auf Produktionsstätten und Empfänger, das heißt berechne u_i und v_j.

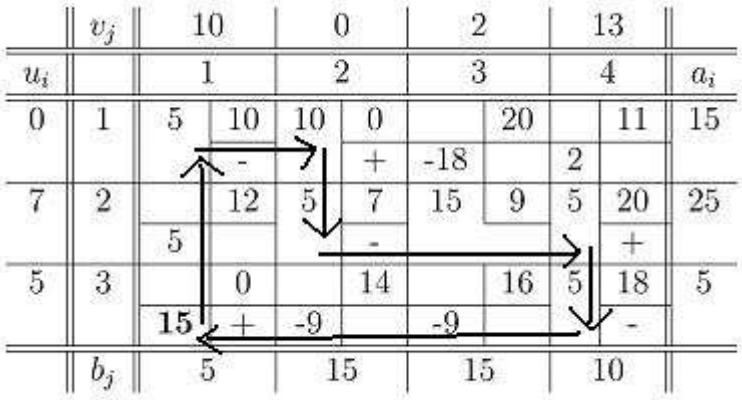
Schritt 4. Berechne für jedes Feld mit Liefermenge 0 die Kostenersparnis $k_{ij} = u_i + v_j - w_{ij}$.

Schritt 5. Gibt es ein Feld mit positiver Kostenersparnis?

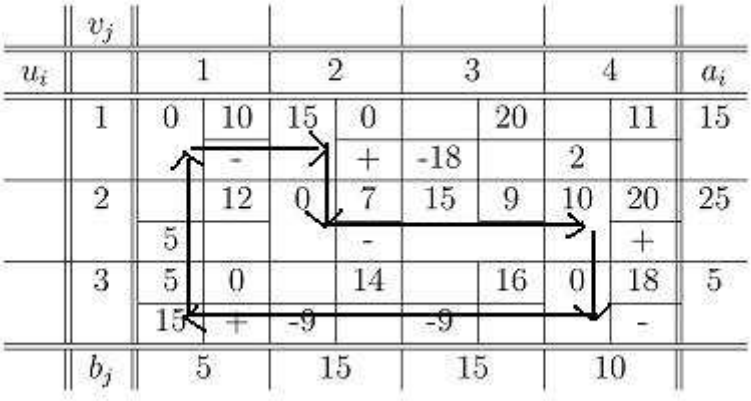
Schritt 6. NEIN → STOPP: Lösung optimal
JA → Wende Algorithmus "Finden einer besseren Lösung" an, dann weiter bei Schritt 3.



(a) Einzeichnen eines geschlossenen Weges



(b) Markieren der Felder mit + und −.



(c) Verschieben der maximal möglichen Menge von 5 Einheiten auf dem eingezeichneten Weg

Abbildung 6.16: Verschieben einer Liefermenge von 5 Einheiten auf einem geschlossenen Weg

	v_j								
u_i		1		2		3		4	a_i
	1		10	15	0		20		11	15
	2		12		7	15	9	10	20	25
	3	5	0		14		16	0	18	5
	b_j	5		15		15		10		

	v_j	10		0		2		13		
u_i		1		2		3		4		a_i
0	1	ϵ	10	15	0		20		11	15+ϵ
						-18		2		
7	2		12	ϵ	7	15	9	10	20	25+ϵ
		5								
-10	3	5	0		14		16		18	5
				-24		-25		-15		
	b_j	5		15		15		10		

Abbildung 6.17: Berechnen der Kostenersparnis für die neue Lösung

Kapitel 7

Netzwerkflussprobleme

7.1 Das klassische Netzwerkflussproblem

Beispiel: Wir betrachten M=3 Produktionsstätten eines speziellen Produkts, das von den N=5 Häusern einer Hotelkette benötigt wird.
Die Produktionsstätten können die folgenden Mengen (in kg) liefern:

Produktionsstätte	P_1	P_2	P_3
lieferbare Menge	100	50	180

Der Bedarf in den fünf Hotels ist der folgende:

Hotels	H_1	H_2	H_3	H_4	H_5
Bedarf	30	80	120	40	60

Der Transport des Produktes erfolgt per Bahn und wird per kg berechnet. Die möglichen Transportwege sind in Abb. 7.1 dargestellt. Dabei sind die Produktionsstätten und die Empfänger als Knoten in dem Transportnetzwerk dargestellt. Die Produktionsstätten haben einen Überschuss an Waren, die Empfänger haben einen Bedarf. Um Produktionsstätten und Empfängern zu unterscheiden, wird der Bedarf der Empfänger als negative Zahl neben die Knoten geschrieben. Der Einfachheit halber nehmen wir an, dass der Transport einer Einheit über eine Kante immer einen Euro kostet. (In der Praxis haben verschiedene Kanten meistens unterschiedliche Kosten).

Die Frage lautet nun: Über welche Wege soll man die in den Produktionsstätten hergestellten Güter zu den Empfängern transportieren, so dass die Summe aller Transportkosten möglichst klein ist?

Zulässige Lösungen des Problems beschreiben also den Warenfluss und werden daher auch als *Fluss* bezeichnet. Zwei solche Flüsse sind in den Abb. 7.2 und 7.3 dargestellt. Die Zahlen an den Kanten geben dabei an, wie viele Einheiten des Produktes über die Kante fließen.
Die Summe der in Abb. 7.2 dargestellten Transportkosten ergibt sich als

$$30 \cdot 1 + 30 \cdot 1 + 60 \cdot 1 + 60 \cdot 1 + 110 \cdot 1 + 190 \cdot 1 + 190 \cdot 1 + 10 \cdot 1 + 120 \cdot 1 + 120 \cdot 1 + 80 \cdot 1 = 1000$$

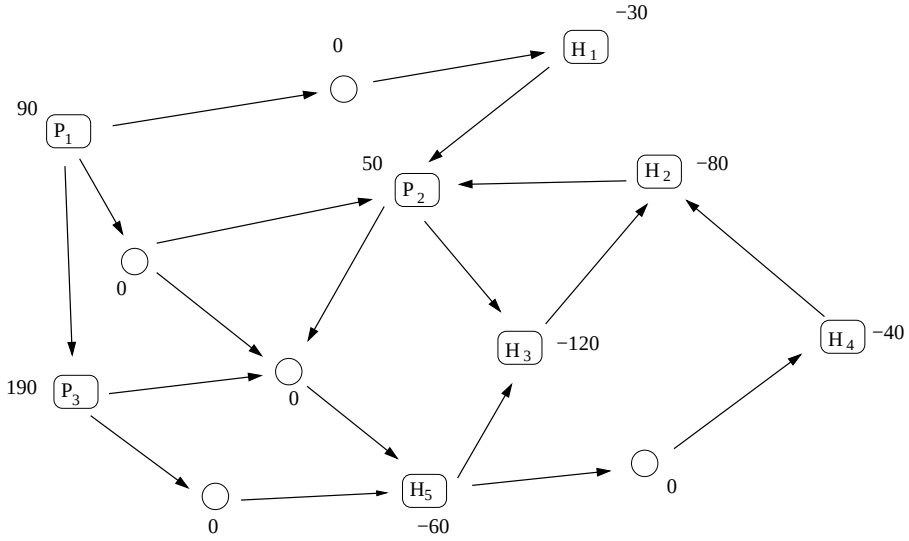


Abbildung 7.1: Das Netzwerkflussproblem aus dem Beispiel.

Die Lösung aus Abb. 7.3 weist nur Transportkosten von

$$30 \cdot 1 + 30 \cdot 1 + 60 \cdot 1 + 60 \cdot 1 + 110 \cdot 1 + 190 \cdot 1 + 190 \cdot 1 + 90 \cdot 1 + 80 \cdot 1 + 40 \cdot 1 + 40 \cdot 1 = 920$$

auf und sollte daher der ersten Lösung vorgezogen werden.

Die zweite gefundene Lösung soll also installiert werden. Dabei stellt sich aber heraus, dass man nicht alle Einheiten wie gewünscht transportieren kann, da auf keiner Strecke mehr als 100 kg gleichzeitig transportiert werden können. Wir müssen also die Waren auf einem anderen Weg transportieren und damit den Fluss verändern, um diese Restriktion zu berücksichtigen. Eine neue Lösung, die diese obere Kapazitätsgrenze nicht überschreitet, ist in Abb. 7.4 zu sehen. Die Kosten dieser Lösung sind aufgrund der zusätzlichen Einschränkung gestiegen und betragen 930.

Formal sind Netzwerkflussprobleme folgendermaßen definiert. Man benötigt dazu zunächst die folgenden Eingangsdaten:

- Einen zusammenhängenden gerichteten Graph $G = (V, E)$ mit n Knoten und m Kanten.
- Werte b_i für jeden Knoten i. Diese Werte bestimmen, was der Knoten hergeben kann oder bekommen möchte. Ist $b_i < 0$, so liegt ein Bedarf vor; der Knoten heißt *Bedarfsknoten* und muss also b_i Einheiten des Produktes erhalten. Für *Vorratsknoten*, die einen Vorrat an dem Produkt ins Netzwerk abgeben können, wählt man $b_i > 0$. Ist $b_i = 0$, so kann der Knoten nichts abgeben und möchte auch nichts bekommen. Solche Knoten nennt man *Durchflussknoten*.
- Die drei verschiedenen Knotenmengen bezeichnen wir wie folgt:
  - $V_{Vorrat} = \{i \in V : b_i > 0\}$ sei die Menge der Vorratsknoten,
  - $V_{Bedarf} = \{i \in V : b_i < 0\}$ sei die Menge der Bedarfsknoten, und

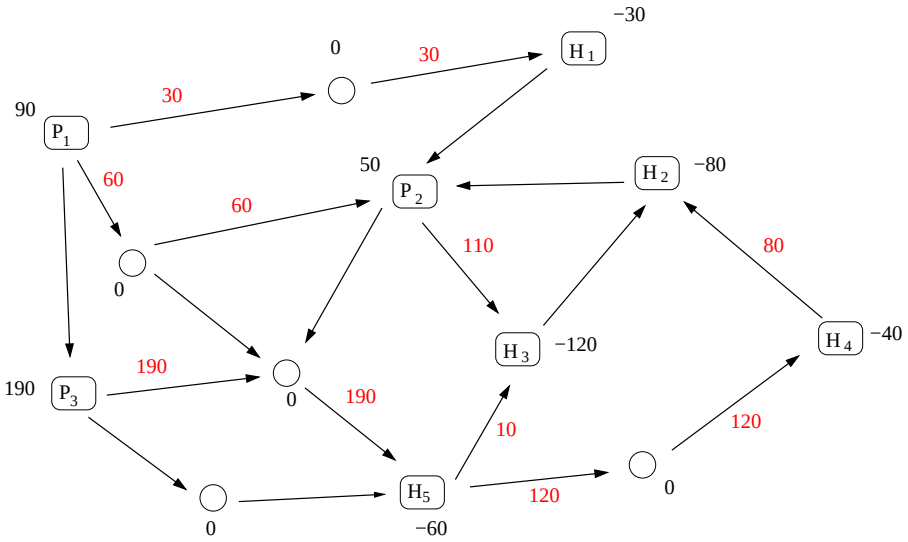


Abbildung 7.2: Eine Lösung des Netzwerkflussproblems ohne Kapazitätsrestriktionen.

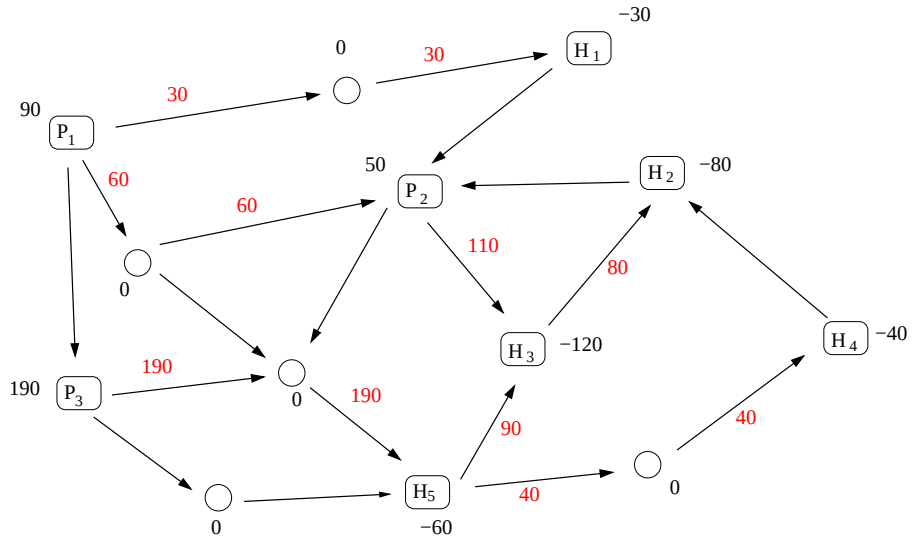


Abbildung 7.3: Eine Lösung des Netzwerkflussproblems ohne Kapazitätsrestriktionen.

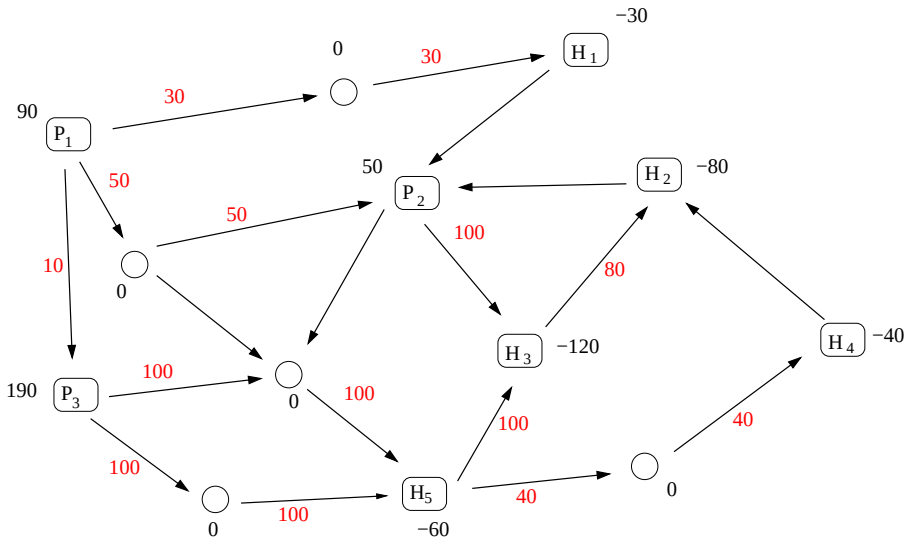


Abbildung 7.4: Eine Lösung des Netzwerkflussproblems, wenn über keine Kante mehr als 100 Einheiten transportiert werden dürfen.

– $V_{Durch} = \{i \in V : b_i = 0\}$ sei die Menge der Durchflussknoten.

Damit das Problem überhaupt lösbar sein kann, muss gelten, dass der Gesamtbedarf gleich dem Gesamtvorrat ist, also

$$\sum_{i \in V_{Vorrat}} b_i = \sum_{i \in V_{Bedarf}} (-b_i).$$

Weil die Durchflussknoten einen Bedarf von Null haben, lässt sich das umformulieren zu der normalerweise verwendeten Bedingung

$$\sum_{i \in V} b_i = 0.$$

- Obere Kapazitäten u_{ij} für alle Kanten $(i, j) \in E$. Die oberen Kapazitäten geben an, wie viel maximal entlang der Kante transportiert werden darf. In dem oben diskutierten Beispiel waren die oberen Kapazitäten alle einheitlich auf 100 gesetzt.

- Kosten c_{ij} für alle Kanten $(i, j) \in E$. Anhand dieser Kostenwerte können die Transportkosten berechnet werden: Werden x_{ij} Einheiten über die Kante (i, j) transportiert, so fallen auf dieser Kante Transportkosten von $x_{ij} \cdot c_{ij}$ an. (In dem oben diskutierten Beispiel hatten wir einheitliche Transportkosten von $c_{ij} = 1$ verwendet.)

Das Netzwerkflussproblem wird dann beschrieben durch $N = (V, E, b, u, c)$.

Das Netzwerkflussproblem besteht nun darin, den überschüssigen Vorrat aus den Vorratsknoten mit möglichst geringen Transportkosten und unter Einhaltung der Kapazitätsbedingungen an die Bedarfsknoten zu verteilen.

Netzwerkflussprobleme können als lineare Programme formuliert und mit ihrer Hilfe effizient gelöst werden. Um das lineare Programm zu entwickeln, gehen wir in den gewohnten Schritten vor.

Variablen: Die Variablen sind die über die Kanten transportierten Mengen, also x_{ij} =Anzahl Einheiten, die über die Kante $(i, j) \in E$ transportiert werden.

Nebenbedingungen: Hier fallen zwei Klassen von Nebenbedingungen an. Die erste Klasse enthält die *Kapazitätsrestriktionen*. Sie stellen sicher, dass die oberen Kapazitätsschranken nicht überschritten werden. Die Kapazitätsrestriktionen sind also

$$x_{ij} \leq u_{ij} \text{ für alle Kanten } (i, j) \in E.$$

Außerdem muss der Warenfluss entlang einer Kante immer positiv sein:

$$x_{ij} \geq 0 \text{ für alle Kanten } (i, j) \in E.$$

Die zweite Klasse von Restriktionen sichert, dass jeder Knoten genau so viel erhält oder abgibt, wie geplant. Die entsprechende Bedingung wird *Flusserhaltungsbedingung* genannt. Wir diskutieren sie getrennt für Vorratsknoten, Bedarfsknoten und Durchflussknoten.

- Für Durchflussknoten muss gelten, dass genau so viele Einheiten in den Knoten hinein fließen wie den Knoten auch verlassen. Dazu hält man einen Knoten i fest und bestimmt
  - alle Kanten, die in den Knoten i hineinführen (also alle j, so dass (j, i) eine Kante ist). Dann summiert man die auf diesen Kanten in den Knoten j gelieferten Waren auf.
  - Analog bestimmt man alle Kanten die aus dem Knoten i herausführen (also alle j, so dass (i, j) eine Kante ist) und summiert auch für diese Kanten die ausgeführten Waren auf.
  - Die beiden Summen müssen gleich sein, damit die Durchflussbedingung erfüllt ist.

  Als Formel erhält man entsprechend:

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = 0 = b_i \text{ für alle Durchflussknoten } i \in V_{Durch}$$

- Für Vorratsknoten geht man analog vor, allerdings darf der Vorratsknoten b_i Einheiten mehr ausführen als er bekommt. Man erhält also

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b_i \text{ für alle Vorratsknoten } i \in V_{Vorrat}$$

  (Spezialfall: Gehen keine weiteren Waren in den Vorratsknoten ein, so vereinfacht sich die Bedingung zu $\sum_{j:(i,j)\in E} x_{ij} = b_i$ und stellt damit sicher, dass genau die richtige Menge den Knoten auch verlässt.)

- Die Bedingung

$$\sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = b_i \text{ für alle Bedarfsknoten } i \in V_{Bedarf}$$

  gilt analog auch für Bedarfsknoten, weil hier das b_i negativ ist. Ein Bedarfsknoten erhält also $-b_i$ Einheiten mehr als er abgibt. (Spezialfall: Verlassen keine Waren den Bedarfsknoten, so vereinfacht sich die Bedingung zu $\sum_{j:(j,i)\in E} x_{ji} = -b_i$ und stellt damit sicher, dass der Knoten die gewünschte Menge erhält.)

Zielfunktion: Als letztes formulieren wir die Zielfunktion. Wir möchten die Summe der Transportkosten über alle Kanten minimieren und erhalten entsprechend

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij}.$$

Es ergibt sich also das folgende Modell:

$$\min \sum_{(i,j)\in E} c_{ij} x_{ij}$$

so dass

$$\begin{aligned} \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} &= b_i \text{ für alle } i \in V \\ 0 \leq x_{ij} &\leq u_{ij} \text{ für alle } (i,j) \in E \end{aligned}$$

Das Programm hat also n Flusserhaltungs-Nebenbedingungen (für jeden Knoten eine) und $2 \cdot m$ Kapazitätsrestriktionen (für jede Kante zwei). Die Anzahl der Variablen beträgt m (für jede Kante der Fluss x_{ij}).

Bemerkung:
Wir können jedes Netzwerkflussproblem so verändern, dass es nur einen Vorratsknoten und nur einen Bedarfsknoten hat. Dazu fügt man einen neuen *virtuellen Vorratsknoten* s ein und verbindet ihn mit allen gegebenen Vorratsknoten. Die Verbindungskanten erhalten Kosten von Null und als Kapazitätsschranken setzt man $u_{si} = b_i$, man wählt also den Vorrat b_i des entsprechenden Vorratsknotens. Als Vorrat für den neuen virtuellen Knoten setzt man die Summe der Vorräte der Vorratsknoten. Der Bedarf der (nicht virtuellen) Vorratsknoten wird schließlich noch auf Null gesetzt.
Analog verfährt man mit den Bedarfsknoten, die ebenfalls mit einem virtuellen Knoten t verbunden werden. In unserem Beispiel erhält man das in Abb. 7.5 dargestellte Netzwerkflussproblem.

Es gibt verschiedene Verfahren, mit denen man Netzwerkflussprobleme effizient (in polynomieller Zeit) lösen kann. Ein oft verwendetes Verfahren ist auch der so genannte Netzwerk-Simplex, der das oben hergeleitete lineare Programm löst. Durch Ausnutzen der zugrunde liegenden Netzwerkstruktur lässt sich das sehr effizient durchführen

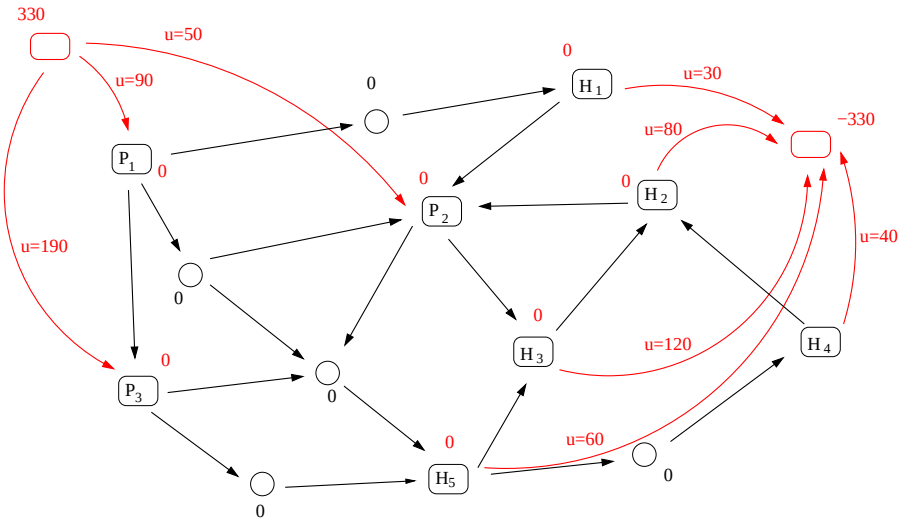


Abbildung 7.5: Das Netzwerkflussproblem aus dem Beispiel.

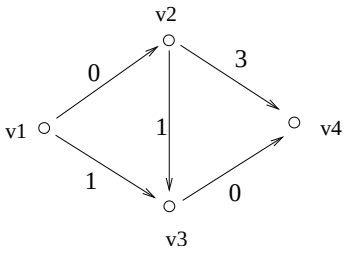


Abbildung 7.6: Ein Beispiel für ein kürzestes-Wege-Problem. Als Flussproblem wählt man $b_{v_1} = 1, b_{v_2} = 0, b_{v_3} = 0$ und $b_{v_4} = -1$.

7.2 Spezialfälle von Netzwerkflussproblemen

Die folgenden bekannten Probleme lassen sich als Netzwerkflussprobleme formulieren und lösen.

Kürzeste Wege Problem

Kürzeste Wege in Graphen haben wir schon in Abschnitt 3.4 behandelt. Wir greifen das Problem hier noch einmal auf.
Gegeben sei ein gerichteter Graph $G = (V, E)$ mit Entfernungen d_{ij} für jede Kante $(i, j) \in E$ und zwei ausgezeichneten Knoten s und t. Die Aufgabe besteht darin, einen möglichst kurzen Weg von s nach t zu finden, also einen Weg P von s nach t mit minimaler Länge.
Kürzeste Wege Probleme können folgendermaßen als Flussprobleme modelliert werden: Definiere das Netzwerkflussproblem $N = (V, E, b, u, c)$ durch den gegebenen Graphen (V, E) und setzte für die Kosten c die gegebenen Kantenlängen d ein, also $c_{ij} := d_{ij}$ für alle Kanten $(i, j) \in E$. Weiterhin setzt man

$$b_s := 1,\ b_t = -1,\quad \text{und } b_i = 0 \text{ für alle } i \notin \{s, t\}.$$

Es soll also genau eine Flusseinheit von s nach t geschickt werden. Schließlich sollen die Kapazitätsbedingungen z.B. durch $u_e = 1$ für alle $e \in E$ vernachlässigt werden. Der sich ergebende Fluss entspricht dann wegen der Flusserhaltung einem Weg, und ein kostenminimaler Fluss entspricht einem kürzesten Weg.

In den kleinen Beispiel aus Abb. 7.6 benötigt man zur Formulierung des Problems als lineares Programm fünf Variablen (für jede Kante eine) $x_{12}, x_{13}, x_{23}, x_{24}$ und x_{34}. Man erhält das folgende lineare Programm zur Bestimmung eines kürzesten Weges in G.

$$
\begin{array}{lll}
\min & 0 \cdot x_{12} + 1 \cdot x_{13} + 1 \cdot x_{23} + 3 \cdot x_{24} + 0 \cdot x_{34} & \\
\text{s.d.} & x_{12} + x_{13} & = 1 \text{ (Flusserhaltung in Knoten } v_1) \\
& -x_{12} + x_{23} + x_{24} & = 0 \text{ (Flusserhaltung in Knoten } v_2) \\
& -x_{13} - x_{23} + x_{34} & = 0 \text{ (Flusserhaltung in Knoten } v_3) \\
& -x_{24} - x_{34} & = -1 \text{ (Flusserhaltung in Knoten } v_4) \\
& 0 \leq x_{ij} \leq 1 & \forall (i,j) \in E \text{ (Kapazitätsrestriktionen)}
\end{array}
$$

Transportproblem

Transportprobleme wurden in Kapitel 6 behandelt. Man betrachtet n Produktionsstätten und m Empfänger. Jeder Produktionsstandort i kann bis zu a_i Einheiten pro Tag produzieren. Jeder Empfänger j benötigt pro Tag b_j Einheiten. Die Produkte können von jedem Standort zu jedem Empfänger transportiert werden, allerdings fallen abhängig von Standort i und Empfänger j unterschiedliche Lieferkosten w_{ij} pro Einheit an.
Die Aufgabe besteht darin, das Produkt möglichst kostengünstig an die Empfänger zu verteilen.

Dieses Problem lässt sich folgendermaßen als Netzwerkflussproblem formulieren: Zunächst benötigt man den Transportgraph $G = (V, E)$. Dazu definiert man einen bipartiten Graph (siehe Seite 22 in Abschnitt 3.2), in dem die "linke" Knotenmenge A den Produktionsstandorten entspricht und die "rechte" Knotenmenge B den Empfängern. Man verbindet dann die Produktionsstandorte mit den Empfängern und erhält dadurch die Kantenmenge

$$E = \{(i,j) : \ i \text{ ist ein Produktionsstandort und } j \text{ ist ein Empfänger}\}.$$

Der eben beschriebene Graph wurde in Kapitel 6 in Abb. 6.1 schon zur Veranschaulichung des Transportproblemes verwendet.
Um die Definition des Netzwerkflussproblemes abzuschließen wählt man als Kosten durch $c_{ij} := w_{ij}$ die im Transportproblem gegebenen Kosten und setzt die oberen Schranken auf $u_{ij} := \infty$. (Statt ∞ reicht es natürlich, die Summe der Waren oder sogar den größten Vorrat zu wählen.)

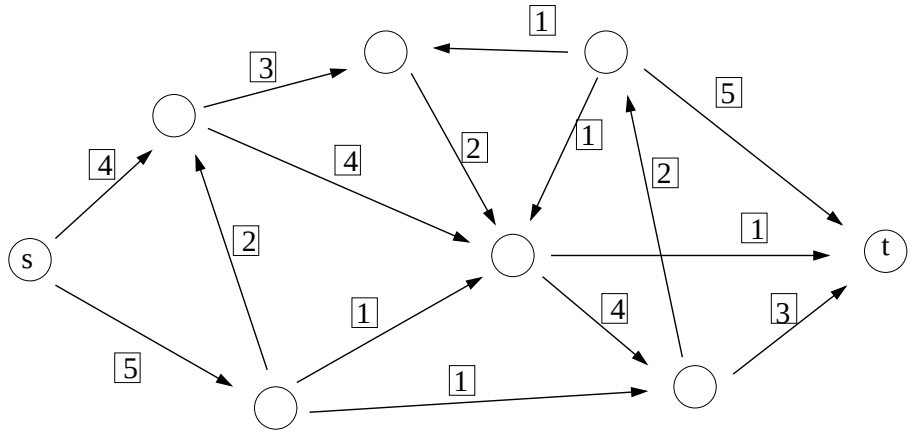


Abbildung 7.7: Ein Netzwerk mit oberen Kapazitäten. Gesucht ist der maximale Fluss von s nach t.

7.3 Maximales Flussproblem

In dem *maximalen Flussproblem* geht es nicht darum, einen gegebenen Bedarf möglichst kostengünstig zu befriedigen, sondern es soll versucht werden, möglichst viel Waren zwischen zwei Knoten zu transportieren.
Gegeben ist wie bisher ein gerichteter Graph $G = (V, E)$ mit Kapazitätsbeschränkungen u_e für alle $e \in E$, allerdings ohne Angaben von Kosten. Zusätzlich seien zwei ausgezeichnete Knoten s und t bekannt.

Die Aufgabe besteht darin, so viel Fluss wie möglich von s nach t zu senden, ohne die Kapazitäten der Kanten zu überschreiten.

Beispiel: Wir betrachten das Netzwerk in Abb. 7.7. Die Zahlen in den Kästchen neben den Kanten bezeichnen die Kapazität der jeweiligen Kante. Es sollen möglichst viele Waren von s nach t geschickt werden. Eine Lösung des Problems ist in Abb. 7.8 angegeben. In dieser Lösung werden sechs Einheiten von s nach t geschickt. Wie wir später sehen werden, ist diese Lösung optimal — mehr Einheiten können nicht verschickt werden.

Auch ein maximales Flussproblem kann man als Netzwerkflussproblem modellieren. Wir definieren Kosten $c_{ij} := 0$ für alle $(i, j) \in E$, und setzen den Bedarf $b_i := 0$ für alle $i \in V$.
Schließlich erweitert man G um eine neue Kante (t, s), für die man definiert:

$$\begin{aligned} u_{ts} &:= \infty \\ c_{ts} &:= -1 \end{aligned}$$

Auch hier setzt man praktisch für ∞ einen echten Wert ein; dieser sollte aber möglichst groß sein (auf jeden Fall größer als der erwartete maximale Fluss).
Die negativen Kosten der neuen Kante kann man als Gewinn interpretieren: Jede Einheit, die über diese Kante fließt, erzeugt einen Gewinn von 1. Das Ziel besteht somit darin, einen möglichst hohen Fluss durch die neue Kante $(t, s) \in E$ zu schicken. Das geht, so lange dieser Fluss auch wieder von s nach t zurück fließen kann, ohne die oberen Kapazitäten zu verletzten

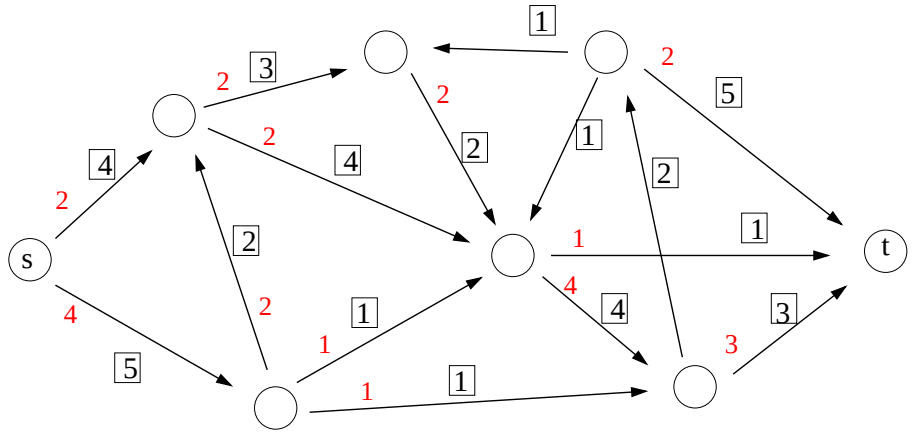


Abbildung 7.8: Der maximale Fluss für das Beispiel.

und löst damit das maximale Flussproblem. Das resultierende Netzwerkflussproblem für das Beispiel ist in Abb.7.9 dargestellt.

Für den maximalen Fluss gibt es eine strukturell schöne Eigenschaft, die zum Abschluss des Kapitels noch dargestellt werden soll. Wir beginnen dazu mit der graphentheoretischen Definition eines *Schnittes*.

Definition 7.1 *Sei $G = (V, E)$ ein gerichteter Graph. Eine Menge $Q \subseteq E$ nennt man einen* **s-t-Schnitt** *falls es eine Menge $X \subseteq V$ gibt, so dass $s \in X$, $t \notin X$ und*

$$Q = (X, V \backslash X) := \{(i, j) \in E : i \in X, j \notin X\}.$$

Die **Kapazität** *des Schnittes ist gegeben durch*

$$C(Q) = \sum_{(i,j) \in Q} u_{ij}.$$

Ein s-t-Schnitt Q heißt **minimal**, *wenn er (unter allen möglichen s-t-Schnitten) $C(Q)$ minimiert.*

Ein s-t-Schnitt Q *trennt* die beiden Knoten s und t in folgendem Sinn: Entfernt man aus G alle Kanten aus Q, so gibt es keinen Weg von s nach t in dem reduzierten Graphen $G' = (V, E \setminus Q)$. Q wird daher auch *trennende Menge* genannt.

Abb. 7.10 zeigt einen Schnitt in dem Beispielsgraphen. Die gefärbten Knoten entsprechen der Menge X. Die Kanten, die von X nach $V \setminus X$ laufen, sind gestrichelt dargestellt. Die Summe ihrer Kapazitäten ergibt die Kapazität des Schnittes, hier also

$$C(Q) = 3 + 4 + 1 + 1 = 9.$$

Es gibt den folgenden Zusammenhang zwischen Schnitten und Flüssen:

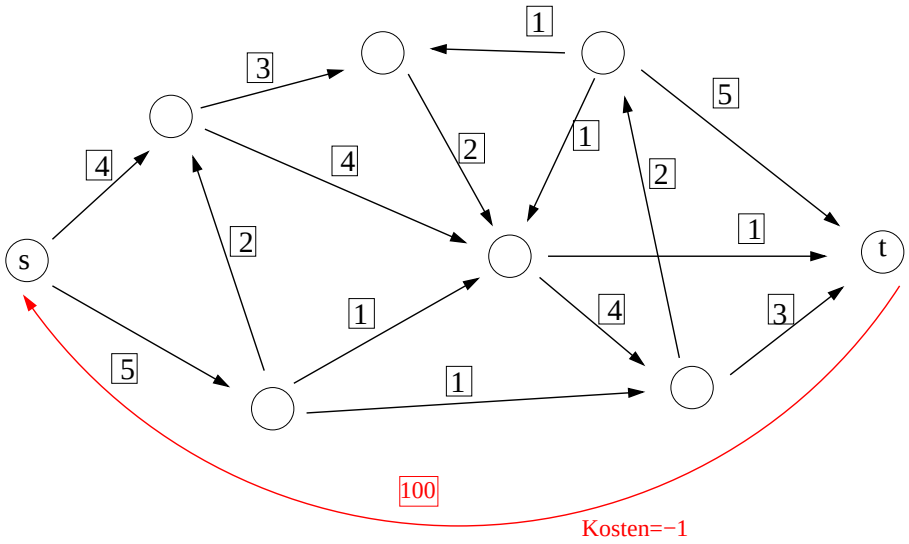


Abbildung 7.9: Modellierung als Flussproblem.

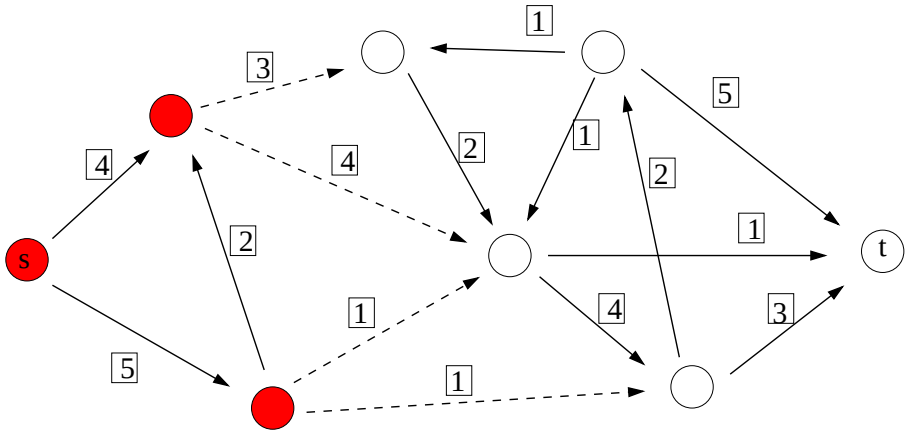


Abbildung 7.10: Ein Schnitt. Die gefärbten Knoten gehören zur Menge X, die gestrichelten Kanten zu dem Schnitt Q. Die Kapazität des Schnittes beträgt 9.

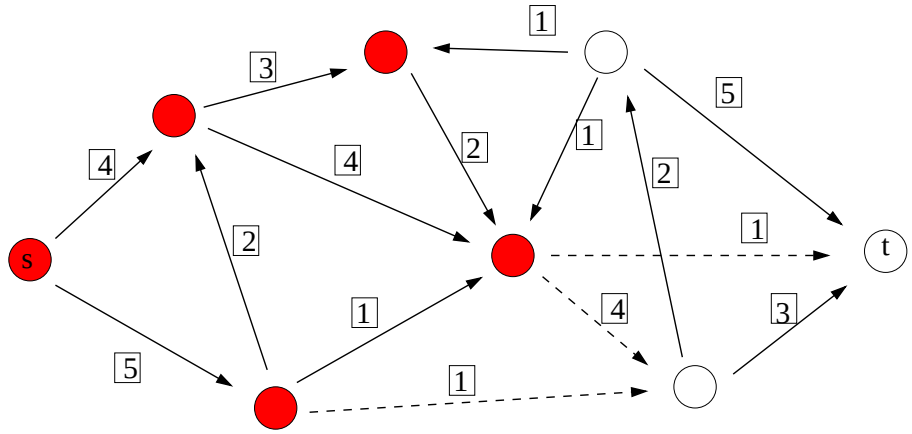


Abbildung 7.11: Ein maximaler Schnitt in dem Graphen. Seine Kapazität beträgt 6.

Satz 7.2 (Ford und Fulkerson: MaxFlow=MinCut) *Ein Fluss von s nach t ist maximal mit Wert v genau dann, wenn es einen minimalen s-t-Schnitt $Q = (X, X \setminus V)$ mit $C(Q) = v$ gibt.*

Für uns bedeutet der Satz das folgende: Ist ein Fluss gefunden, bei dem v Einheiten transportiert werden können, und gelingt es, auch einen Schnitt mit Kapazität v zu finden, so ist der gefundene Fluss maximal. Man kann die Schnitte also verwenden, um zu beweisen, dass ein maximaler Fluss vorliegt. In unserem Beispiel haben wir einen Fluss von $v = 6$ gefunden (siehe Abbildung 7.8). Abb. 7.11 zeigt einen Schnitt mit Kapazität $v = 6$. Also ist der Fluss aus Abb. 7.8 maximal.

Satz 7.2 wurde von Ford und Fulkerson veröffentlicht und ist daher unter ihrem Namen bekannt. Das Resultat wurde aber in einem geheimen Bericht schon vorher von Harris und Ross gezeigt, die für das US-Militär untersuchten, wie man den Transport von Material (maximaler Fluss) von der Sowjetunion nach Osteuropa durch Zerstörung möglichst weniger Brücken (minimaler Schnitt) blockieren kann.

Neben dieser militärischen Anwendung hat Satz 7.2 noch viele weitere praktische Anwendungen. Er hat auch in der Theorie eine wichtige Bedeutung. So gibt es verschiedene Anwendungen in der reinen Mathematik, u.a. folgen aus ihm direkt die Sätze von Menger und Dilworth.

Kapitel 8

Standortprobleme in der Ebene

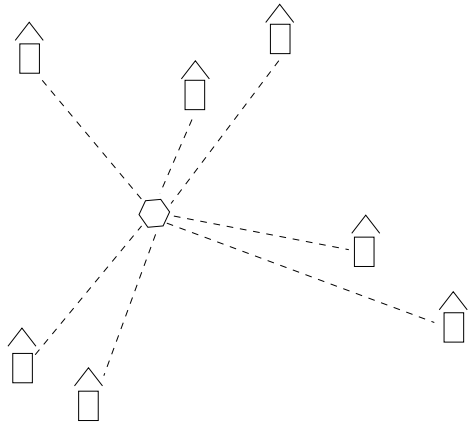
8.1 Klassifizierung von Standortproblemen

Ein Standortproblem ist wie folgt definiert.

- Gegeben sind *existierende Standorte* $\mathcal{A} = \{A_1, \cdots, A_M\}$, die eventuell mit Gewichten $w_m \geq 0$ bezüglich ihrer Bedeutung bewertet sind. Weiter benötigt man eine Entfernungsfunktion, z.B. die Luftlinien- Entfernung.
- Gesucht sind ein oder mehrere neue Standorte. Sie sollen so gewählt werden, dass die Entfernungen zwischen den neuen und den existierenden Standorten möglichst klein sind.

Wir beschreiben zunächst einige (stark vereinfachte) Anwendungen, um zu illustrieren, wo Standortprobleme überall auftreten können. Weitere Anwendungen, Theorie und zahlreiche Verfahren finden sich beispielsweise in den Lehrbüchern von [Ham95, LMW88].

Beispiel: Brunnen in der Wüste



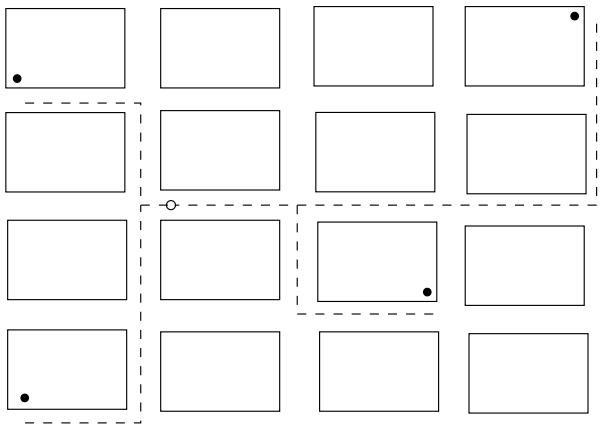
Wo in der Wüste soll ein Anbieter einen Brunnen bauen, so dass er einen möglichst geringen Weg zurücklegen muss, um alle existierenden Standorte mit Wasser zu versorgen?

Das Beispiel *Wüste* wird gerne herangezogen, weil es die Luftlinien-Entfernung (auch *Euklidische Entfernung* genannt) rechtfertigt. Sie wird mit l_2 abgekürzt. Die Euklidische Entfernung zwischen zwei Punkten $A = (a_1, a_2)$ und $B = (b_1, b_2)$ ist definiert als

$$l_2(A, B) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$

Zu minimieren ist in diesem Beispiel die *Summe* der Entfernungen von dem gesuchten neuen Standort zu den existierenden Standorten. Standortprobleme, in denen die Summe der Entfernungen minimiert werden soll, werden *Medianprobleme* oder *Minisum-Probleme* genannt.

Beispiel: Feuerwache in Manhattan
Was ist ein guter Standort für eine Feuerwache oder einen privaten Einbruchschutz in Manhattan?

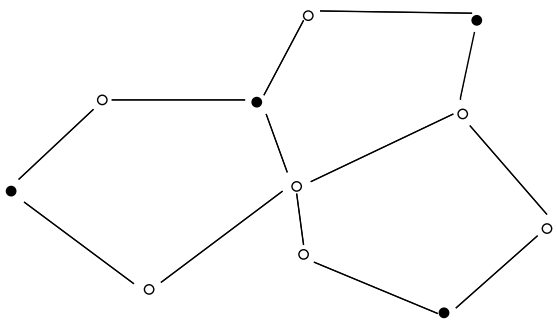


In Manhattan macht es wenig Sinn, Entfernungen mit der Euklidischen Metrik zu messen, sondern es bietet sich die l_1 Entfernung (auch *Manhattan-Entfernung* genannt) an. Die l_1 Entfernung zwischen zwei Punkten $A = (a_1, a_2)$ und $B = (b_1, b_2)$ ist definiert als

$$l_1(A, B) = |b_1 - a_1| + |b_2 - a_2|$$

Eine sinnvolle Zielfunktion in diesem Beispiel bewertet nicht die Summe der Entfernungen von der Feuerwache zu den existierenden Standorten sondern misst die *maximale* Entfernung, d.h. die Entfernung von der Feuerwache zu dem am weitesten entfernt liegenden existierenden Standort. Solche Standortprobleme werden als *Center-Probleme* bezeichnet.

Nicht immer ist der Grundraum für Standortprobleme die Ebene $\mathbb{R}^2$. Beispielsweise treten oft Standortfragen in Netzwerken auf. Eine typische Fragestellung sucht z.B. den besten Platz für ein oder mehrere Warenlager innerhalb eines gegebenen Straßennetzes. Auch hier soll der neue Standort so gewählt werden, dass z.B. die Summe der Entfernungen zu den Kunden möglichst gering ist. Entfernungen werden in diesem Problem mit Hilfe der Netzwerkentfernung entlang der Kanten des gegebenen Netzwerkes gemessen. Auf Netzwerkstandortprobleme werden wir in Kapitel 9 näher eingehen.



Um die Vielfalt von Standortproblemen zu sortieren, wurde von Hamacher und Nickel [HN98] ein Klassifikationsschema eingeführt. Es besteht aus den folgenden fünf Positionen

Neue Standorte / Grundraum / Besonderheiten / Entfernung / Zielfunktion

Als Einträge für die fünf Positionen sind die folgenden Angaben möglich:

- *Neue Standorte* gibt Typ und Anzahl der zu platzierenden Objekte an. Möglichkeiten sind z.B.
  - 1 Punkt,
  - N Punkte,
  - 1 Gerade,
  - ...
- Der *Grundraum* bezeichnet die Menge, in der die neuen Standorte gesucht werden. Das kann der $\mathbb{R}^n$ sein, oder die Ebene $P = \mathbb{R}^2$ (in letzterem Fall spricht man von *planaren* Standortproblemen). Liegt ein Problem in einem Netzwerk vor, so wird ein N eingetragen. Ein D steht für ein *diskretes* Standortproblem: Hier ist eine endliche Menge an potentiellen Standorten vorgegeben, unter denen man die neuen Standorte wählt.
- In die Position *Besonderheiten* wird alles eingetragen, was man als Modellerweiterung mit betrachtet. Eine häufig betrachtete Erweiterung ist es, z.B. *verbotene Gebiete* zuzulassen, die für die Platzierung neuer Standorte nicht zulässig sind (man schreibt dann R). Auch so genannte *Barrieren*, durch die man nicht durchreisen darf, können als Besonderheit vermerkt werden.
- Als Entfernungsmaße sind alle aus Normen abgeleiteten Metriken (z.B. l_2, l_1, l_∞) üblich, aber auch mit anderen Abstandsmaßen werden Standortprobleme diskutiert.

Die drei genannten Standortprobleme kann man folgendermaßen klassifizieren.

Brunnen in der Wüste. $1/\mathbb{R}^n/./l_2/\sum$

Feuerwache in Manhattan. $1/\mathbb{R}^2/R =$Häuser$/l_1/\max$

Warenlager im Straßennetz. $N/N/./d/\sum$

8.2 Medianprobleme mit Manhattan-Entfernung l_1

Beispiel: Gegeben seien $m = 6$ existierenden Standorte (z.B. Geschäfte) innerhalb des Straßennetzes von Manhattan. Die Standorte liegen bei den folgenden Koordinaten (siehe Abb. 8.1):

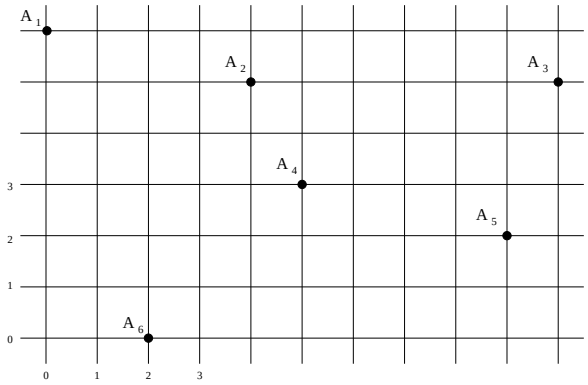


Abbildung 8.1: Das Manhattan-Standortproblem.

A_1	(0,6)
A_2	(4,5)
A_3	(10,5)
A_4	(5,3)
A_5	(9,2)
A_6	(2,0)

Wohin sollte man ein Warenlager bauen, so dass die sechs Läden mit möglichst geringen Transportkosten versorgt werden können? Dabei ist zu beachten, dass man nicht quer durch die Häuserblocks fahren darf, sondern immer auf den rechtwinkeligen Straßen bleiben muss.
Zwei mögliche Vorschläge für das neue Warenlager zusammen mit den dann zurückzulegenden Wegen zu den Geschäften sind in Abb. 8.2 dargestellt. Der Standort im linken Teil der Abbildung liegt an den Koordinaten (3,2). Die Entfernungen von dem Punkt (3,2) zu den einzelnen existierenden Standorten ist in der folgenden Tabelle dargestellt. Z.B. ergibt sich die Entfernung zwischen A_1 =(0,6) und (0,3) als

$$l_1((3,2),(0,6)) = |0-3| + |6-2| = 3+4 = 7.$$

Geschäft	A_1	A_2	A_3	A_4	A_5	A_6
Koordinate	(0,6)	(4,5)	(10,5)	(5,3)	(9,2)	(2,0)
Entfernung zu (3,2)	7	4	10	3	6	3

Die Summe der Entfernungen beträgt also

$$\sum_{i=1}^{6} l_1((3,2),A_i) = 7+4+10+3+6+3 = 33.$$

Im rechten Bild von Abbildung 8.2 ist ein anderer Vorschlag dargestellt, dieses Mal mit den Koordinaten (5,4). Auch hier berechnen wir die Zielfunktion, indem zuerst die einzelnen Entfernungen zu den Geschäften bestimmt werden:

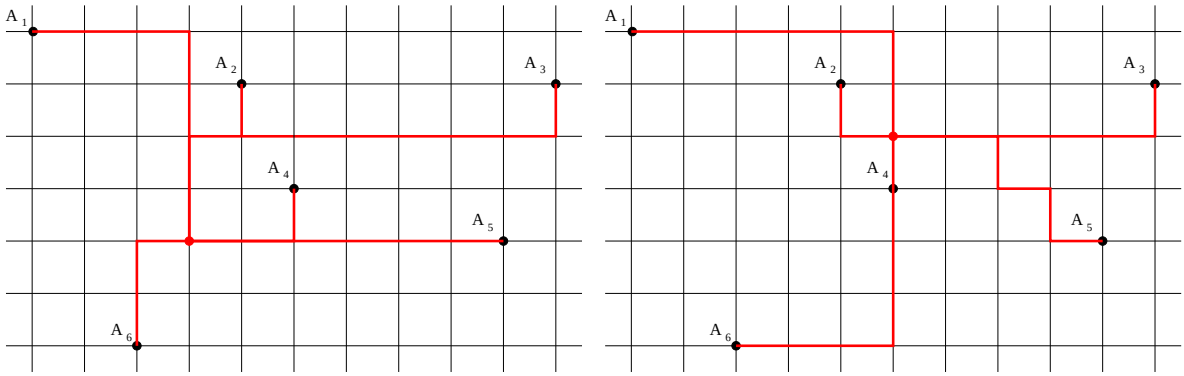


Abbildung 8.2: Zwei Lösungen für das Manhattan-Standortproblem.

Geschäft	A_1	A_2	A_3	A_4	A_5	A_6
Koordinate	(0,6)	(4,5)	(10,5)	(5,3)	(9,2)	(2,0)
Entfernung zu (5,4)	7	2	6	1	6	7

Als Summe ergibt sich 29, das zweite Warenlager scheint also günstiger gelegen zu sein.

Die Klassifizierung des Beispielproblems ist $1/\mathbb{R}^2/\cdot/l_1/\sum$. Um ein Modell für dieses Standortproblem zu formulieren, gehen wir wieder in den in Kapitel 1 beschriebenen drei Schritten vor:

Variablen: Die Variablen sind die beiden Koordinaten des gesuchten neuen Standortes, also z.B. (x_1, x_2).

Nebenbedingungen: Es liegen in diesem einfachen Problem keine Nebenbedingungen vor.

Zielfunktion: Die Zielfunktion minimiert die Summe der Entfernungen von dem neuen Standort zu den Geschäften. Angenommen, die Koordinaten von Standort A_i sind (a_{i1}, a_{i2}), $i = 1, \ldots, m$. Dann erhält man die folgende Zielfunktion:

$$\min f(x) = \sum_{i=1}^{m} l_1(A_i, (x_1, x_2)) = \sum_{i=1}^{m} |a_{i1} - x_1| + |a_{i2} - x_2|$$

Es liegt also ein Optimierungsproblem ohne Nebenbedingungen vor. Um es zu lösen, beobachtet man, dass die beiden Koordinaten unabhängig voneinander gewählt werden können. Man kann die Zielfunktion $f(x)$ nämlich in zwei voneinander unabhängige Teile zerlegen, einen für die x_1-Koordinate und einen für die x_2-Koordinate.

$$\min_{x_1 \in \mathbb{R}} f_1(x_1) = \sum_{i=1}^{m} |a_{i1} - x_1| \quad \text{und} \quad \min_{x_2 \in \mathbb{R}} \sum_{i=1}^{m} |a_{i2} - x_2|.$$

Ein solches eindimensionales Optimierungsproblem

$$\min_{x \in \mathbb{R}} h(x) := \sum_{i=1}^{m} |a_i - x| \tag{8.1}$$

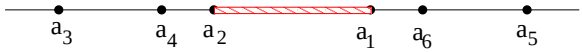


Abbildung 8.3: Eindimensionales Standortproblem.

kann leicht gelöst werden: x kann immer als eines der a_i gewählt werden, genauer: x ist der Median der Werte $a_1, a_2, \ldots, a_m$. Den Median findet man, indem man die Werte der Größe nach sortiert und dann den in der Mitte stehenden Wert wählt. Ist dieser nicht eindeutig, so sind alle Werte zwischen den beiden mittleren Einträgen optimal. Das folgende Beispiel soll das verdeutlichen.

Beispiel:
Seien $a_1 = 7, a_2 = 4, a_3 = 1, a_4 = 3, a_5 = 10, a_6 = 8$. Um den Median zu finden, sortieren wir diese Zahlen der Größe nach und erhalten:

$$1, 3, 4, 7, 8, 10$$

Die Werte in der Mitte sind also die Zahlen an dritter und vierter Stelle, d.h. die 4 und die 7. Das heißt, dass alle Zahlen zwischen 4 und 7 zu der optimalen Lösung von

$$f_1(x) = |1 - x| + |3 - x| + |4 - x| + |7 - x| + |8 - x| + |10 - x|$$

führen. Veranschaulichen kann man sich das auch am Zahlenstrahl wie in Abb. 8.3 dargestellt. Der optimale Bereich liegt zwischen den Standorten a_2 und a_1. Nehmen wir nun einmal an, der neue Standort läge außerhalb dieses Bereiches, z.B. bei a_6. Dann würde er sich bei einer Verschiebung nach links vier Standorten nähern und sich dabei nur von zwei Standorten (a_5 und a_6) entfernen. Man könnte die Gesamtentfernung also durch Verschieben nach links verbessern. Das geht so lange, bis gleich viele Standorte rechts und links liegen: Diese Punkte sind dann optimal.

Das Ergebnis dieses Abschnittes ist also der folgende Satz.

Satz 8.1 *Die Menge der optimalen Standorte für das Standortproblem* $1/\mathbb{R}^2/./l_1/\sum$ *mit existierenden Standorten* $A_i = (a_{i1}, a_{i2})$, $i = 1, \ldots, m$ *ist ein (im allgemeinen degeneriertes) Rechteck der Form*

$$[a_{i1}, a_{j1}] \times [a_{k2}, a_{l2}]$$

mit $[a_{i1}, a_{j1}]$ *ist der Median der ersten Koordinaten* $a_{11}, a_{21}, \ldots a_{m1}$ *der existierenden Standorte und* $[a_{k2}, a_{l2}]$ *der Median der zweiten Koordinaten* $a_{12}, a_{22}, \ldots a_{m2}$ *der existierenden Standorte.*

Angewendet auf das Beispiel aus Abb. 8.1 ergibt sich ein echtes Rechteck, dargestellt in Abb. 8.4. Lässt man den Standort 6 in dem Beispiel weg, so ist der optimale Standort eindeutig auf den Koordinaten (5,5), das Rechteck ist also zu einem Punkt entartet (siehe linker Teil der Abb. 8.5). Ein Beispiel, in dem die Menge der optimalen Lösungen kein Rechteck, sondern eine Strecke ist, erhält man, wenn man Standort 6 auf die Koordinaten (5,0) verschiebt (wie im rechten Teil der Abb. 8.5 dargestellt).

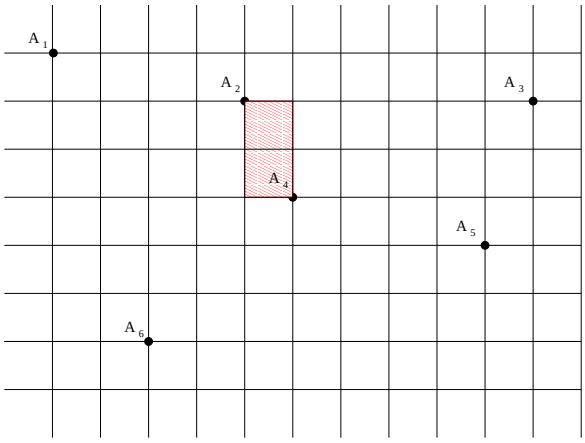


Abbildung 8.4: Die optimalen Lösungen für das Beispiel.

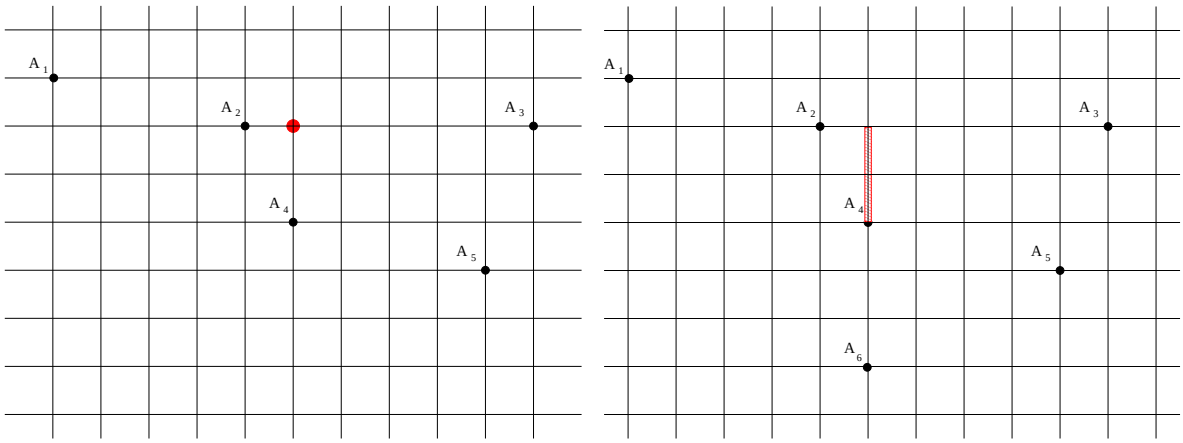


Abbildung 8.5: Die optimalen Lösungen bilden einen Punkt und eine Strecke.

8.3 Medianprobleme mit quadrierter Euklidischer Entfernung

Bevor wir uns mit der Euklidischen Entfernung beschäftigen, betrachten wir zunächst die quadrierte Euklidische Entfernung. Wir suchen also einen Standort, der die Summe der quadrierten Euklidischen Abstände minimiert, d.h. wir betrachten das Problem $1/\mathbb{R}^2/\cdot/l_2^2/\sum$. Seine Zielfunktion lautet:

$$\min_{x\in\mathbb{R}^2} f(x) = \sum_{m=1}^{M} w_m((a_{m1}-x_1)^2 + (a_{m2}-x_2)^2).$$

Wegen

$$f(x) = \sum_{m=1}^{M} w_m(a_{m1}-x_1)^2 + \sum_{m=1}^{M} w_m(a_{m2}-x_2)^2$$

lässt sich dieses Problem wie das Problem mit der Manhattan-Entfernung in zwei voneinander unabhängige, eindimensionale Probleme des Typs

$$\min_{x\in\mathbb{R}} h(x) = \sum_{m=1}^{M} w_m(a_m - x)^2$$

separieren. Die Funktion $h(x)$ ist differenzierbar und konvex. Wir leiten also ab und erhalten

$$h'(x) = -\sum_{m=1}^{M} 2w_m(a_m - x) = 2\sum_{m=1}^{M} w_m(x - a_m).$$

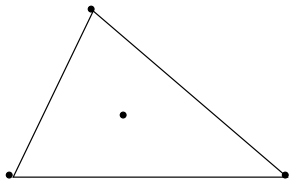
Null setzen der Ableitung ergibt entsprechend einen (eindeutigen) Minimierer:

$$h'(x) = 0 \Leftrightarrow x = \frac{\sum_{m=1}^{M} w_m a_m}{\sum_{m=1}^{M} w_m}.$$

Auch hier fassen wir das Ergebnis als Satz zusammen.

Satz 8.2 $x^* = (\frac{\sum w_m a_{m1}}{\sum w_m}, \frac{\sum w_m a_{m2}}{\sum w_m})$ *ist eindeutige Lösung von* $1/\mathbb{R}^2/./l_2^2/\sum$.

Geometrisch ist die optimale Lösung gerade der Schwerpunkt der gegebenen Standorte. In einem Dreieck entspricht sie also dem gemeinsamen Schnittpunkt der drei Seitenhalbierenden, oder dem Punkt auf dem man das Dreieck auf einer Bleistiftspitze (oder vielleicht besser auf dem Daumen) balancieren könnte.



8.4 Medianprobleme mit Euklidischer Entfernung l_2

Jetzt betrachten wir Standortprobleme mit Euklidischer Entfernung, also Probleme vom Typ $1/\mathbb{R}^2/\cdot/l_2/\sum$. Hier treten gleich mehrere Schwierigkeiten auf. Zunächst ist die sich ergebende Zielfunktion

$$\min_{x\in\mathbb{R}^2} f(x) = \sum_{m=1}^{M} w_m \sqrt{(a_{m1}-x_1)^2 + (a_{m2}-x_2)^2}$$

nicht differenzierbar in den existierenden Standorten, also für alle $a \in \{A_1, \ldots, A_M\}$. Aber auch im differenzierbaren Bereich kann man die Nullstelle der Ableitung nicht analytisch bestimmen oder durch eine Formel angeben. Wir sind daher auf approximative Verfahren angewiesen, um den optimalen Standort zu berechnen. Immerhin hat die Funktion nach wie vor eine schöne Eigenschaft, sie ist nämlich konvex. Diese Eigenschaft sichert, dass die anzuwendenden Approximationsverfahren gegen eine optimale Lösung konvergieren.

Es kann sein, dass einer der existierenden Standorte auch der beste Platz für den neuen Standort ist. Das lässt sich vorab durch folgenden Satz testen.

Satz 8.3 *Die Zielfunktion $f(x)$ von $1/P/\cdot/\sum/l_2$ hat ein Minimum in $x^* = A_m$ für ein $m = 1, \ldots, M$ genau dann, wenn*

$$Test_i = \sqrt{\sum_{m=1,\cdots,M,m\neq i} w_m \frac{a_{i1}-a_{m1}}{l_2(A_i,A_m)} + \sum_{m=1,\cdots,M,m\neq i} w_m \frac{a_{i2}-a_{m2}}{l_2(A_i,A_m)}} \leq w_i$$

Liegt kein Minimum in einem der existierenden Standorte vor, dann verwendet man ein Iterationsverfahren, mit dem man sich beliebig gut an die Lösung annähern kann. Das Verfahren heißt nach seinem Erfinder das *Iterationsverfahren von Weiszfeld*. Zu einem gegebenen Punkt $x^{(l)}$ nach l Schritten berechnet man in Schritt $l+1$ einen neuen Punkt $x^{(l+1)} = (x_1^{(l+1)}, x_2^{(l+1)})$ durch

$$x_k^{(l+1)} = \frac{\sum w_m \frac{a_{mk}}{l_2(A_m,x^{(l)})}}{\sum w_m \frac{1}{l_2(A_m,x^{(l)})}} \quad \text{für } k = 1, 2.$$

Man kann zeigen, dass dieses Verfahren gegen den optimalen Standort konvergiert.

Es soll noch erwähnt werden, dass man für $M = 3$ existierende Standorte die optimale Lösung (für beliebige Gewichte) konstruieren kann. Konstruktive Lösungen sind auch für $M = 4$ existierende Standorte möglich, wenn die Gewichte für alle Standorte gleich sind, also $w_m = 1$ für alle $m = 1, 2, 3, 4$ gilt. Ab fünf existierenden Standorten lässt sich eine Optimallösung dagegen nicht mehr konstruieren.

8.5 Centerprobleme mit Euklidischer Entfernung l_2

Beispiel: Die Bergwacht einer Region in den Alpen sucht einen neuen Standort für eine Hubschrauber-Rettungswacht. Die typischen Unfallstellen, an denen in den letzten Jahren Wanderer verunglückt sind, sind bekannt.

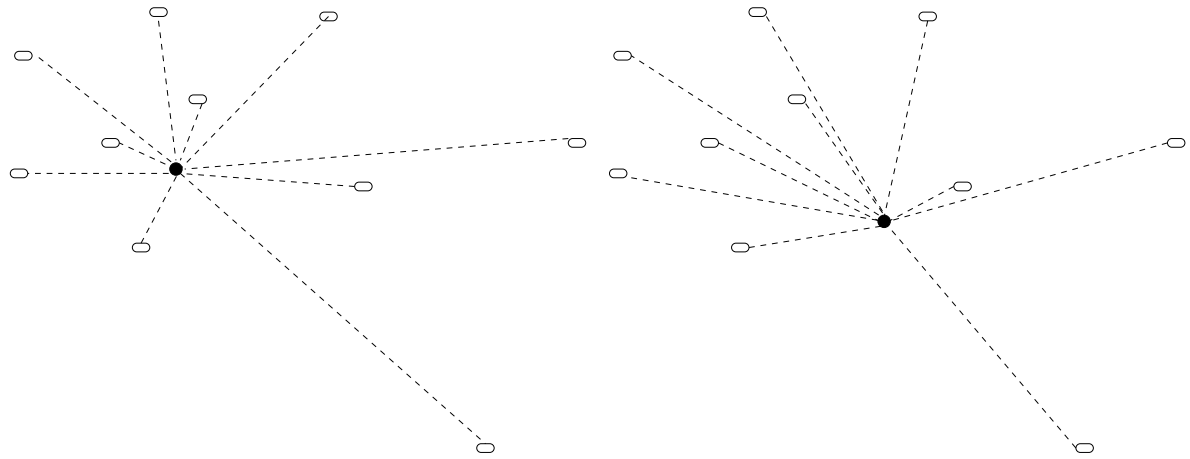


Abbildung 8.6: Zwei mögliche Standorte für eine Hubschrauber-Rettungswache.

Wohin soll die Hubschrauber-Rettungswacht gebaut werden?

Abb. 8.6 zeigt die Gefahrenpunkte und zwei mögliche Standorte für die Rettungswache. Die Entfernungen zu den Gefahrenpunkten sind mit gestrichelten Linien dargestellt. Wir gehen davon aus, dass man keine Umwege um Berggipfel fliegen muss, sondern die Gefahrenpunkte auf der Luftlinienentfernung anfliegen kann.

In diesem Beispiel macht es wenig Sinn, den neuen Standort für die Rettungswacht so zu wählen, dass die Summe der Entfernungen zu den Gefahrenpunkten minimiert wird. Vielmehr ist das Ziel, jeden einzelnen Verunglückten möglichst schnell zu erreichen und medizinisch zu versorgen, bevor es zu spät ist. Bei der im linken Teil von Abb. 8.6 eingezeichneten Lösung sind vor allem die beiden östlichen Gefahrenpunkte nur langsam zu erreichen. Der längste Weg ist bis zu dem Gefahrenpunkt ganz im Südosten zurückzulegen. Diese *maximale* Entfernung möchten wir minimieren.
Eine in diesem Sinne bessere Lösung ist im rechten Teil der Abb. 8.6 dargestellt. Man sieht, dass sich die Entfernung zu dem südöstlichen Gefahrenpunkt und auch zu dem anderen Gefahrenpunkt im Osten deutlich verkürzt hat. Dafür ist der Hubschrauber nun weiter von den westlichen Gefahrenpunkten entfernt. Diese rechts dargestellte Lösung ist tatsächlich die Lösung, die die maximale Entfernung zwischen dem neuen Standort und den Gefahrenpunkten minimiert.

Standortprobleme, bei denen man nicht die Summe der Entfernungen sondern die maximale Entfernung minimieren möchte, nennt man *Center-Probleme*. In unserem Fall betrachten wir die Euklidische Entfernung, so dass man das Problem als $1/\mathbb{R}^2/\cdot/l_2/\max$ klassifizieren kann.

Es gibt eine sehr anschauliche geometrische Interpretation von solchen Center-Problemen mit Euklidischer Entfernung: Anstatt eines Punktes suchen wir den kleinsten Kreis, der alle existierenden Standorte überdeckt. Der Mittelpunkt dieses Kreises ist dann der gesuchte Standort und sein Radius die maximale Entfernung.

Im folgenden betrachten wir dieses Problem, wenn nur zwei oder drei existierende Standorte gegeben sind, vergleiche auch Abb. 8.7.

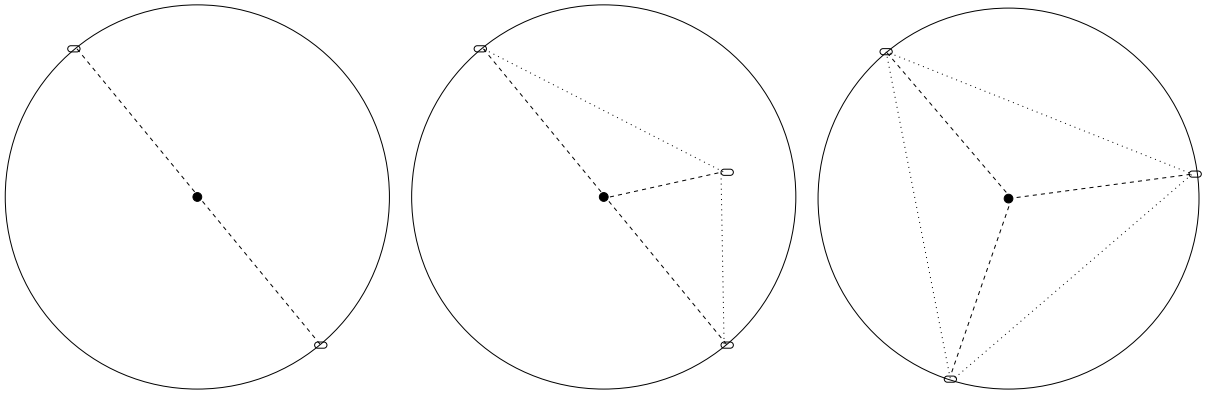


Abbildung 8.7: Der kleinste überdeckende Kreis mit dem optimalen Standort als Mittelpunkt für zwei oder drei existierende Standorte.

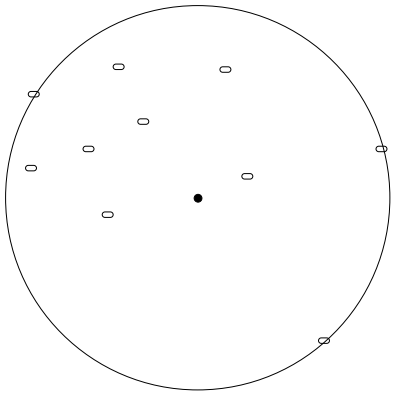


Abbildung 8.8: Der kleinste überdeckende Kreis für das Hubschrauber-Beispiel als Umkreis von drei Gefahrenpunkten.

- Hat man nur zwei existierende Standorte, so liegt die Lösung des Centerproblemes genau in der Mitte zwischen ihnen. Der kleinste überdeckende Kreis hat als Umfang genau die Entfernung zwischen den beiden Standorten.
- Bei drei existierenden Standorten muss man nochmal weiter die folgenden zwei Fälle unterscheiden:
  - Entweder die drei Standorte bilden ein stumpfwinkeliges Dreieck, das also einen ein Winkel größer als 90 Grad enthält. Dann ist der Standort, der zu dieser stumpfwinkligen Ecke gehört, irrelevant und man wählt wieder die Mitte der beiden anderen Punkte.
  - Bilden die drei Standorte dagegen ein spitzwinkeliges Dreieck (es sind also alle drei Winkel kleiner als 90 Grad), dann ist der gesuchte kleinste überdeckende Kreis gerade der Umkreis des Dreiecks. Der Mittelpunkt des Umkreises ist genau der Schnittpunkt der drei Mittelsenkrechten und lässt sich entsprechend schnell finden.

Sind mehr als drei Standorte gegeben, so kann man diese Ideen verallgemeinern: Man überlegt

sich schnell, dass der minimale Kreis immer zwei oder drei der Standorte auf seinem Rand enthalten muss (sonst könnte man ihn durch Verschieben des Mittelpunktes und Anpassen des Radius verkleinern). Man kann also alle Paare und Tripel von Punkten durchprobieren und jeweils einen passenden Kreis bestimmen, von denen man anschließend den besten auswählt. Der kleinste überdeckende Kreis in dem Hubschrauber-Beispiel enthält drei der Gefahrenpunkte: Er ist also der Umkreis zu diesen drei Gefahrenpunkten und lässt sich als Schnittpunkt von zweien ihrer drei Mittelsenkrechten bestimmen (siehe Abb. 8.8).

Geht man bei so einem Verfahren geschickt vor, muss man nicht alle möglichen Kreise durchprobieren. Diese Idee geht auf Elzinga-Hearn zurück und wird daher auch als *Elzinga-Hearn Verfahren* bezeichnet. Daneben sind auch andere Verfahren zum Lösen des Euklidischen Center-Problems möglich.

Kapitel 9

Diskrete Standortplanung

Im Gegensatz zu Kapitel 8, in dem wir Lösungsverfahren für kontinuierliche Standortprobleme unter unterschiedlichen Metriken kennengelernt haben, werden wir uns in diesem Kapitel mit zwei Standortproblemen in Netzwerken (Abschnitt 9.1) und einem diskreten Standortproblem, "Uncapacitated Facility Location" genannt (Abschnitt 9.2), beschäftigen.

9.1 Standortplanung in Netzwerken

9.1.1 Das Median-Problem

Beispiel - Bauen eines Lagers 1:
Es soll ein Lager gebaut werden, von dem aus 6 Städte, die untereinander mit Straßen verbunden sind (siehe Abbildung 9.1), mit Waren beliefert werden. Einige der Staßen sind in beide Richtungen befahrbar, andere sind Einbahnstraßen. Transportkosten fallen pro zur Auslieferung der Waren zurückgelegtem Kilometer an. Das Lager kann in einer der Städte oder außerhalb, an einer Straße liegen.
Wo soll das Lager gebaut werden, um die Transportkosten zu minimieren?

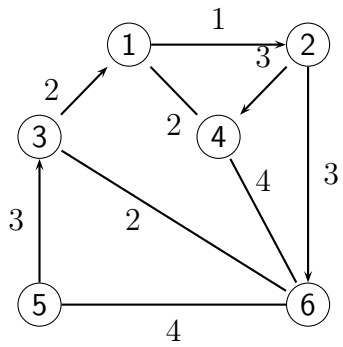


Abbildung 9.1: Das Netzwerk aus dem Beispiel, bestehend aus Städten und Straßen.

Allgemein formuliert stehen wir also vor folgendem Problem:

Für ein Netzwerk $N = (V, E)$ mit Kantenlängen c_e für alle $e \in E$ wird ein Punkt p gesucht (der entweder ein Knoten des Netzwerks ist oder auf einer Kante des Netzwerks liegt), so dass

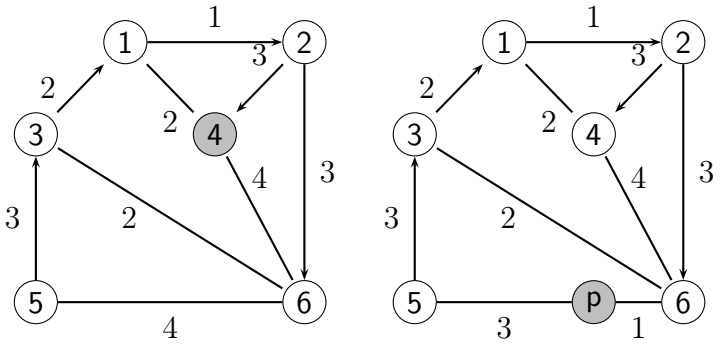
die Summe der kürzesten Wege $kW(v,p)$ von p zu jedem Knoten v minimal ist:

$$\min_{p} \sum_{v \in V} kW(p,v)$$

Benutzen wir das Klassifikationsschema aus Kapitel 8, so ist das Median-Problem mit

$$1/N/./kW(N,V)/\sum$$

zu bezeichnen, wobei $kW(N,V)$ dafür stehen soll, dass wir einen Punkt aus dem Netzwerk N suchen, so dass die Distanz zu den Knoten V minimal ist, wobei die Distanz wischen zwei Knoten als kürzester Weg gemessen wird.
Zwei mögliche Standorte für das Warenlager aus dem Beispiel und die dazugehörigen Zielfunktionswerte findet man in Abbildung 9.2.



(a) Auswahl von Knoten 4 als Standort führt auf einen Zielfunktionswert von 23.

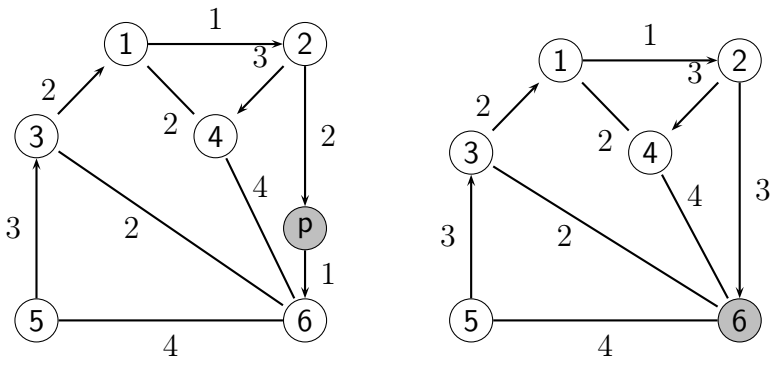
(b) Auswahl von Knoten p, auf der Kante zwischen Stadt 5 und Stadt 6 als Standort führt ebenfalls auf einen Zielfunktionswert von 23.

Abbildung 9.2: Zwei mögliche Standorte für das Warenlager aus Abbildung 9.1.

Glücklicherweise muss man nicht für jeden Punkt im Netzwerk, also für die Knoten und alle Punkte auf den Kanten, den Zielfunktionswert bestimmen, um eine Lösung des Medianproblems zu finden.

- Als erstes beobachtet man, dass Punkte, die auf gerichteten Kanten liegen immer einen höheren Zielfunktionswert haben, als der Knoten am Ende der gerichteten Kante. Dies ist in Abbildung 9.3 veranschaulicht.

- Betrachtet man einen Punkt p auf einer ungerichteten Kante, hat immer einer der Endknoten einen mindestens genauso guten Zielfunktionswert wie der Punkt p. Das kann man folgendermaßen einsehen:
  Angenommen p liegt auf der Kante zwischen v_1 und v_2. Jeder kürzeste Weg $kW(p,v)$ von p in einen Knoten v führt entweder über den Knoten v_1 oder über den Knoten v_2. Verschiebt man p in Richtung von v_1 um l, verlängert sich für einen Knoten v, der vom



(a) Auswahl von Knoten p, auf der Kante zwischen Stadt 2 und Stadt 6 als Standort führt auf einen Zielfunktionswert von 25.

(b) Auswahl von Knoten 6 als Standort führt auf einen Zielfunktionswert von $19 = 25 - 6 \cdot 1$.

Abbildung 9.3: Der Endknoten einer gerichteten Kante hat immer einen besseren Zielfunktionswert als jeder Punkt auf der Kante.

ursprünglichen Knoten p auf kürzestem Wege über v_2 erreicht wird, der Weg über v_2 um l. Die Entfernung von p nach v wächst also höchstens um l. Andererseits verringert sich für jeden Knoten, für den der kürzeste Weg von p über v_1 führt, die Entfernung um l. Führen nun für mehr Knoten die kürzesten Wege über v_1 als über v_2, lohnt es sich also, den Knoten in Richtung v_1 zu verschieben. Je weiter man den Knoten verschiebt, um so mehr verringert sich die Gesamtdistanz, also verschieben wir den Punkt ganz bis ans Ende der Kante und legen ihn auf den Knoten v_1. Also hat dieser einen besseren Zielfunktionswert als der ursprüngliche Punkt p. Nach genau der gleichen Überlegung ist es sinnvoll, p in Richtung v_2 zu verschieben, wenn mehr kürzeste Wege über v_2 nach p führen als über v_1. Wenn es gleich viele Knoten v gibt, so dass $kW(p, v)$ über v_1 beziehungsweise v_2 führt, verschlechtert sich der Zielfunktionswert beim Verschieben von p zumindest nicht. Oft verbessert er sich sogar, weil neue kürzeste Wege über den Knoten, in dessen Richtung verschoben wird, entstehen.
Betrachtet man zum Beispiel den Punkt p in Abbildung 9.2(b) so stellt man fest, dass fast alle kürzesten Wege von p zu den Knoten über den Knoten 6 führen. Verschieben des Knotens in Richtung von Knoten 6 bringt also eine schrittweise Verbesserung vom Zielfunktionswert 23 in Punkt p bis zu Zielfunktionswert 19 in Knoten 6.

Wir stellen fest:

Satz 9.1 *Es gibt immer einen Knoten $v \in V$, der Optimallösung des Median-Problems ist.*

Dieser Satz wird auch Knotensatz genannt. Zum Lösen des Median-Problems müssen wir also nur die Zielfunktionswerte in allen Knoten ausrechnen. Folgenden Algorithmus können wir dabei zur Hilfe nehmen:

Algorithmus: Lösen des Median-Problems

Schritt 1 Berechne die Länge der kürzesten Wege zwischen den Knoten (z.B. mit dem Algorithmus von Floyd-Warshall).

Schritt 2 Summiere für jeden Knoten die Länge der kürzesten Wege.

Schritt 3 Wähle Knoten mit kleinster Summe als Lösung des Median-Problems.

Um die Aufgabe aus dem Beispiel zu lösen, berechnen wir also die kürzesten Wege mit Hilfe des Algorithmus von Floyd-Warshall und fassen sie in einer Matrix zusammen. Der Eintrag in der i-ten Zeile und j-ten Spalte steht dabei für die Länge des kürzesten Weges von Knoten i zu Knoten j. In die rechte, durch einen Strich abgeteilte Spalte schreiben wir die Zielfunktionswerte, also die Summe der Einträge der jeweiligen Zeile:

$$\left(\begin{array}{cccccc|c} 0 & 1 & 6 & 2 & 8 & 4 & 21 \\ 5 & 0 & 5 & 3 & 7 & 3 & 23 \\ 2 & 3 & 0 & 4 & 6 & 2 & 17 \\ 2 & 3 & 6 & 0 & 8 & 4 & 23 \\ 5 & 6 & 3 & 7 & 0 & 4 & 25 \\ 4 & 5 & 2 & 4 & 4 & 0 & 19 \end{array}\right)$$

Wir wählen Stadt 3 als Standort für das Lager, da hier die Summe der Distanzen zu allen anderen Städten mit 17 am geringsten ist.

Beispiel - Bauen eines Lagers 2:
Bisher wurde davon ausgegangen, dass die Lieferkosten vom Lager in die Städte unabhängig von der gelieferten Stückzahl anfallen. Das ist zum Beispiel der Fall, wenn die Produkte so klein und die Stückzahlen so gering sind, dass für den Transport immer nur ein Fahrzeug benutzt wird.
Nun soll ein weiteres Lager gebaut werden, von dem aus sperrigere Güter in die umliegenden Städte transportiert werden. Jeweils 10 Stück passen in einen Lastwagen. Die Anzahl der in den Städten benötigten Güter ist hier tabellarisch zusammengefasst:

Stadt	1	2	3	4	5	6
Liefermenge	60	40	10	20	60	20

Um dieses Problem zu lösen, gewichten wir in der Zielfunktion die Länge der kürzesten Wege mit der Anzahl der benötigten LKWs. Jeder Weg zu Knoten 1 erhält also das Gewicht 6, jeder Weg zu Knoten 2 das Gewicht 4, ... Bezeichnet w_v die Anzahl der LKWs die zu Knoten v fahren müssen ist die Zielfunktion in diesem Fall also:

$$\begin{aligned} &\min_p \sum_{v \in V} w_v \cdot kW(p, v) \\ =&\min_p \left(6 \cdot kW(p,1) + 4 \cdot kW(p,2) + 1 \cdot kW(p,3) + 2 \cdot kW(p,4) + 6 \cdot kW(p,5) + 2 \cdot kW(p,6)\right) \end{aligned}$$

Die Lösung können wir berechnen, indem wir jede Spalte der kürzeste-Wege-Matrix mit dem entsprechenden Gewicht multiplizieren, die Einträge addieren und wieder den Knoten mit kleinster Summe auswählen:

Algorithmus: Lösen des Median-Problems mit unterschiedlichen Liefermengen

Schritt 1 Berechne die Länge der kürzesten Wege zwischen den Knoten (z.B. mit dem Algorithmus von Floyd-Warshall).

Schritt 2 Multipliziere in jeder Matrixspalte j jeden Eintrag mit dem zugehörigen Gewicht w_j.

Schritt 3 Summiere die Einträge jeder Zeile.

Schritt 3 Wähle Knoten mit kleinster Summe als Lösung des Median-Problems mit unterschiedlichen Liefermengen.

Um die Beispielaufgabe zu lösen multiplizieren wir also die Spalten der kürzeste-Wege-Matrix, die wir schon für Beispiel 1 berechnet hatten, mit den Einträgen aus der Tabelle und addieren diese zu den durch einen Strich abgetrennten Einträge in der rechten Matrix auf.

$$\begin{pmatrix} 0\cdot 6 & 1\cdot 4 & 6\cdot 1 & 2\cdot 2 & 8\cdot 6 & 4\cdot 2 \\ 5\cdot 6 & 0\cdot 4 & 5\cdot 1 & 3\cdot 2 & 7\cdot 6 & 3\cdot 2 \\ 2\cdot 6 & 3\cdot 4 & 0\cdot 1 & 4\cdot 2 & 6\cdot 6 & 2\cdot 2 \\ 2\cdot 6 & 3\cdot 4 & 6\cdot 1 & 0\cdot 2 & 8\cdot 6 & 4\cdot 2 \\ 5\cdot 6 & 6\cdot 4 & 3\cdot 1 & 7\cdot 2 & 0\cdot 6 & 4\cdot 2 \\ 4\cdot 6 & 5\cdot 4 & 2\cdot 1 & 4\cdot 2 & 4\cdot 6 & 0\cdot 2 \end{pmatrix} = \left(\begin{array}{cccccc|c} 0 & 4 & 6 & 4 & 48 & 8 & 70 \\ 30 & 0 & 5 & 6 & 42 & 6 & 89 \\ 12 & 12 & 0 & 8 & 36 & 4 & 72 \\ 12 & 12 & 6 & 0 & 48 & 8 & 86 \\ 30 & 24 & 3 & 14 & 0 & 8 & 79 \\ 24 & 20 & 2 & 8 & 24 & 0 & 78 \end{array}\right)$$

Durch die Gewichtung mit den unterschiedlichen Liefermengen ist also nicht mehr Stadt 3 sondern jetzt Stadt 1 mit einem Zielfunktionswert von 70 optimal für das Bauen des Lagers.

9.1.2 Das Center-Problem

Beispiel - Einrichtung einer Feuerwehrwache:
Für die 6 Städte aus dem ersten Beispiel (siehe auch Abbildung 9.4) soll eine Feuerwehrwache eingerichtet werden, von der aus im Notfall alle Städte möglichst schnell erreicht werden können. Die Feuerwehrwache kann in einer Stadt oder an einer der Verbindungsstraßen liegen.

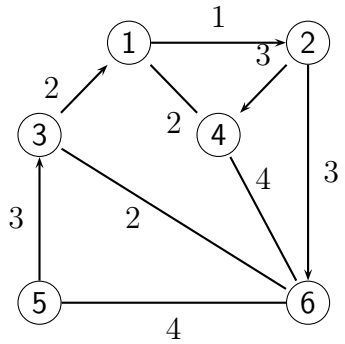


Abbildung 9.4: Das Netzwerk aus dem Beispiel, bestehend aus Städten und Straßen.

Wenn ein Feuer ausbricht, muss die Feuerwehr möglichst schnell zur Stelle sein. Die dabei entstehenden Benzinkosten sind nebensächlich. Ausschlaggebend für die Wahl eines optimalen

Standorts ist hier nicht die Summe der Distanzen sondern die maximale Distanz von der Feuerwehrwache zu einer der Städte.
Allgemein formuliert stehen wir also vor folgendem Problem:

Für ein Netzwerk $N = (V, E)$ mit Kantenlängen c_e für alle $e \in E$ wird ein Punkt p gesucht (der entweder ein Knoten des Netzwerks ist oder auf einer Kante des Netzwerks liegt), so dass der längste der kürzesten Wege $kW(p, v)$ von p zu Knoten v möglichst kurz ist. Die betrachtete Zielfunktion ist also:

$$\min_{p} \max_{v \in V} kW(p, v)$$

Auch dieses Problem lässt sich in das Klassifikationsschema aus Kapitel 8 als

$$1/N/./kW(N, V)/\max$$

einordnen.

Im Gegensatz zum Median-Problem kann beim Center-Problem die Optimallösung auf einer Kante liegen. Dies kann man sich zum Beispiel in Abbildung 9.5 veranschaulichen.

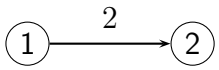


Abbildung 9.5: Der optimale Punkt für eine Feuerwache liegt offensichtlich in der Mitte zwischen Stadt 1 und Stadt 2, da dort die Distanz zu jeder Stadt 1 beträgt. Legt man die Feuerwache in eine der Städte, ist die Distanz zur anderen Stadt 2.

Nehmen wir aber einmal an, die Feuerwache aus obigem Beispiel müsste in einer Stadt platziert werden, um sie für Mitarbeiter leichter erreichbar zu machen ohne das Nahverkehrsnetz der Region anzupassen. Als Optimallösungen kommen in diesem Fall also nur Knoten des Netzwerks in Frage. Mit folgendem Algorithmus lässt sich die Lösung des Center-Problems berechnen, wenn man nur Knoten als Standorte zulässt:

Algorithmus: Lösen des Center-Problems, wenn der optimale Standort nur auf Knoten liegen darf.

Schritt 1 Berechne die Länge der kürzesten Wege zwischen den Knoten (z.B. mit dem Algorithmus von Floyd-Warshall).

Schritt 2 Bestimme für jeden Knoten den Knoten, zu dem der kürzeste Weg maximal ist.

Schritt 3 Wähle Knoten mit kleinstem Wert als Lösung des Center-Problems.

Um die Optimallösung für den Standort der Feuerwache zu bestimmen markieren wir also in jeder Zeile der schon zum Lösen des Median-Problems berechneten kürzeste-Wege-Matrix die Entfernung zum am weitesten entfernten Knoten:

$$\begin{pmatrix} 0 & 1 & 6 & 2 & \mathbf{8} & 4 \\ 5 & 0 & 5 & 3 & \mathbf{7} & 3 \\ 2 & 3 & 0 & 4 & \mathbf{6} & 2 \\ 2 & 3 & 6 & 0 & \mathbf{8} & 4 \\ 5 & 6 & 3 & \mathbf{7} & 0 & 4 \\ 4 & \mathbf{5} & 2 & 4 & 4 & 0 \end{pmatrix}$$

Optimal ist in diesem Fall also Stadt 6 als Standpunkt der Feuerwache mit einem Zielfunktionswert von 5.

Lassen wir auch die Kanten des Netzwerks als potentielle Standorte zu, verkompliziert sich das Lösungsverfahrens zum Finden einer Optimallösung für das Center-Problem.

- Zuerst bemerken wir, dass der optimale Punkt nicht auf einer gerichteten Kante liegen kann, da zu jedem Knoten im Netzwerk vom Endknoten der Kante kürzer ist als von einem beliebigen Punkt p auf der Kante, genau wie beim Medianproblem. Für die Bestimmung des Optimums müssen wir also nur die Netzwerkknoten und Punkte auf ungerichteten Kanten in Betracht ziehen.

- Sei p ein Knoten auf einer ungerichteten Kante $\{v_1, v_2\}$. Dann gilt für jeden Knoten v, dass die Entfernung von p nach v größer ist als das Minimum der Entfernungen von v_1 zu v und v_2 zu v, also

  $$kW(p, v) > \min\{kW(v_1, v), kW(v_2, v)\}.$$

  Da als Zielfunktionswert das Maximum über die Entfernungen zu allen Knoten genommen wird, können wir also ungerichtete Kanten ausschließen, wenn die Entfernung von beiden Endknoten zu einem Knoten v schon größer oder genauso groß wie die Optimallösung des Problems, wenn man nur Ecken als Lösungen zulässt, ist. Im Beispiel können wir also die Kante $\{1, 4\}$ ausschließen, da die Distanz von Knoten 1 zu Knoten 5 und die Distanz von Knoten 4 zu Knoten 5 schon grö ßer ist als der Zielfunktionswert von Knoten 6.
  Für jede Kante $\{v_1, v_2\}$ die wir nicht ausschließen können, gehen wir nun folgendermaßen vor:
  Für jeden Knoten v zeichnen wir die Länge von $kW(p, v)$ als Funktion der Position von p auf der Kante $\{v_1, v_2\}$. Dazu bestimmen wir den Knoten $\tilde{p}$ auf $\{v_1, v_2\}$, von dem v am Weitesten entfernt ist. In ein Koordinatensystem tragen wir nun links den Zielfunktionswert von v_1 und rechts den Zielfunktionswert von v_2 ein. An dem Punkt, dessen Abstand dem Abstand von $\tilde{p}$ zu v_1 und v_2 entspricht, zeichen wir den Zielfunktionswert von $\tilde{p}$ ein. Durch Verbinden dieser Punkte erhalten wir die Funktion, die die Länge des kürzesten Weges für jeden Punkt aus $\{v_1, v_2\}$ nach v angibt. In Abbildung 9.6 ist diese Abbildung für die Distanz der Punkte p auf der Kante $\{3, 6\}$ zu Knoten 4 dargestellt.

Zeichnet man die Funktionen nun für jeden Punkt v in das gleiche Koordinatensystem, wie in Abbildung 9.7(a) für die Kante $\{3, 6\}$ geschehen, kann man für jeden Punkt auf der Kante den Zielfunktionswert als Maximum der Funktionen ablesen. Die Maximumsfunktion $\max_{v \in V} kW(p, v)$

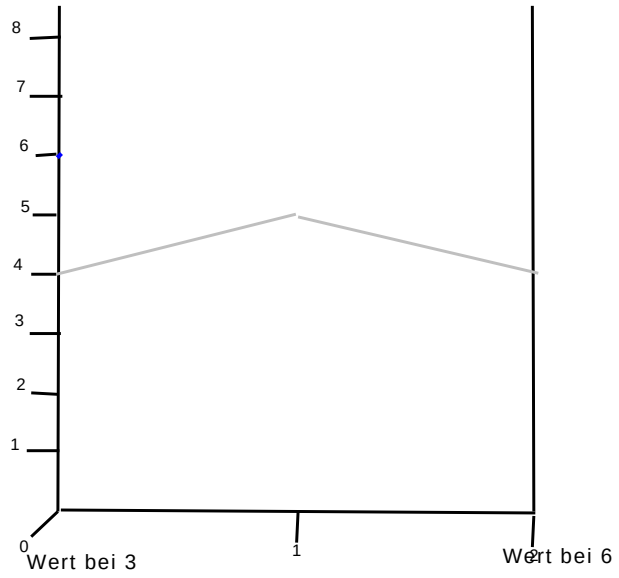


Abbildung 9.6: Die Distanz der Punkte p auf der Kante $\{3, 6\}$ zu Knoten 4. Die x-Achse definiert die Punkte p durch Angabe der Distanz zu Knoten 3.

ist in Abbildung 9.7(b) dargestellt. Den optimalen Zielfunktionswert p_e auf der betrachteten Kante e kann man wie in Abbildung 9.7(b) nun ebenfalls ablesen. Durch Vergleich der Zielfunktionswerte für die optimalen Punkte p_e jeder nicht-ausgeschlossenen gerichteten Kante und des optimalen Zielfunktionswert für die Knoten erhält man die Lösung des Center-Problems. Der folgende Algorithmus fasst das Vorgehen noch einmal zusammen:

Algorithmus: Lösen des Center-Problems

Schritt 1 Bestimme die Lösung p_v des Center-Problems, unter der Annahme, dass der optimale Punkt nur auf einem Knoten liegen darf.

Schritt 2 Überprüfe, welche ungerichteten Kanten betrachtet werden müssen.

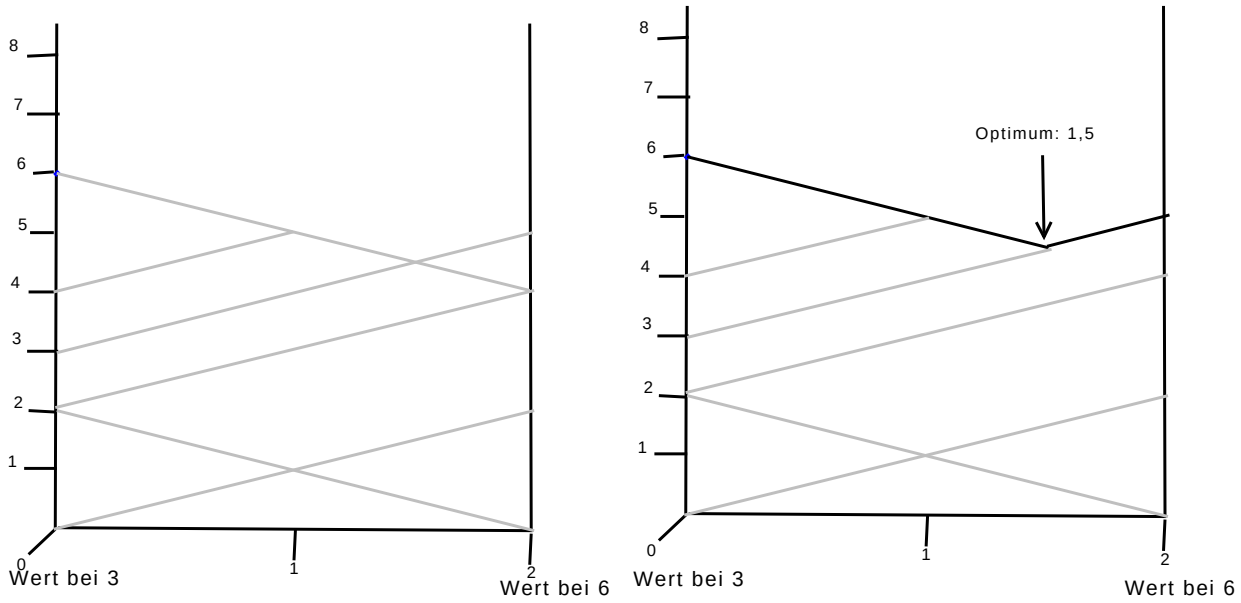
Schritt 3 Finde für jede dieser Kanten e den optimalen Punkt p_e.

Schritt 4 Vergleiche die Zielfunktionswerte von p_v und p_e für alle Kanten e. Wähle Punkt mit kleinstem Zielfunktiosnwert als Optimallösung.

In unserem Beispiel sind die einzigen Kanten, die näher betrachtet werden müssen $\{3, 6\}$ und $\{4, 6\}$. Überprüfung von Kante $\{4, 6\}$ ergibt als optimalen Punkt für diese Kante den Punkt $p_{\{4,6\}}$, der 3 Längeneinheiten von Knoten 4 entfernt ist (und dementsprechend eine Längeneinheit von Knoten 6) mit Zielfunktionswert 6. Optimallösung ist demnach der Punkt $p_{\{3,6\}}$ mit Zielfunktionswert $4, 5$.

9.1.3 Verallgemeinerungen

Neben den vorgestellten Problemen kann man viele weitere Standortprobleme in Netzwerken betrachten. Manchmal sind andere Zielfunktionen als die des Median- und die des Centerpro-



(a) Die Distanzfunktionen von p auf $\{3, 6\}$ für alle Knoten aus dem Beispiel.

(b) Die Maximumsfunktion und der optimale Punkt $p_{\{3,6\}}$ mit Zielfunktionswert $4,5$.

blems sinnvoll. Beispielsweise ist bei der Einrichtung einer Müllkippe ein Ort zu wählen, der von den anderen Standorten im Netzwerk möglichst großen statt minimalen Abstand hat.
Bisher wurden die zu platzierenden Standorte als punktförmig angenommen. Aber auch andere Strukturen können Sinn machen. Beispielsweise könnte in einer Stadt durch Platzierung einer ringförmigen Straßenbahnlinie die Reisezeit zwischen verschiedenen Orten der Stadt verkleinert werden. Fährt die Straßenbahn in kurzen Abständen, könnte für das Benutzen der Straßenbahn eine durchschnittliche Reisezeitersparnis gegenüber zu Fuß gehenden Personen angenommen werden. Ein Optimierungsproblem wäre beispielsweise, die Linie so zu platzieren, dass die durchschnittliche Reisezeit aller Reisenden in der Stadt minimiert wird.
Stehen mehr finanzielle Mittel zur Verfügung, können im Eingangsbeispiel mehrere Lager statt nur ein einziges gebaut werden. Das Gleiche gilt für die Einrichtung mehrerer Feuerwachen.

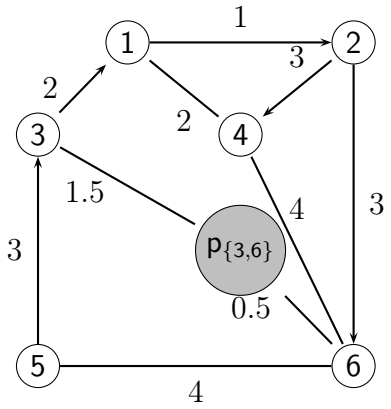


Abbildung 9.7: Das Netzwerk aus dem Beispiel, bestehend aus Städten und Straßen.

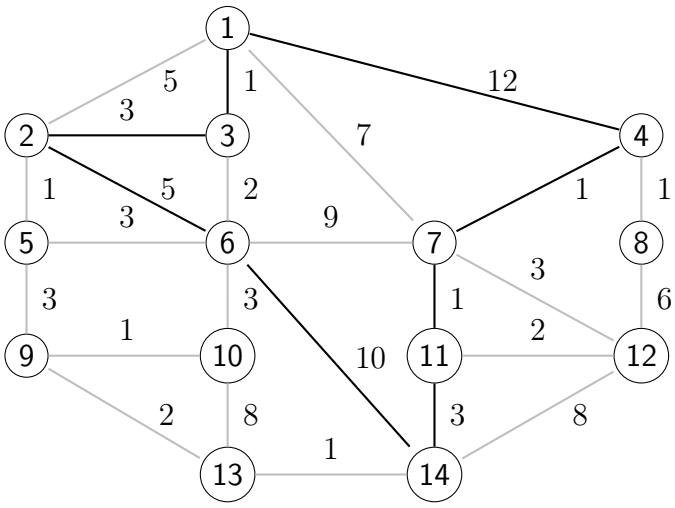


Abbildung 9.8: Platzierung eines kreisförmigen Graph in einem Netzwerk

Das Problem, p Standorte einzurichten, so dass die Summe der Distanzen minimiert wird, nennt man p-Medianproblem. Soll die maximale Distanz minimiert werden spricht man von p-Centerproblemen.
In diesem Zusammenhang soll auch auf Supply-Chain-Management hingewiesen werden, wo Standorte für die verschiedenen Stufen des Herstellungsprozesses, die ein Produkt nacheinander durchläuft, gefunden werden müssen.

9.2 Uncapacitated Facility Location

In diesem Abschnitt betrachten wir ein bekanntes diskretes Standortproblem: "Uncapacitated Facility Location", kurz "UFL" genannt. Im Folgenden wird ein Netzwerkstandortproblem als Beispiel für UFL gegeben. Wie wir später sehen werden, muss aber nicht jedes UFL ein Standortproblem aus einem Netzwerk sein.

Beispiel:
Im Netzwerk aus Abbildung 9.9 sollen mehrere Lagerstandorte platziert werden, aus denen Kunden beliefert werden. Knoten 3 und 6 eignen sich nicht für den Bau von Lagern. Kundennachfrage besteht in Knoten 1, 2, 5 und 6. Kosten entstehen durch das Bauen der Lager und die Transportkosten, die beim Transport vom Lager zum Kunden auftreten. Die Baukosten für die Lager sind $f_1 = 5$, $f_2 = 10$, $f_5 = 6$ und $f_6 = 3$. Die Transportkosten betragen pro Kilometer 1 Euro.

Um dieses Problem als ganzzahliges Programm zu formulieren, berechnen wir zuerst die Transportkosten t_{ij}, die entstehen, wenn man den Kunden j aus Lager i beliefert. Dazu greifen wir

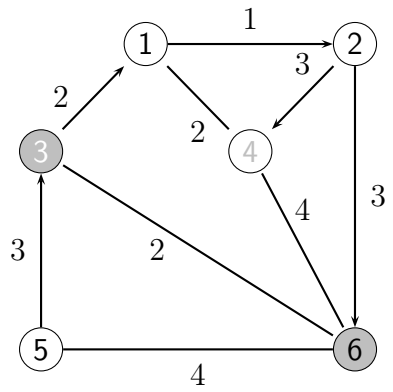


Abbildung 9.9: Das Netzwerk in dem Lager eingerichtet werden sollen. Bedarf besteht in den Knoten mit schwarzer Schrift. In den grau markierten Knoten darf kein Lager gebaut werden.

auf unsere kürzeste-Wege-Matrix

$$\begin{pmatrix} 0 & 1 & 6 & 2 & 8 & 4 \\ 5 & 0 & 5 & 3 & 7 & 3 \\ 2 & 3 & 0 & 4 & 6 & 2 \\ 2 & 3 & 6 & 0 & 8 & 4 \\ 5 & 6 & 3 & 7 & 0 & 4 \\ 4 & 5 & 2 & 4 & 4 & 0 \end{pmatrix}$$

zurück und übertragen die jeweiligen Werte in folgende Tabelle, wobei wir nur die erlaubten Standorte und die Städte mit Bedarf betrachten.

		j			
		1	2	5	6
	1	0	1	8	4
i	2	5	0	7	3
	4	2	3	8	4
	5	6	3	0	4

Allgemein gesehen betrachten wir eine Instanz des "Uncapacitated Facility Location"-Problems. Gegeben sind:

- eine Menge von potentiellen Standorten I,
- eine Menge von Kunden J,
- Transportkosten von den Standorten zu den Kunden t_{ij} und
- Einrichtungskosten für die Standorte f_j

Gesucht ist eine Untermenge $Q \subset I$ von geöffneten Lagern und eine Zuordnung von Kunden zu den Lagern so dass die Summe der entstehenden Kosten minimal ist. Da für eine gegebene

Menge von Lagern Q jeder Kunde j idealerweise aus dem Lager $i \in Q$ bedient wird, für das t_{ij} minimal ist, ist die Zielfunktion also

$$\sum_{i \in Q} f_i + \sum_{j \in J} \min_{i \in Q} t_{ij}. \tag{9.1}$$

Formulierung als ganzzahliges Programm

Im Gegensatz zum Center- und zum Median-Problem, die am Anfang der Vorlesung betrachtet wurden, ist UFL im Allgemeinen NP-schwer, das heißt insbesondere, dass es keinen schnellen Lösungsalgorithmus für UFL gibt. Um unser Problem wenigstens für kleine Instanzen oder approximativ lösen zu können, formulieren wir es als ganzzahliges Programm:

Variablen: Wir definieren Variablen y_i die angeben, ob Standort i geöffnet werden soll:

$$y_i = \begin{cases} 1 & \text{falls Standort } i \text{ geöffnet,} \\ 0 & \text{sonst.} \end{cases} \tag{9.2}$$

Nun können wir Variablen x_{ij} definieren, die angeben, ob Kunde j von Standort i beliefert wird:

$$x_{ij} = \begin{cases} 1 & \text{falls Kunde } j \text{ aus Standort } i \text{ beliefert wird,} \\ 0 & \text{sonst.} \end{cases} \tag{9.3}$$

Eigentlich ist es unwesentlich, ob ein Kunde von einem oder mehreren Standorten beliefert wird, solange sein Bedarf gedeckt ist. Wir können x_{ij} also auch definieren als

$$x_{ij} = \text{ Anteil des Bedarfs von Kunden } j, \text{ der von Lager } i \text{ gedeckt wird.} \tag{9.4}$$

Da es für eine vorgegebene Menge von Standorten Q immer am sinnvollsten ist j aus dem Standort $i \in Q$ zu beliefern, für den t_{ij} minimal ist, wird x_{ij} aber trotz dieser Definition immer binär sein, solange nicht mehrere Standorte $j \in Q$ die gleiche Entfernung zu i haben.

Nebenbedingungen: Ein Kunde kann nur aus einem Lager beliefert werden, falls es geöffnet ist:

$$x_{ij} \leq y_i \text{ für alle Standorte } i \text{ und Kunden } j. \tag{9.5}$$

Der Bedarf jedes Kunden muss gedeckt werden:

$$\sum_{i \in I} x_{ij} = 1 \text{ für alle Kunden } j. \tag{9.6}$$

Die Lager sind entweder geöffnet oder geschlossen, y_i ist also binär:

$$y_i \in \{0, 1\} \text{ für alle Lager } i. \tag{9.7}$$

Definiert man x_{ij} wie in 9.3 so fehlt noch die Nebenbedingung

$$x_{ij} \in \{0,1\} \text{ für alle Standorte } i \text{ und Kunden } j. \tag{9.8}$$

Verzichtet man auf diese eigentlich unnötige Bedingung so muss gelten, dass der Anteil des Bedarfs von Kunden j, der von Standort i gedeckt wird, zwischen 0 und 1 liegt:

$$0 \leq x_{ij} \leq 1 \text{ für alle Standorte } i \text{ und Kunden } j. \tag{9.9}$$

Wegen Bedingung 9.6 reicht stattdessen auch schon

$$0 \leq x_{ij} \text{ für alle Standorte } i \text{ und Kunden } j. \tag{9.10}$$

Zielfunktion: Zu minimieren sind die Kosten, die sich aus den Transportkosten und den Baukosten der Einrichtungen ergeben:

$$\min \sum_{i \in I} f_i y_i + \sum_{i \in i} \sum_{j \in J} t_{ij} x_{ij} \tag{9.11}$$

Es ergibt sich also das folgende Modell:

$$\begin{aligned}
&\min \sum_{i \in I} f_i \cdot y_i + \sum_{i \in i} \sum_{j \in J} t_{ij} \cdot x_{ij} && \\
&\text{s.d. } x_{ij} \leq y_i && \text{für alle Standorte } i \text{ und Kunden } j \\
&\sum_{i \in I} x_{ij} = 1 && \text{für alle Kunden } j \\
&y_i \in \{0,1\} && \text{für alle Standorte } i \\
&0 \leq x_{ij} && \text{für alle Standorte } i \text{ und Kunden } j.
\end{aligned} \tag{9.12}$$

Das bei der Einführung des UFL betrachtete Beispiel war ein Beispiel für die Einrichtung von Standorten in einem Netzwerk. Aber auch andere Probleme, die nichts mit Netzwerken und auf den ersten Blick noch nicht einmal etwas mit Standorten zu tun haben kann man als UFL formulieren. Ein Beispiel haben wir auf Übungsblatt 2 kennengelernt:

Beispiel:
Peter möchte sich einen neuen Computer anschaffen und hat beschlossen, die Einzelteile separat zu bestellen und den Computer dann selbst zusammenzubauen. Nach langer und aufwändiger Internetrecherche hat er 5 Anbieter ausgemacht, die die gesuchten Komponenten (in der gewünschten Qualität) anbieten und folgende Preisliste zusammengestellt:

	Anbieter 1	Anbieter 2	Anbieter 3	Anbieter 4	Anbieter 5
Mainboard	100	120	150	120	110
CPU	190	200	205	190	210
RAM	65	50	45	50	65
Festplatte	105	80	105	100	105
Gehäuse	50	45	50	40	40

Unabhängig von der Größe der Bestellung fallen Lieferkosten an, und zwar 20 Euro bei Bestellung bei Anbieter 1, 2 und 3, 15 Euro bei Anbieter 4 und nur 10 Euro bei Anbieter 5. Peter möchte seinen neuen Computer natürlich möglichst günstig erwerben.

Dieses Problem lässt sich als UFL formulieren, wenn man die Anbieter als einzurichtende Standorte I und die Computereinzelteile als Kunden J ansieht. Die Transportkosten des UFL entsprechen dann den Preisen bei den unterschiedlichen Anbietern, die Baukosten der Standorte den Lieferkosten, die unabhängig von der Bestellmenge anfallen.

Kapitel 10

Optimierungsprobleme in der Telekommunikation (von S. Schwarze)

10.1 Einführende Beispiele

Telekommunikation befasst sich mit der Übertragung von Nachrichten über Distanzen hinweg unter Nutzung nachrichtentechnischer Übermittlungstechnologien. Die Aufgabengebiete der Telekommunikation sind vielfältig und reichen von technologischen Aspekten wie Nachrichtendarstellung und -codierung, Auswahl von Übertragungsprotokollen und -technik hin zu strategisch/planerischen Aufgaben wie Datenrouting und Netzwerkdesign.
In dem vorliegenden Kapitel wenden wir uns der planerischen Seite zu und betrachten im Besonderen algorithmisches Vorgehen zur Bestimmung von Netzwerken und Routing.

Typische planerische Probleme der Telekommunikation ergeben sich sowohl im strategischen Bereich, d.h. in der langfristig orientierten Planung (Netzwerkdesign, Auswahl der Übertragungstechnologie,...) als auch im mittelfristig, taktischen Bereich (Kapazitätenplanung, Hardwareanschaffung,...) und im kurzfristig, operativen Feld (Routing, Wellenlängenzuordnung,...). Typischerweise sind die vielfältigen Problemstellungen eng miteinander verzahnt, d.h., es bestehen starke wechselseitige Abhängigkeiten. So hat beispielsweise die Entscheidung bezüglich einer Netzwerkstruktur starken Einfluss auf das anschließende Datenflussrouting. Umgekehrt können Anforderungen an das Netzwerkdesign aus Routingentscheidungen begründet sein.
Mathematische Modelle dienen der Strukturierung und Analyse von komplexen Problemstrukturen und können zur Entscheidungsunterstützung herangezogen werden. Im folgenden Kapitel wollen wir uns einer Auswahl an Problemen der Telekommunikation widmen und geeignete Methoden der Optimierung zu deren Lösung kennenlernen.

10.2 Graphentheoretische Grundlagen: Bäume

Bevor wir uns in den folgenden Kapiteln einigen Fragen des Netzwerkdesigns und Datenroutings in Telekommunikationsnetzen widmen wollen, führen wir an dieser Stelle in die Grundlagen (graphentheoretischer) Bäume ein. Bäume sind Strukturen, die es erlauben, mehrere Bedarfspunkte (z.B. Telefonkunden) in einem Graphen zu verbinden. Solche Strukturen sind also naturgemäß im Netzwerkdesign und -routing von Interesse. Insbesondere die Kombination mit Kosten- oder

Kapazitätsaspekten machen Baumstrukturen bedeutend für Planungsaufgaben im Telekommunikationsbereich.
Im Folgenden wiederholen wir einige grundlegende Begriffe und führen auch einige neue Begriffe ein. Dabei soll, wenn wir von Graphen sprechen, in der Regel von ungerichteten Graphen die Rede sein, auf Ausnahmen weisen wir gesondert hin. Telekommunikationsprobleme sind häufig in ihrer Natur „ungerichtet“, da Informationskanäle in der Regel in beide Richtungen nutzbar sind.

Definition 10.1 *Sei $G = (V, E)$ ein Graph mit Knoten $v \in V$ und Kanten $e \in E$.*

- *$G' = (V', E')$ ist ein* **Untergraph** *von G, wenn gilt $V' \subseteq V$, $E' \subseteq E$.*
- *In G definieren wir einen* **Weg** *P als eine Folge von Knoten*

$$P = \{v_{i(1)}, v_{i(2)}, \ldots, v_{i(\ell)}\}$$

  mit $e_{i(p)} = \left(v_{i(p)}, v_{i(p+1)}\right) \in E, \ \forall\ p = 1, \ldots, \ell - 1$.
- *Ein Weg P heißt* **Kreis**, *wenn $v_{i(\ell)} = v_{i(1)}$ gilt.*
- *Ein Graph G heißt zusammenhängend, wenn je zwei Knoten in G durch mindestens einen Weg verbunden sind.*
- *Ein* **Baum** *$T = (V, E)$ ist ein zusammenhängender Graph, der keine Kreise enthält.*
- *Als* **Blatt** *eines Baumes $T = (V, E)$ bezeichnen wir einen Knoten $v \in V$, der Endknoten genau einer Kante ist. Alle Knoten in V, die keine Blätter sind, bezeichnen wir als* **innere Knoten**.

Beispiel:
Zur Illustration soll Abbildung 10.1 des Graphen G dienen. Wege in G sind beispielsweise $P_1 = \{A, B, E, F\}$ und $P_2 = \{A, B, F, E, D\}$. Dabei ist P_2 ein Kreis, P_1 hingegen nicht. Ein Untergraph G' von G ist in Abbildung 10.2 dargestellt. Es ist zu bemerken, dass G' nicht zusammenhängend ist. Abbildung 10.3 zeigt einen Untergraphen T von G. Dieser Untergraph ist ein Baum, da er zusammenhängend und kreisfrei ist. Hinzufügen einer zusätzlichen Kante, z.B. von $e = (E, F)$, würde einen Kreis erzeugen und damit die Baumeigenschaft zerstören.

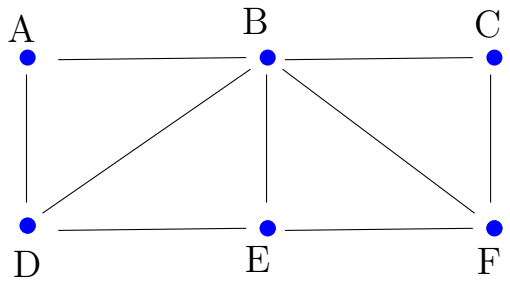


Abbildung 10.1: Zu Beispiel 10.2: Graph G.

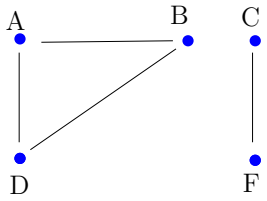


Abbildung 10.2: Untergraph G'.

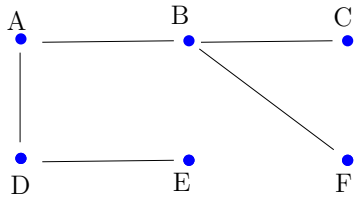


Abbildung 10.3: Untergraph T.

Wir nehmen nun an, dass alle Knoten $v \in V$ Bedarfspunkte des Telekommunikationsnetzwerkes G sind. Um diese Kunden zu verbinden, suchen wir einen Baum, der alle Knoten in V enthält. Darüber hinaus sind wir an den Kosten solch eines Konstruktes interessiert. Beispielsweise fallen Kosten für Installation und Wartung der Kabel in Abhängigkeit der verlegten Kabelmeter an. Diese Kosten lassen sich mittels Kantengewichtung im Graphen abbilden.

Definition 10.2 *Sei $G = (V, E)$ ein zusammenhängender Graph.*

- *Ein* **spannender Baum** *$T = (V', G')$ von G ist ein Untergraph mit $V' = V$ der ein Baum ist.*
- *Gegeben seien Kantengewichte $c_e \in \mathbb{R}$ für alle $e \in E$. Die* **Kosten** *eines spannenden Baumes $T = (V', G')$ sind gegeben als*

$$C(T) = \sum_{e \in E'} c_e \ .$$

- *Ein spannender Baum $T = (V', G')$ von G mit minimalen Kosten $C(T)$ heißt* **minimal spannender Baum**.

Im Folgenden beschreiben wir das *Verfahren von Prim* zur Erzeugung eines minimal spannenden Baumes. Dieses Verfahren ist exakt, d.h. es endet mit einer optimalen Lösung. Zudem wächst der Rechenaufwand für das Verfahren von Prim polynomial in Abhängigkeit der Problemgröße. Das Problem, einen minimal spannenden Baum zu finden, gehört also zu der Klasse $\mathcal{P}$ der polynomial lösbaren Probleme.
Die Idee des Verfahrens ist die folgende: Starte mit einem Teilbaum bestehend aus einem beliebigen Knoten. Erweitere den Teilbaum durch sukzessive Hinzunahme von Kanten und Knoten nach folgender Regel: Füge aus den Kanten, die „Baumknoten“ mit „Nichtbaum-Knoten“verbinden, eine kostengünstigste zum Teilbaum hinzu. Fahre fort, so lange noch nicht alle Knoten im Teilbaum aufgenommen sind.
Formale ist das Verfahren in folgendem Algorithmus beschrieben:

Algorithmus: Verfahren von Prim

Input: Zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c_e \in \mathbb{R},\ \forall e \in E$.

Schritt 1. $V' := \{v_1\}$ (mit v_1 beliebiger Startknoten)

Schritt 2. $E' := \emptyset$

Schritt 3. $C(T) := 0$

Schritt 4. Solange $V \neq V'$

- Wähle Kante $e = (i, j) \in E$ mit $i \in V'$ und $j \notin V'$ und minimalem Gewicht c_e.
- $V' := V' \cup \{j\}$, $E' := E' \cup \{e\}$, $C(T) := C(T) + c_e$

Output: Minimal spannender Baum $T = (V, E')$ und Kosten $C(T)$.

Konstruktive Methoden, die in jedem Schritt eine bestmögliche nächste Lösung auswählen, ohne dabei aber zukünftige Auswirkungen abzuwägen, nennt man **Greedy-Verfahren**. Häufig sind Greedy-Verfahren, bedingt durch das kurzsichtige Herangehen, **heuristisch**. Ein optimales Ergebnis wird also nicht notwendig gefunden, bzw. die Optimalität einer Lösung kann nicht nachgewiesen werden. Das Verfahren von Prim hingegen ist ein Beispiel für ein exaktes Greedy-Verfahrens.

Beispiel:
Betrachten Sie den Graphen G in Abbildung 10.4. Die Kantengewichte c_e sind jeweils an den betreffenden Kanten e notiert. Das Verfahren von Prim ergibt folgenden Ablauf:

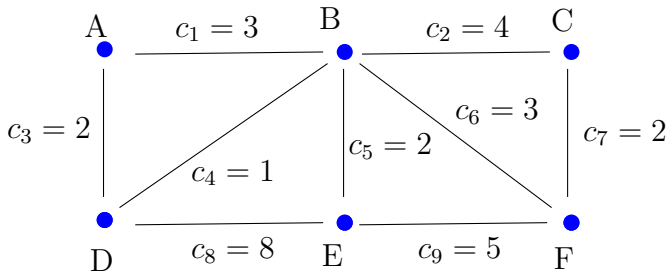


Abbildung 10.4: Zu Beispiel 10.2: Graph G.

Initialisierung: $V = \{A\}$, $E = \emptyset$, $C(T) = 0$

Iterationen k:

k	Gewählte Kante	c_e	V'	E'	$C(T)$
1	$e = 3$	$c_3 = 2$	$\{A, D\}$	$\{3\}$	2
2	$e = 4$	$c_4 = 1$	$\{A, B, D\}$	$\{3, 4\}$	3
3	$e = 5$	$c_5 = 2$	$\{A, B, D, E\}$	$\{3, 4, 5\}$	5
4	$e = 6$	$c_6 = 3$	$\{A, B, D, E, F\}$	$\{3, 4, 5, 6\}$	8
5	$e = 7$	$c_7 = 2$	$\{A, B, C, D, E, F\}$	$\{3, 4, 5, 6, 7\}$	10

Es ergibt sich ein minimal spannender Baum $T = (V, E')$ mit einer Kantenmenge $E' = \{3, 4, 5, 6, 7\}$ und Kosten $C(T) = c_3 + c_4 + c_5 + c_6 + c_7 = 10$, siehe Abbildung 10.5.
In Kapitel 3.4 wurden bereits kürzeste Wege besprochen. Ein kürzester Weg verbindet jeweils *zwei* Bedarfspunkte zu minimalen Kosten. Bäume hingegen erlauben das Verbinden von n Bedarfspunkten. Beispiel 10.2 illustriert, dass für n Bedarfspunkte das Abweichen von kürzesten Wegen vorteilhaft sein kann. Beispielsweise sind Bedarfspunkte A und C im Baum T nicht über einen kürzesten Weg verbunden.

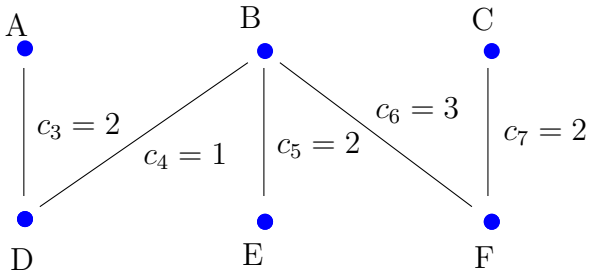


Abbildung 10.5: Minimal spannender Baum T (Bsp. 10.2)

10.3 Netzwerkdesign

Die in Kapitel 10.2 eingeführten Baumstrukturen eignen sich, um Bedarfspunkte eines Netzwerkes zu verbinden. Mit dem Verfahren von Prim haben wir darüber hinaus die Möglichkeit, kostenminimale Lösungen herbeizuführen. Eine rein kostenorientierte Analyse der Netzwerkstrukturen reicht aber für Telekommunikationsanwendungen nicht aus. Ein weiterer zentraler Aspekt ist in solchen Netzwerken die Stabilität des Netzwerkes hinsichtlich von Ausfällen. Grundsätzlich können sowohl Knoten als auch Kanten im Netzwerk ausfallen. Beispielsweise können Hard- oder Softwarekomponenten in einer Schaltstelle versagen. Stromausfälle, Steuerungsfehler oder der berühmte Bagger, der über das Kabel gefahren ist, können ebenso für Unterbrechungen des Netzes verantwortlich sein. Kunden reagieren sehr sensibel auf solche Störungen des Telekommunikationsnetzwerkes, Kundenverlust und Vertragsstrafen drohen bei Verbindungsunterbrechungen. Dagegen kann eine hohe Ausfallsicherheit, evtl. verbunden mit einer hohen Übertragungsrate ein Verkaufsargument sein, um Kunden zu binden und für Premium-Preismodelle zu gewinnen.
Im Folgenden wollen wir verschiedene Netzwerktopologien vorstellen und hinsichtlich Kosten und Ausfallsicherheit beurteilen.

Bäume und Sterne

Bäume wurden in Kapitel 10.2 eingeführt. Minimal spannende Bäume bieten die Möglichkeit, alle Knoten eines Netzwerkes unter minimalem Kosteneinsatz miteinander zu verbinden.
Bezüglich der Ausfallsicherheit schneiden Bäume allerdings schlecht ab. In einer Baumstruktur existiert zwischen jedem Knotenpaar genau ein Weg. Wird dieser Weg durchbrochen, besteht keine Möglichkeit, die betreffenden Bedarfspaare über einen „Umweg“ zu verbinden. Wenn ein Knoten oder eine Kante eines Baumes ausfällt, zerfällt der Baum sofort in zwei Teile und Knoten der verschiedenen Teilbäume können nicht mehr miteinander kommunizieren.
Bäume sind also kostengünstige Strukturen, die eine sehr geringe Ausfallsicherheit bieten. Sie eignen sich daher in Netzwerken mit geringem Sicherheitsbedarf, z.B. in lokalen Netzwerken oder Zugangsnetzwerken.
Sterne sind Bäume mit der Besonderheit, dass es nur einen inneren Knoten gibt, siehe Abbildung 10.6
Es folgt, dass in einem Stern kein Weg mehr als 2 Kanten beinhaltet und dass jeder Weg im Stern den zentralen Knoten Z enthält. Häufig finden sich Sterne als lokale Zugangsnetzwerke zu leistungsstarken Backbone-Netzwerken, wobei der zentrale Knoten Z als Verbindungsknoten

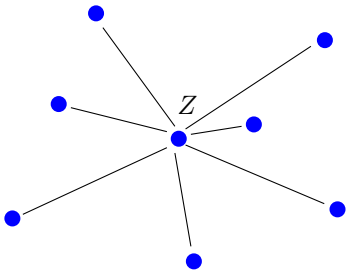


Abbildung 10.6: Topologieform Stern

zum Backbone fungiert. Im resultierenden sternförmigen Zugangsnetzwerk werden Informationen immer direkt an den Verbindungsknoten Z gesandt, ohne Nutzung von Transitknoten. Diese simple Netzstruktur erleichtert das Routing in Planung und Umsetzung. Da Sterne Bäume sind, ergeben sich darüber hinaus die gleichen Vor- und Nachteile wie bei Bäumen.

Maschen und Ringe

In Definition 10.1 hatten wir Kreise in Graphen definiert. Im Folgenden definieren wir Ringe als eine besondere Form von Kreisen.

Definition 10.3 *Ein Kreis* $R = \{v_{i(1)}, v_{i(2)}, \ldots, v_{i(\ell)}, v_{i(1)}\}$ *in einem Graphen* G *heißt* **Ring**, *wenn die Anzahl der Knoten im Kreis gleich der Anzahl der Kanten* ℓ *ist.*

Definition 10.3 impliziert, dass in einem Ring keine Knoten mehrfach vorkommen, formal gesprochen gilt $v_{i(p)} \neq v_{i(q)}$ für $p \neq q$, $p, q = 1, \ldots, \ell$, siehe Abbildung 10.7.

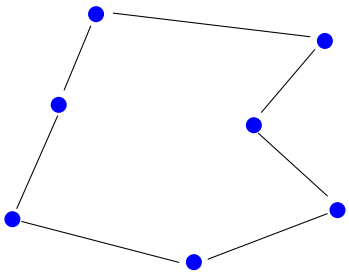


Abbildung 10.7: Topologieform Ring

Bezüglich der Ausfallsicherheit gilt für einen Ring: Bei Ausfall einer einzigen, aber beliebigen Kante, bleibt die Kommunikation zwischen allen Knoten des Ringes gewährleistet. Etwas formaler: Ein Ring bleibt auch nach Entfernen einer beliebigen Kante zusammenhängend.
Bezüglich der Kosten ist es offensichtlich, dass man die Ausfallsicherheit des Ringes durch erhöhten Kostenaufwand bezahlt. Denn durch Entfernen einer beliebigen Kante (mit positivem Kantengewicht) würde man Kosten reduzieren, ohne dabei Verbindungen zu unterbrechen.
Ein weiterer Aspekt für den Einsatz von Ringen ist der geringe operative Aufwand für einen installierten Ring, da das Routing innerhalb des Ringes starr verläuft. Gegen den Einsatz von Ringen spricht allerdings die häufig aufwändige Erweiterung von Ringnetzwerken. Diese

ist bedingt durch technologische Vorschriften bezüglich Ringlänge und maximaler Anzahl an Ringknoten.[1]
In einem **Maschennetzwerk** ist jeder Knoten zu zwei oder mehr Knoten benachbart, siehe z.B. Abbildung 10.8

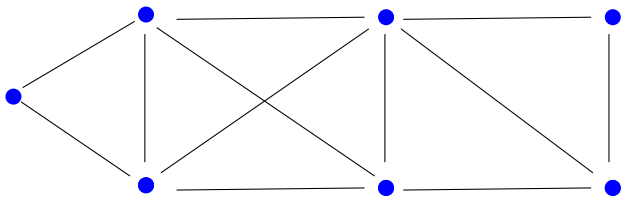


Abbildung 10.8: Topologieform Maschen

Wie Ringnetzwerke bieten vermaschte Netzwerke eine erhöhte Ausfallsicherheit aber unter Erhöhung der Kosten.
Maschen- und Ringnetzwerke sind daher gut geeignet für Netzwerke mit hohem Sicherheitsbedarf, wie Backbonenetzwerke. Ringe werden häufig dann eingesetzt, wenn weite Distanzen zu überbrücken sind und die Region zwischen den Bedarfspunkten schwer zugänglich ist. Als Beispiel sei der Glasfaserring um Südamerika genannt. Maschen hingegen kommen dann zum Einsatz, wenn die Bedarfspunkte vergleichbar nahe beieinander liegen und die Region zwischen den Knoten gut zugänglich ist. Ein Beispiel hierfür ist das Backbone-Netz der Deutschen Telekom.

Kombinierte Formen

Häufig treten kombinierte Formen auf, um die Vorteile der einzelnen Topologien zu nutzen. Vorstellbar ist ein hierarchischer Ansatz, der einzelne Regionen durch ein leistungsstarkes Backbonenetzwerk verbindet und die regionalen Bedarfspunkte durch Zugangsnetzwerke anknüpft. Um den hohen Sicherheitsanforderungen des Backbones gerecht zu werden, wird ein Maschen- oder Ringnetzwerk implementiert. Die Zugangsnetzwerke der einzelnen Regionen nutzen kostengünstige Stern- oder Baumstrukturen. Nahstreckenverkehre verbleiben in den regionalen Netzen, während Langstreckenverkehre über den Backbone geführt werden. So können Hochleistungsverbindungen für Langstreckenverkehre genutzt werden und die Anzahl der Transitknoten kann reduziert werden.

Konnektivität

Eine Kennzahl zur Beurteilung der Ausfallsicherheit eines Graphen ist seine Konnektivität.

Definition 10.4 *Die* **Konnektivität** α *eines Graphen ist gegeben durch die kleinste Zahl an Kanten, die entfernt werden müssen, um den Graphen zu teilen.*

[1] Besonders in Glasfasernetzen ergibt sich eine Abschwächung des Signals durch Dämpfung. Um die Lesbarkeit des Signals zu garantieren, muss die Länge eines Ringes und die Anzahl der enthaltenen Transitknoten beschränkt werden.

Beispiel:
Die Konnektivität eines Baumes ist $\alpha = 1$, da für jedes Knotenpaar (i, j) gilt: i ist mit j durch genau einen Weg verbunden. Entfernt man eine Kante dieses Weges kann j von i aus nicht mehr erreicht werden, der Graph ist dann nicht mehr zusammenhängend.

Beispiel:
Die Konnektivität eines Rings ist $\alpha = 2$, da der Graph bei Entfernen von einer (beliebigen) Kante zusammenhängend bleibt. Um den Graphen in zwei Teile zu teilen, müssen mindestens zwei Kanten entfernt werden.
Die Modellierung praktischer Probleme kann sich schwierig gestalten, z.B. wenn in der Realität Abhängigkeiten zwischen Kanten bestehen, die im Graphen nicht erkennbar sind. Dies kann auftreten, wenn z.B. zwei Kabelstränge, abgebildet durch zwei Kanten, über eine gemeinsame Brücke verlaufen. Bei Einsturz der Brücke würden beide Kanten gleichzeitig ausfallen. Solche Abhängigkeiten bezeichnet man auch als „shared link risk groups“.

10.4 Routing

Punkt-zu-Punkt Routing

Sollen zwei Bedarfspunkte eines Netzwerkes verbunden werden, so bieten sich Kürzeste-Wege-Verfahren wie das Verfahren von Floyd-Warshall (siehe Kapitel 3.4) an. Im Folgenden wollen wir zwei ergänzende Aspekte der kürzesten-Wege-Führung ansprechen, die in Telekommunikationsproblemen häufig eine Rolle spielen.

Minimum Hop Routing Für den Verbindungsaufbau in Telekommunikationsnetzwerken ist häufig nicht allein die Länge eines Weges, sondern auch die Anzahl der besuchten **Transitknoten** entscheidend. So sind zum Beispiel in Glasfasernetzwerken an den Knoten sogenannte **Multiplexer** installiert. Diese Hardware ermöglicht dass Ein- und Ausspeisen von Daten aus dem Netzwerk. Bezüglich des Transitverkehrs verursachen diese Hardwarekomponenten eine technologisch bedingte Dämpfung des Signals. Um die Lesbarkeit des Signals zu gewährleisten, muss die Anzahl der Transitknoten limitiert werden. Diese Beschränkung kann durch den Einsatz von weiteren Hardwarekomponenten, den **Verstärkern** (engl.: Amplifier) aufgeweicht werden. Da Verstärker kostenintensiv sind, gilt es den Einsatz dieser Geräte gering zu halten. Das **Amplifier Location Problem**, siehe [SKSKB06], beschäftigt sich mit der Frage, wo Verstärker zu platzieren sind, um die Anzahl der eingesetzten Geräte zu minimieren.
Ein zweiter Ansatz geht den Weg, die Anzahl der Transitknoten eines Weges so gering wie möglich zu halten, um den Einsatz von Verstärkern zu vermeiden. Wir suchen also Wege mit einer kleinsten Anzahl an Transitknoten (auch: **Hops**). Dieses Problem lässt sich durch einen einfachen Trick mittels klassischer kürzeste-Wege-Verfahren lösen:

1. Modifiziere die Gewichte aller Kanten $e \in E$: $c_e = 1$.
2. Finde einen kürzesten Weg bezüglich der modifizierten Kantengewichte.

Jeder Hop eines Weges erhöht die Kosten des Weges um Eins. Ein kürzester Weg bezüglich der modifizierten Kosten ist ein Weg mit minimaler Anzahl an Hops.

Kürzeste Wege bei Knotenkosten Im eben angesprochenen Minimum Hop Routing beeinflussen besuchte Knoten die Länge eines Weges. Man könnte sagen, besuchte Knoten verursachen Kosten (z.B. durch zu installierende Verstärker). Tatsächlich fallen in Telekommunikationsnetzwerken häufig Kosten an Knoten an, da hier die oft teure Routinghardware zu installieren ist. Diese Kosten können durchaus lastenabhängig sein, d.h. erhöhtes Datenverkehrsaufkommen an den Knoten erzeugt erhöhte Kosten, z.B. durch erforderliche Kapazitätsausweitungen. Um diesem Aspekte im Modell Rechnung zu tragen, empfiehlt es sich, den Knoten lastenabhängige Kosten zuzuordnen. Auch hier lassen sich bewährte kürzeste-Wege-Algorithmen durch einen einfachen Trick wiederverwenden. Da die klassischen kürzeste-Wege-Algorithmen mit Kantengewichten arbeiten, müssen wir die Kosten auf Kanten umlegen.
In Abweichung unserer eingangs getroffenen Vereinbarung wollen wir uns für das folgende Verfahren gerichtete Graphen betrachten. Seien c_v die Kosten eines Knotens v. Konkret wird folgendes Vorgehen gewählt:

1. Ersetze jeden Knoten v durch zwei neue, künstliche Knoten v_1 und v_2.
2. Alle Kanten, die in v eingehen, werden nun von v_1 abgeschlossen.
3. Alle Kanten, die aus v ausgehen, werden nun von v_2 eröffnet.
4. Erzeuge eine **künstliche** Kante t, die v_1 mit v_2 verbindet
5. Setze $c_t = c_v$ gleich der Knotenkosten.

Siehe auch Abbildungen 10.9 und 10.10.

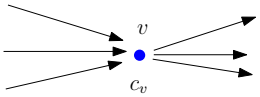


Abbildung 10.9: Knotenkosten c_v.

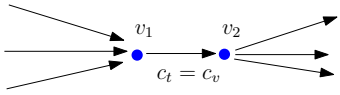


Abbildung 10.10: Modfikation zu Kantenkosten c_t.

Die künstlichen Knoten v_1 und v_2 sorgen dafür, dass der Verkehr, der ursprünglich den Knoten v passierte, nun über die Kante t geführt wird und so die Kosten verfolgt und berücksichtigt werden können.

Multicast-Routing

In Kapitel 10.4 hatten wir Aspekte von direkten Punkt-zu-Punkt Verbindungen angesprochen und die Eignung von kürzeste-Wege-Algorithmen veranschaulicht. Sollen mehr als zwei Bedarfspunkte verbunden werden, so sind Baumstrukturen ein geeigneter Modellierungsansatz. In Kapitel 10.2 wurden bereits minimal spannende Bäume diskutiert, d.h. Strukturen, die alle Knoten eines Netzwerkes zu minimalen Kosten verbinden. Was ist aber zu tun, wenn nur ein Teil der Knoten im Netzwerk mit Informationen zu versorgen ist? Sind identische Informationen an eine Gruppe von Bedarfspunkten zu übermitteln (z.B. an Abonnenten eines Internet-TVs oder Teilnehmer einer Videokonferenz), dann spricht man von einem **Multicast**. Wir bleiben zur Untersuchung dieser Anwendung bei den Baumstrukturen und betrachten im Folgenden sogenannte **Steiner-Bäume**.

Steiner-Bäume

Das Problem der minimalen Steiner-Bäume kann wie folgt verbal beschrieben werden: Gegeben sei ein Graph mit Kantengewichten. Verbinde eine gegebene Teilmenge seiner Knoten (die sogenannten **Terminale**) mit einer gegebenen **Wurzel** zu minimalen Kosten. Die Wurzel stellt üblicherweise die Informationsquelle dar, während die zu verbindenden Knoten die Empfänger des Multicasts sind.

Definition 10.5 *Gegeben sei ein Graph $G = (V, E)$, eine Wurzel $w \in V$ und Terminale $S \subset V$, sowie Kantengewichte $c_e \geq 0$, $\forall e \in E$.*

- *Ein* **Steiner-Baum** *$T = (V', E')$ ist ein zusammenhängender Untergraph von G mit $(\{w\} \cup S) \subseteq V'$.*
- *Die Knoten eines Steiner-Baumes, die nicht Terminale sind, heißen* **Steiner-Knoten**. *Wir bezeichnen:*
$$SK = V' \setminus (S \cup \{w\})$$
- *Das Steiner-Baum-Problem lautet:*
Finde einen Steiner-Baum $T = (V', E')$ mit minimalen Kosten
$$C(T) = \sum_{e \in E'} c_e \ .$$

Das folgende Beispiel zeigt, dass bei der Ermittlung von minimalen Steiner-Bäumen das Heranziehen von Steiner-Knoten notwendig sein kann, auch wenn die Terminal „direkt“ miteinander verbunden werden könnten.

Beispiel:
Betrachte den Graphen G aus Abbildung 10.11. Die drei Bedarfspunkte w, A und B sind zu verknüpfen. Dabei ist die Gesamtlänge der verlegten Kabel (repräsentiert durch die Kantengewichte) zu minimieren. Ein Baum, der die Terminale direkt mit der Wurzel verbindet, ist $T_1 = (V_1, E_1)$ mit $V_1 = \{w, A, B\}$ und $E_1 = \{(w, A), (w, B)\}$ und Kosten $c(T_1) = 10$, siehe Abbildung 10.12. Nimmt man hingegen C als Steiner-Knoten hinzu, erhält man einen kostengünstigeren Baum $T_2 = (V_2, E_2)$ mit $V_2 = \{w, A, B, C\}$ und $E_2 = \{(w, C), (A, C), (B, C)\}$ und Kosten $C(T_2) = 9$, siehe Abbildung 10.13.
Bezüglich Steiner-Bäumen lassen sich eine Reihe von Bemerkungen treffen:

- Nimmt man Definition 10.5 streng, dann sind Steiner-Bäume nicht notwendig „echte“ Bäume nach Definition 10.1 (es wird nur ein Untergraph gefordert, d.h. Kreise dürfen enthalten sein). Tatsächlich lässt sich aber zeigen, dass man sich bei der Suche nach kostenminimalen Steiner-Bäumen auf echte Bäume zurückziehen kann, da man Kreise in einem Untergraphen durch Entfernen einer beliebigen Kante des Kreises öffnen kann. Die Kosten des Steiner-Baums werden wegen $c_e \geq 0$ durch dieses Entfernen nicht steigen und Terminalknoten werden durch das Entfernen einer Kreiskante nicht vom Steiner-Baum abgeschnitten.

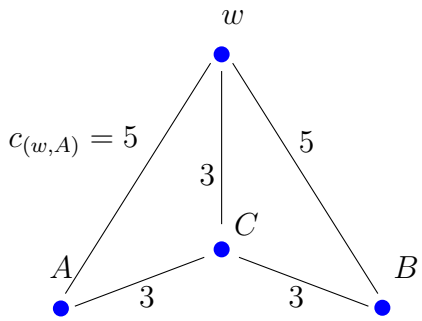


Abbildung 10.11: Graph G (Bsp. 10.4)

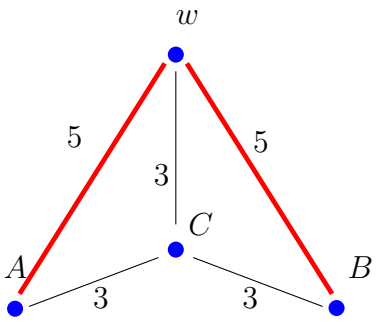


Abbildung 10.12: Baum T_1 mit $c(T_1) = 10$

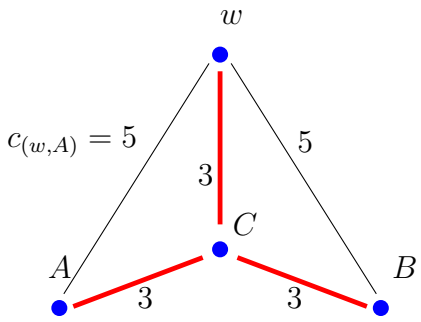


Abbildung 10.13: Baum T_2 mit $c(T_2) = 9$

- Der obigen Argumentation folgend kann man ebenso Blätter eines Steiner-Baums löschen, die keine Terminalknoten oder die Wurzel sind. Eine solche Löschung wird keinen Terminalknoten abschneiden und die Kosten nicht erhöhen.

- Gibt es genau ein Terminal, gilt also: $|S| = 1$, dann gilt es eine Verbindung zwischen zwei Punkten, dem Terminal und der Wurzel zu finden. Man hat also ein Kürzeste-Wege-Problem zu lösen. Dieses Problem kann in polynomialer Zeit, z.B. mittels des Verfahrens von Floyd-Warshall, gelöst werden.

- Sind alle Knoten (ohne die Wurzel) Terminalknoten, gilt also: $S = V \setminus \{w\}$ dann hat man einen minimal spannenden Baum zu finden. Dieses Problem kann in polynomialer Zeit, z.B. mittels der Verfahrens von Prim gelöst werden.

Die letzten beiden Bemerkungen zeigen polynomiale Spezialfälle des Steiner-Baum-Problems auf. Tatsächlich lässt sich zeigen, dass das Steiner-Baum-Problem im allgemeinen NP-schwer ist. Diese Tatsache motiviert den Einsatz von heuristischen Verfahren, also von Methoden, die nicht notwendig mit einem optimalen Lösung terminieren. Im Folgenden wollen wir zwei Heuristiken zur Erzeugung von Steiner-Bäumen kennenlernen.

Distanzgraph-Heuristik Diese Heuristik nutzt die Verwandtschaft von Steiner-Bäumen und minimal spannenden Bäumen. Sie fußt auf folgender Idee: Berechne die kürzesten Wege zwischen allen Paaren von Terminal- und Wurzelknoten. Erzeuge so den **Distanzgraphen**, d.h. einen Graphen, der nur die Knoten $V = (S \cup \{w\})$ und für alle Knotenpaare aus V die entsprechenden Distanzen als Kantengewichte enthält. Finde nun einen minimal spannenden

Baum im Distanzgraphen und übertrage dieses Ergebnis zurück in den Originalgraphen: Ersetze die gewählten Kanten durch die tatsächlich kürzesten Wege. Ist der so erzeugte Untergraph noch kein Baum, finde einen minimal spannenden Unterbaum und lösche Blätter, die keine Terminale sind. Formal beschreibt der folgende Algorithmus dieses Vorgehen:

Algorithmus: Distanzgraph-Heuristik

Input: Zusammenhängender Graph $G = (V, E)$, Kantengewichten $c_e \geq 0$, Terminalmenge $S \subset V$ und Wurzel $w \in V$.

Schritt 1. Berechne für jedes Paar $i, j \in (S \cup \{w\})$: $d(i, j) :=$ Kürzeste Distanz von i nach j in G.

Schritt 2. Erzeuge Distanzgraphen $G_D = (V_D, E_D)$ mit $V_D = (\{w\} \cup S)$, $E_D = \{(i, j) : i, j \in V_D, i \neq j\}$ und $c_{(i,j)} = d(i, j)\ \forall\ (i, j) \in E_D$.

Schritt 3. Finde minimal spannenden Baum T_D in G_D.

Schritt 4. Erzeuge G': Ersetze jede Kante (i, j) in T_D durch einen kürzesten Weg von i nach j in G.

Schritt 5. Finde minimal spannenden Baum in G'. Erzeuge einen Steiner-Baum T durch Löschen aller Steiner-Knoten die Blätter sind.

Output: Steiner-Baum T.

Beispiel:
Gegeben sei der Graph $G = (V, E)$ aus Abbildung 10.14. Die Kantengewichte c_e sind an den jeweiligen Kanten e notiert. Weiterhin sind die Wurzel w und die Terminale $S = \{s_1, s_2\}$ gekennzeichnet. Folgende kürzeste Distanzen sind in Schritt 1 zu bestimmen:

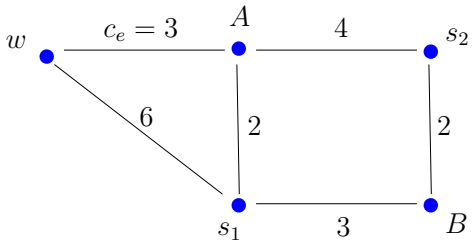


Abbildung 10.14: Graph G aus Beispiel 10.4

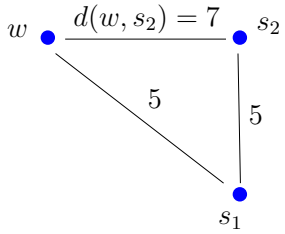


Abbildung 10.15: Distanzgraph G_D

Paar (i, j)	$d(i, j)$
(w, s_1)	5
(w, s_2)	7
(s_1, s_2)	5

Es ergibt sich also ein Distanzgraph wie er in Abbildung 10.15 angegeben ist. Ein minimal spannender Baum T_D in diesem Graphen ist gegeben durch die beiden Kanten $E_D = \{(w, s_1), (s_1, s_2)\}$ mit Kosten $C(T_D) = 10$. Durch Rückführung der kürzesten Distanzen auf kürzeste Wege im Graphen G ergibt sich der Graph G' wie in Abbildung 10.16 gegeben. Da G' bereits ein Baum ist, ergibt sich ein Steiner-Baum $T = G'$ mit Kosten $C(T) = 10$.

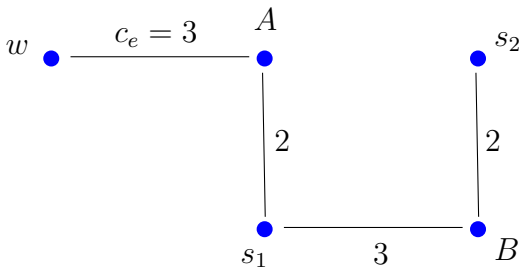


Abbildung 10.16: Steiner-Baum aus Distanzgraph-Heuristik.

Ein Nachteil des gerade vorgestellten Vorgehens ist zum einen, dass bei der Reduzierung des Problems auf den Distanzgraphen Informationen zu den hinter den Distanzen verborgenen kürzesten Wegen verloren gehen und man des Weiteren mit kürzesten Wegen nicht immer gut beraten sein muss. Für Beispiel 10.4 (Seite 115) würde die Distanzgraph-Heuristik den Baum T_1 mit $C(T_1) = 10$ erzeugen.
Wir wollen im Folgenden eine Heuristik betrachten, die ebenso mit kürzesten-Wegen arbeitet, diese aber direkt im Originalgraphen notiert und damit bereits bestehende Teilbäume nutzen kann.

Cheapest Insertion (CHINS) Die Idee des von Takahashi und Matsuyama [TM80] vorgeschlagenen Verfahren ist die folgende. Mit der Wurzel startend generiert man einen Teilbaum durch sukzessive Hinzunahme kürzester Wege zwischen noch nicht errfassten Terminalknoten und dem Teilbaum. Man fährt fort, bis alle Terminalknoten im Teilbaum enthalten sind. Formal ergibt sich folgender Algorithmus:

Algorithmus: Cheapest Insertion (CHINS)

Input: Zusammenhängender Graph $G = (V, E)$, Kantengewichten $c_e \geq 0$, Terminalmenge $S \subset V$ und Wurzel $w \in V$.

Schritt 1. Setze $V' = \{w\}$, $T' = (\{w\}, \emptyset)$..

Schritt 2. Solange $V' \neq (S \cup \{w\})$

- Finde Knoten $i^* \in V'$ und $j^* \in S \setminus V'$ mit minimaler Distanz:
  $d(i^*, j^*) = \min\{d(i, j) : i \in V', j \in S \setminus V'\}$ und zugehörigen Weg $P(i^*, j^*)$.
- Füge Knoten und Kanten des Weges $P(i^*, j^*)$ zu V' und E' hinzu.

Output: Steiner-Baum $T = (V', E')$.

Beispiel:
Wir betrachten erneut den Graphen $G = (V, E)$ aus Abbildung 10.14 mit Kantengewichten c_e, Wurzel w und Terminalen $S = \{s_1, s_2\}$. Das Verfahren *Cheapest Insertion* zeigt folgenden Ablauf.

Initialisierung: $V' = \{w\}$, $T = (w, \emptyset)$

Iterationen k:

k	i^*	j^*	$d(i^*, j^*)$	$P(i^*, j^*)$	V'	E'
1	w	s_1	5	(w, A, s_1)	$\{w, A, s_1\}$	$\{(w, A), (A, s_1)\}$
2	A	s_2	4	(A, s_2)	$\{w, A, s_1, s_2\}$	$\{(w, A), (A, s_1), (A, s_2)\}$

Es ergibt sich ein Steiner-Baum $T = (V', E')$ mit $E' = \{(w, A), (A, s_1), (A, s_2)\}$ und Kosten $C(T) = 9$, siehe Abbildung 10.17.

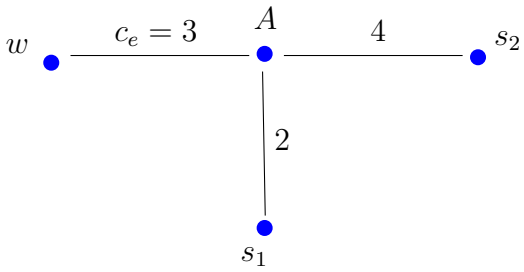


Abbildung 10.17: Steiner-Baum aus CHINS-Heurstik.

Für obiges Beispiel zeigt das CHINS-Verfahren, verglichen mit dem Ergebnis der einfachen Distanzgraph-Heuristik, ein besseres Verhalten. Bei der Anbindung von s_2 an den Steiner-Baum wurden die Informationen über den ersten eingebundenen kürzesten Weg genutzt und s_2 wurde nicht über einen kürzesten Weg an w oder s_1, sondern über einen kürzesten Weg zum bestehenden Teilbaum angebunden. Dagegen muss für das Beispiel 10.4 festgestellt werden, dass wie schon die einfache Distanzgraph-Heuristik auch das CHINS-Verfahren den Baum T_1 mit $C(T_1) = 10$ erzeugt, die optimale Lösung also nicht findet.

Push-Tree

In den bisherigen Kapiteln haben wir uns mit der Verbindung aller Knoten eines Graphen (spannende Bäume) und der Verbindung einer Teilmenge an Knoten (Steiner-Bäume) beschäftigt. Probleme dieser Art erwachsen z.B. durch das Senden identischer Informationen an eine Menge an Bedarfsknoten. Wir nehmen nun an, dass diese Informationen in festgelegten Abständen aktualisiert werden. Darüber hinaus nehmen wir an, dass die Bedarfsknoten die Informationen in festgelegten Abständen abfragen. Ist nun die Rate der Abfrage hoch, dazu die Rate der Informationsaktualisierung gering, so könnte es sich lohnen, die Informationen nicht nur am Wurzelknoten zu belassen, sondern auf ausgewählten Knoten des Netzwerkes zu **spiegeln**. Das heißt, die Informationen werden an verschiedenen Stellen des Netzwerkes bevorratet und müssen dort gemäß der festgelegten Abstände aktualisiert werden. Bedarfsknoten müssen sich dann

nicht mehr zur Wurzel verbinden, sondern können eine Verbindung zum nächstgelegenen Spiegel wählen. Dieses Vorgehen ist also eine Erweiterung des klassischen Steiner-Baum-Problems bei der es gilt, zwei Kostentypen abzuwägen: die Kosten für das Aussenden der Informationen (in Abhängigkeit einer Updaterate) und die Kosten für das Beziehen von Informationen (in Abhängigkeit einer Requestrate). Wir bezeichnen den Vorgang des Aussendens von Informationen zu den Spiegeln als den **Push** und den Vorgang des Beziehens von Informationen als den **Pull**. Sei $G = (V, E)$ ein (ungerichteter) Graph. Im Folgenden verwenden wir folgende Bezeichnungen:

$S \subset V$	Menge der Bedarfsknoten
$w \in V$	Informationsquelle (Wurzel)
$r(v)$	Requestrate [Abfragen pro Zeiteinheit] eines Knotens $v \in V$ $r(v) = 0$ für $v \notin S$
μ	Updaterate [Updates pro Zeiteinheit] (der Informationsquelle)
$PT = (V^{PT}, E^{PT})$	Push-Tree: Untergraph von G der die Informationsquelle enthält: $w \in V^{PT}$
P_v	Anbindung Bedarfsknoten an Push-Tree: Weg von Knoten $v \in S$ zu Knoten $j \in V^{PT}$

Das Aussenden der Informationen soll über eine Baumstruktur erfolgen, alle Knoten des sogenannten **Push-Trees (PT)** fungieren als Spiegel. Die auf diesen Knoten bevorrateten Informationen müssen also abhängig von der Updaterate aktualisiert werden, beispielsweise fünf mal pro Zeiteinheit bei $\mu = 5$. Die Kosten eines Pushs ergeben sich also aus den Kosten des zugrundeliegenden Baums und der Updaterate:

Definition 10.6 *Die* **Push-Kosten** *sind gegeben durch*

$$C_{PUSH} = \mu \cdot \sum_{e \in PT} c_e \ .$$

Zu ermitteln ist weiterhin für jeden Bedarfsknoten $v \in S$ ein Weg P_v zum Push-Tree. Die Kosten des Pulls ergeben sich aus der Länge dieser Wege jeweils multipliziert mit der zugehörigen Abfragerate. Um die Schreibweise übersichtlich zu halten wir in folgender Definition etwas nachlässig mit der Notation um und schreiben $e \in P_v$ wenn wir die Kanten eines Weges P_v ansprechen wollen.

Definition 10.7 *Die* **Pull-Kosten** *sind gegeben durch*

$$C_{PULL} = \sum_{v \in S} \left(r(v) \cdot \sum_{e \in P_v} c_e \right) \ .$$

Zusammenfassend wollen wir folgendes Problem lösen:

Definition 10.8 (Push-Tree-Problem) *Gegeben seien ein Graph $G = (V, E)$, eine Menge an Bedarfsknoten $S \subset V$, eine Informationsquelle $w \in V$, eine Updaterate $\mu > 0$, Requestraten $r(v) \geq 0$, sowie Kantengewichte c_e.*
Finde einen Push-Tree PT sowie Wege P_v für alle $v \in S$ so dass die Kosten aus Push und Pull minimal sind:

$$\textit{minimiere } C_{PT} = C_{PUSH} + C_{PULL}$$

Beispiel:
Betrachte den Graphen aus Abbildung 10.18.

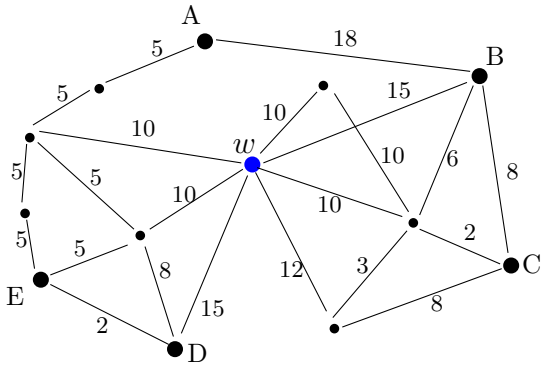


Abbildung 10.18: Graph aus Beispiel 10.4

Die Informationsquelle w und die Bedarfsknoten $S = \{A, B, C, D, E\}$ sind gegeben. Weiterhin seien die Updaterate $\mu = 5$ und die Requestraten $r(A) = 2, r(B) = 5, r(C) = 4, r(D) = 3$ und $r(E) = 5$.
Betrachten wir nun Abbildung 10.19 zur Veranschaulichung einer Push-Tree Lösung: Durchgehend gezogene Kanten repräsentieren den Push-Tree PT, gestrichelte Kanten repräsentieren Wege von Bedarfsknoten zum Push-Tree. Dabei ist zu beachten, dass einige Bedarfsknoten bereits auf dem Push-Tree liegen.

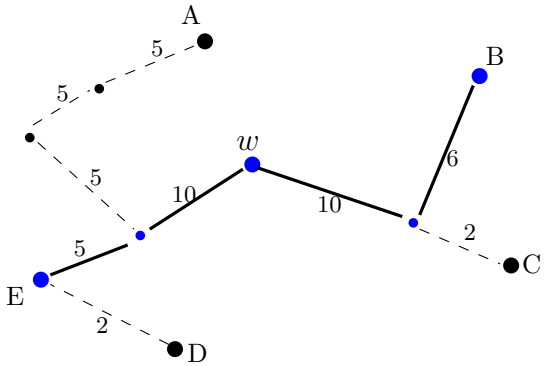


Abbildung 10.19: Lösung zu Beispiel 10.4

Die Kosten der in Abbildung 10.19 dargestellten Lösung ergeben sich als:

$$C_{PUSH} = 5 \cdot (10 + 10 + 5 + 6) = 155 \ ,$$

$$C_{PULL} = 15r(A) + 0r(B) + 2r(C) + 2r(D) + 0r(E) = 15 \cdot 2 + 2 \cdot 4 + 2 \cdot 3 = 44$$

und

$$C_{PT} = 155 + 44 = 199 \ .$$

Spezialfall: Steiner-Baum Ist die Updaterate hinreichend klein, so lohnt es sich jeden Bedarfsknoten als Spiegel einzurichten, d.h. alle Bedarfsknoten werden in den Push-Tree integriert. Genauer tritt dieser Fall ein, wenn die Updaterate kleiner als die kleinste Requestrate ist: $\mu < min_{v \in S} r(v)$. Da nun alle Knoten in S mit w über einen Baum im Graphen G zu verbinden sind, ist diese Aufgabe äquivalent zum Steiner-Baum-Problem (siehe Kapitel 10.4). Für diesen Spezialfall lassen sich also Methodiken des Steiner-Baum-Problems, wie CHINS oder die einfache Distanzgraph Heuristik anwenden. Da das Steiner-Baum-Problem nachgewiesen NP-schwer ist lässt sich ableiten, dass das Push-Tree-Problem ebenfalls NP-schwer ist (da ein polynomialer Algorithmus für das Push-Tree-Problem das Steiner-Problem ebenfalls mit polynomialem Aufwand lösen würde).

Beispiel:
Wir betrachten erneut die Aufgabenstellung aus Beispiel 10.4. Sei die Updaterate nun als $\mu = 1$ festgelegt. Damit folgt $\mu = 1 < 2 = min_{v \in S} r(v)$, das Kriterium für den Spezialfall Steiner-Baum ist also erfüllt. Das zugehörige Push-Tree-Problem erfordert daher das Lösen eines Steiner-Baum-Problems. Die Lösung dieses Problems ist in Abbildung 10.20 illustriert.

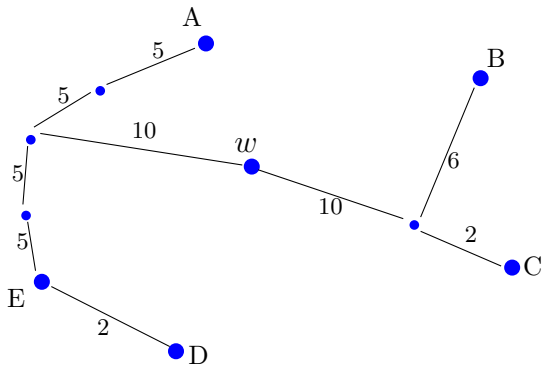


Abbildung 10.20: Lösung zu Beispiel 10.4

Spezialfall: Kürzeste Wege Werden die Informationen hingegen sehr häufig erneuert, ist also die Updaterate hinreichend groß, dann lohnt sich das Spiegeln auf zusätzlichen Knoten nicht und alle Bedarfsknoten müssen sich mit der Informationsquelle w verbinden. „Hinreichend groß“ heißt genauer: größer als die Summe aller Requestraten, also $\mu > \sum_{v \in S} r(v)$. In diesem Fall besteht ein Push-Tree nur aus der Informationsquelle w. Es folgt dass für jeden Bedarfsknoten, also $|S|$-mal, ein kürzeste-Wege-Problem zu lösen ist. Da es polynomiale Algorithmen für das kürzeste-Wege-Problem gibt (siehe Kapitel 3.4) kann dieser Spezialfall mit polynomialem Aufwand gelöst werden.

Beispiel:
Betrachte die Aufgabenstellung aus Beispiel 10.4. Sei die Updaterate nun als $\mu = 20$ festgelegt.

Damit folgt $\mu = 20 > 19 = \sum_{v \in S} r(v)$. Das Lösen von fünf kürzeste-Wege-Problemen liefert das in Abbildung 10.21 gezeigte Ergebnis.

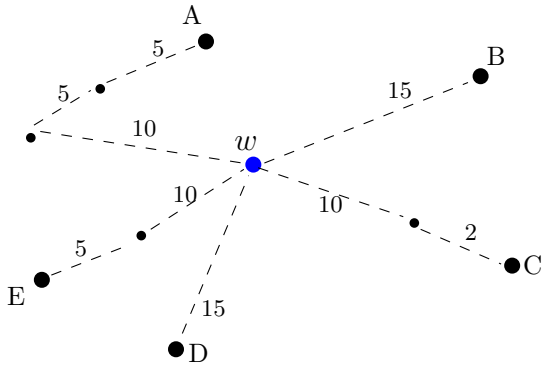


Abbildung 10.21: Lösung zu Beispiel 10.4

Exakte Methode für Bäume Wir stellen im Folgenden eine exakte Methode von Havet und Wennink [HW04] zur Lösung des Push-Tree-Problems vor. Diese Methode setzt voraus, dass der Graph G ein Baum ist. Diese Einschränkung ist sehr stark, da Telekommunikationsgraphen in der Regel nicht als reine Bäume vorliegen. Die Studie der exakten Methode ist trotzdem dienlich, da diese Methode später als Grundlage einer Heuristik für den allgemeinen (nicht-Baum) Fall dienen wird.
Wir nehmen also an, dass $G = (V, E)$ ein Baum ist. Dann folgt, dass für jeden Knoten $v \in S$ genau ein Weg zur Informationsquelle w existiert. Der Weg, den Informationen von der Quelle zu einem Knoten v nehmen, ist also bereits mit der Aufgabenstellung festgelegt. Es bleibt die Frage, ob die Kanten dieses Weges Teil des Push-Trees sein sollen oder nicht. Um dies für eine Kante $e \in E$ zu entscheiden, wird folgendes Vorgehen gewählt: Der Graph G zerfällt bei Entfernung von e in genau zwei Untergraphen (da G ein Baum ist). Wir betrachten den Untergraphen G_e, der w nicht enthält und summieren die Requestraten, die aus diesem Untergraphen entstehen. Sei $V(G_e)$ die Knotenmenge des Untergraphen G_e. Wir definieren:

$$\lambda_e = \sum_{v \in V(G_e)} r(v) \ .$$

Es ist zwingend, dass Bedarfe in Höhe von λ_e die Kante e passieren werden. Ist also $\lambda_e > \mu$, dann lohnt es sich, e in den Push-Tree aufzunehmen. Ist andernfalls $\lambda_e \leq \mu$, dann bringt das Einbinden von e in den Push-Tree keinen Vorteil. Es ist zu beachten, dass für diese Überlegungen das Gewicht c_e der Kante e völlig unerheblich ist. Die gerade aufgezeigte Entscheidungsregel soll also zur Erstellung eines Push-Trees dienen:

Exakte Methode für Baum:

Erzeuge Push-Tree aus allen Kanten e für die gilt:

$$\lambda_e = \sum_{v \in V(G_e)} r(v) > \mu$$

Es verbleibt die Frage, ob der so erzeugte Untergraph tatsächlich ein Baum, also zusammenhängend ist. Dies kann man sich überlegen, indem man aufzeigt, dass die Werte λ_e nichtwachsend sind wenn man sich entlang eines Weges von der Informationsquelle w entfernt.

Beispiel:
Betrachte den Graphen G aus Abbildung 10.22. Die Requestraten $r(v)$ sind an den jeweiligen Knoten vermerkt. Für $\mu = 4$ zeigt Abbildung 10.23 den Push-Tree mittels durchgezogener Kanten an, die Werte λ_e sind an jeder Kante notiert.

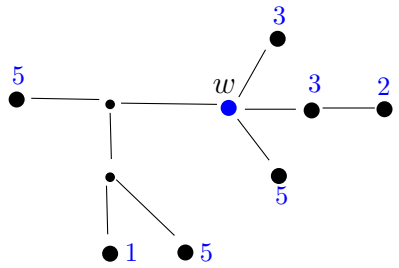


Abbildung 10.22: Graph aus Beispiel 10.4

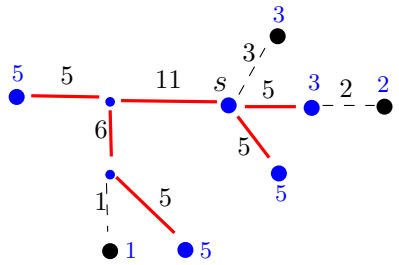


Abbildung 10.23: Lösung zu Beispiel 10.4

Heuristik für den allgemeinen Fall Die soeben beschriebene exakte Methode für Bäume soll nun als Grundlage für den allgemeinen Fall dienen, das Verfahren ist in folgendem Algorithmus beschrieben:

Algorithmus: Push-Tree-Problem: Heuristik für den allgemeinen Fall

Input: Graph $G = (V, E)$ mit Kantengewichten c_e
und S, w, μ, $r(v)$ $\forall v \in S$

Schritt 1. Erzeuge in G einen Baum $T = (V', E')$ so, dass
1. T Untergraph von G ist und
2. $S \cup \{w\} \subseteq V'$ gilt.

Schritt 2. Wende auf T die exakte Methode für den Spezialfall Baum an.

Es verbleibt die Frage, wie der Baum T in G erzeugt werden soll. Grundlagen könnten minimalspannende-Bäume, kürzeste-Wege-Bäume oder Steiner-Bäume sein. Havet und Wenningk [HW04]

haben gezeigt, dass sich eine heuristische Lösung basierend auf minimal-spannenden-Bäumen oder kürzeste-Wege-Bäumen beliebig schlecht verhalten kann, d.h. beliebig weit von einer optimalen Lösung abweichen kann. Dagegen lässt sich die Abweichung vom Optimum für Heuristiken basierend auf Steiner-Bäumen beschränken. Vertiefend vergleichen Caserta et al. [CFR$^+$09] in experimentellen Studien heuristische Ergebnisse basierend auf minimal-spannenden Bäumen und Steiner-Bäumen.

Ausfallsicheres Routing

In Kapitel 10.3 wurden Netzwerktopologien hinsichtlich ihrer Ausfallsicherheit diskutiert. Es wurde der Begriff der Konnektivität eingeführt. Die Konnektivität eines Graphen gibt eine Aussage darüber, wieviele Kanten des Graphen man entfernen kann, bevor der Graph in zwei Teile zerfällt. Diese strukturellen Betrachtungen lassen aber die tatsächliche Wegeführung des Datenflusses außer Acht. Angenommen, wir fordern eine Konnektivität von $\alpha = 2$. D.h. für jedes Bedarfspaar sollen zwei kanten- (oder knoten-)disjunkte Wege gegeben sein. Betrachten wir Ringe, ist die Wegeführung einfach, denn dann existieren für jedes Bedarfspaar genau zwei Wege. In vermaschten Netzen hingegen finden sich häufig mehr als zwei Wege zur Verbindung eines Bedarfspaares. Dies wirft die Frage nach kostenminimalen Paaren von kanten-(knoten-)disjunkten Wegen auf.

Definition 10.9 *Eine Menge von Wegen heißt kanten- (knoten-)disjunkt, wenn die Wege keine gemeinsamen Kanten (Knoten) haben.*

In den folgenden Untersuchungen wollen wir ausschließlich gerichtete Graphen betrachten.

Beispiel:
Betrachte den Graphen $G = (V, E)$ mit Kantengewichten c_e und dem zu verbindenden Bedarfspaar (A, H), siehe Abbildung 10.24. Ein kürzester Weg ist gegeben durch $P^* = (A, F, G, H)$. Ein kürzestes Paar kantendisjunkter Wege ist gegeben durch $P_1^e = (A, F, G, H), P_2^e = (A, B, F, C, H)$ mit Kosten von 9, siehe Abbildung 10.25. Ein kürzestes Paar knotendisjunkter Wege findet sich dagegen in Abbildung 10.26 mit $P_1^v = (A, F, C, H), P_2^v = (A, B, G, H)$ und Kosten 10.

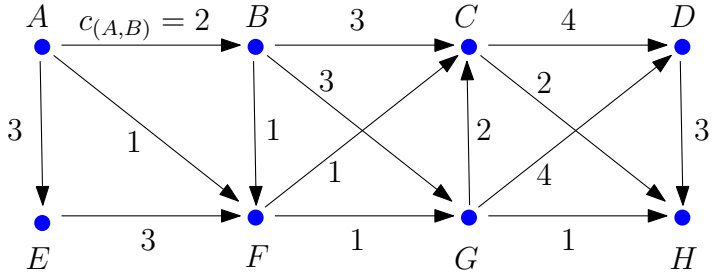


Abbildung 10.24: Graph G zu Beispiel 10.4.

Aus Beispiel 10.4 lassen sich folgende Beobachtungen ableiten:

- Ein kürzester Weg von O nach D ist nicht notwendigerweise Teil des kürzesten Paares knotendisjunkter Wege, wie aus Abbildung 10.26 zu ersehen ist. Die gleiche Beobachtung lässt sich für kürzeste Paare kantendisjunkter Wege feststellen. Ein entsprechendes Beispiel zu entwickeln soll als Übung verbleiben.

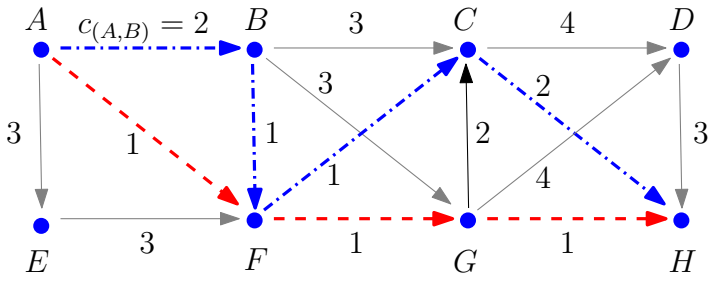


Abbildung 10.25: Kürzestes Paar kantendisjunkter Wege.

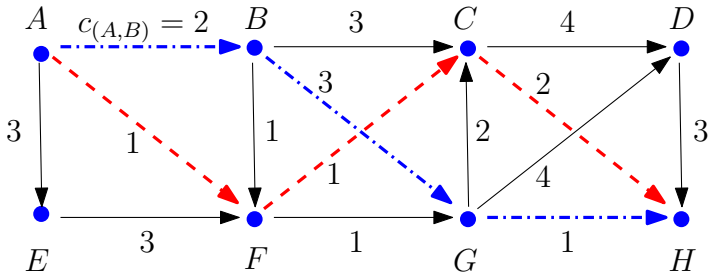


Abbildung 10.26: Kürzestes Paar knotendisjunkter Wege.

- Knotendisjunktheit zieht Kantendisjunktheit nach sich, aber nicht umgekehrt.

Einfacher Lösungsansatz Im Folgenden beschreiben wir einen sehr einfachen, intuitiven Ansatz zur Ermittlung zweier kantendisjunkter Wege: Suche einen kürzesten Weg P_1, entferne anschließend die Kanten von P_1 aus dem Graphen und suche einen zweiten kürzesten Weg P_2. Die formale Beschreibung ist in folgendem Algorithmus gegeben:

Algorithmus: Problem kürzester, kantendisjunkter Wege: Einfacher Ansatz

Input: Gerichteter Graph $G = (V, E)$ mit Kantengewichten c_e und Bedarfspaar (O, D).

Schritt 1. Finde einen kürzesten Weg P_1 von O nach D.

Schritt 2. Erzeuge einen reduzierten Graphen G' durch Entfernen der Kanten des Weges P_1 aus dem Graphen G.

Schritt 3. Finde einen zweiten kürzesten Weg P_2 von O nach D im reduzierten Graphen G'.

Output Falls es keinen Weg P_2 gibt: „Kein Paar kantendisjunkter Wege gefunden.“
Sonst: Lösung P_1, P_2.

Der einfache Ansatz birgt zwei Nachteile:

- Eventuell wird kein zweiter Weg P_2 gefunden obwohl ein Paar kantendisjunkter Wege im Graphen existiert (weil P_1 „blockiert“). (Beispiel: siehe Übung)

- Der Weg P_2 ist möglicherweise deutlich länger als der Weg P_1.

Um diese Nachteile auszuräumen, sollte der zweite Weg die Möglichkeit haben, Teile des ersten kürzesten Weges „auszulöschen“, wenn dadurch Kosten gespart werden können oder wenn dadurch ein zweiter Weg erst möglich wird. Dieser Idee folgend stammt ein Ansatz von Suurballe, den wir im Folgenden darstellen.

Kürzeste Paare von disjunkten Wegen Wir beschreiben eine modifizierte Variante des Algorithmus von Suurballe [Suu74]. Sei ein Bedarfspaar (O, D) in einem Graphen G gegeben. Finde zuerst einen kürzesten Weg von O nach D (analog zum einfachen Ansatz). Finde dann einen zweiten kürzesten „Erweiterungs“-Weg in einem modifizierten Graphen. Diese Idee wurde von Bhandari [Bha94] aufgegriffen und für **spann-disjunkte** Wege erweitert.[2]
Wir beginnen mit einer vereinfachten Variante des Algorithmus für kantendisjunkte Wege. Die folgende Definition wird benötigt.

Definition 10.10 *Die* **Vereinigung** *zweier Wege P_1 und P_2 ist gegeben als die Menge aller Kanten in P_1 und P_2 abzüglich der Kanten $(i, j), (j, i)$ für die gilt: $(i, j) \in P_1$ und $(j, i) \in P_2$.*

Die Vereinigung zweier Wege löscht also Kanten aus, die vom ersten Weg „hinwärts“ und vom zweiten Weg „rückwärts“ passiert werden. Die entstehenden Teilstücke lassen sich zu neuen Wegen zusammenfassen, siehe folgendes Beispiel.

Beispiel:
Betrachte den Graphen in Abbildung 10.27 und die Wege $P_1 = (A, B, G, F, C, H)$ und $P_2 = (A, F, G, H)$. Da die Kante (G, F) in P_1 enthalten ist und ihre Umkehrung (F, G) Teil von P_2 ist, werden diese Kanten durch eine Vereinigung von P_1 und P_2 gelöscht. Die Vereinigung dieser Wege führt also zu zwei neuen Wegen $P_1' = (A, F, C, H)$ und $P_2' = (A, B, G, H)$, siehe Abbildung 10.28.

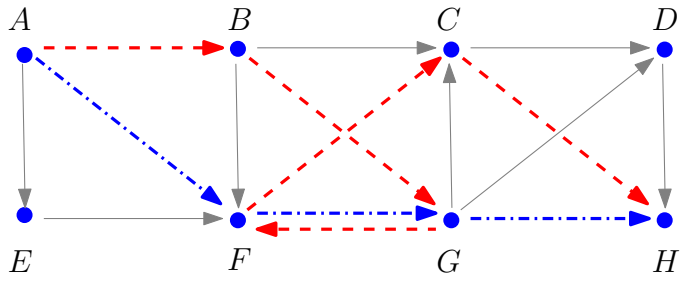


Abbildung 10.27: Wege P_1 und P_2 (Bsp. 10.4).

Die Vereinigung zweier Wege erlaubt also dem zweiten Weg, Teile des ersten Weges „rückwärts“ zu gehen und damit die Auslöschung dieser Teilwege zu erzwingen. Dieses Vorgehen kann für unsere Problematik genutzt werden, indem man zu jeder Kante des ersten (kürzesten) Pfades eine zweite, rückwärtsgerichtete Kante erzeugt, welche das negative Kantengewicht der vorwärtsgewandten Kante trägt. Das Passieren so einer Kante würde dem zweiten Weg also einen negativen Kostenanteil einbringen, der zweite Weg würde also „etwas sparen“ wenn er so

[2]Ein **Spann** in Telekommunikationsnetzwerken bezeichnet eine Menge von Kanten, die sich beispielsweise eine physikalische Ressource (z.B. Kanal) teilen und dadurch Abhängigkeiten bezüglich Ausfällen aufzeigen. Solche Mengen werden auch als **shared-link risk group** bezeichnet.

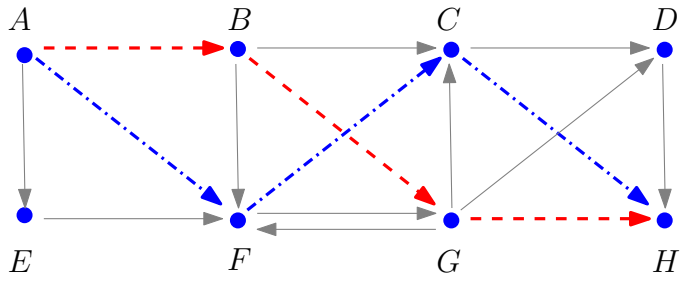


Abbildung 10.28: Vereinigung P_1 und P_2.

eine rückwärtsgewandte Kante nutzt. Dies entspricht dem Sachverhalt, dass bei einer Vereinigung der Wege die vorwärtsgerichtete Kante tatsächlich ausgelöscht wird und die Kosten damit entfallen. Der folgende Algorithmus greift diese Idee auf:

Algorithmus: Problem kürzester, kantendisjunkter Wege: Suurballes Algorithmus (modifizierte Variante)

Input: Gerichteter Graph $G = (V, E)$ mit Kantengewichten c_e und Bedarfspaar (O, D).

Schritt 1. Finde einen kürzesten Weg $\bar{P}_1$ von O nach D.

Schritt 2. Erzeuge modifizierten Graphen G' durch folgendes Vorgehen: Für alle Kanten $e = (i, j)$ im kürzesten Weg $\bar{P}_1$:
1. Lösche Kante $e = (i, j)$.
2. Erzeuge Kante $e' = (j, i)$ (soweit noch nicht vorhanden).
3. Setze Kantengewicht $c_{e'} = -c_e$.

Schritt 3. Finde einen zweiten kürzesten Weg $\bar{P}_2$ von O nach D im modifizierten Graphen G'.

Schritt 4. Falls es keinen Weg P_2 gibt: Output: „Es existiert kein Paar kantendisjunkter Wege."

Schritt 5. Sonst: Erzeuge kürzestes Paar kantendisjunkter Wege P_1, P_2 durch Vereinigung von $\bar{P}_1$ und $\bar{P}_2$.
Output: Lösung P_1, P_2.

Beispiel:
In Beispiel 10.4 auf Seite 125 wurde bereits für einen Graphen $G = (V, E)$ und ein Bedarfspaar (A, D) ein kürzestes Paar kantendisjunkter Wege angegeben. Wir wollen für diesen Graphen die Bestimmung dieses Paares formal nachvollziehen. Gegeben sei also der Graph G aus Abbildung 10.29.
Dem Algorithmus von Suurballe folgend ermitteln wir einen kürzesten Weg $\bar{P}_1 = (A, F, G, H)$. Der modifizierte Graph, erzeugt in Schritt 2, ist in Abbildung 10.30 gegeben.
Ein zweiter kürzester Weg in G' ergibt sich als $\bar{P}_2 = (A, B, F, C, H)$. Wir erhalten also ein kürzestes Paar kantendisjunkter Wege aus der Vereinigung von $\bar{P}_1$ und $\bar{P}_2$, welche in diesem Fall keine Löschung von Kanten nach sich zieht. Es gilt also: $P_1 = \bar{P}_1 = (A, F, G, H)$ und $P_2 = \bar{P}_2 = (A, B, F, C, H)$, siehe Abbildung 10.31.

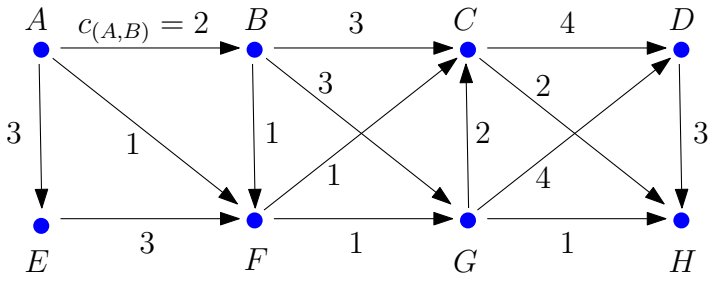


Abbildung 10.29: Graph G zu Beispiel 10.4.

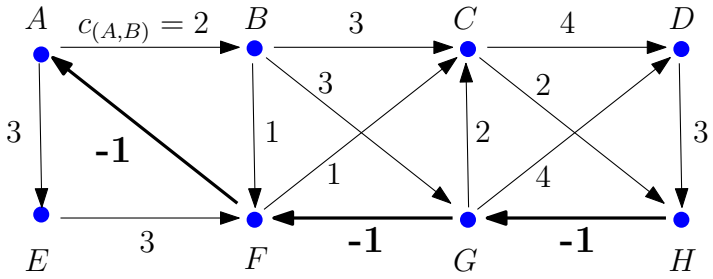


Abbildung 10.30: Modifizierter Graph G'.

Soll nun auch die Knotendisjunktheit sichergestellt werden, so behelfen wir uns mit einer einfachen Modifikation des Graphen. Betroffen von einer möglichen Verletzung der Knotendisjunktheit im Algorithmus von Suurballe sind nur die Knoten entlang des ersten kürzesten Weges $\bar{P}_1$ (ausgenommen Knoten O und D, diese müssen natürlich in beiden Wegen vorkommen). Diese Knoten werden durch einen einfachen „Trick“ geschützt: Dupliziere jeden Knoten i entlang des Weges $\bar{P}_1$ zu i' und i''. Verbinde i' und i'' mit einer der Flussrichtung von $\bar{P}_1$ entgegengesetzten Kante. Verbinde weiterhin ein- und ausgehende Kanten des Knotens i, die nicht zu $\bar{P}_1$ gehören, mit i' und i'' derart, dass die Knoten nicht als Transitknoten benutzt werden können. Folgender Algorithmus gibt eine formale Beschreibung:

Algorithmus: Problem kürzester knotendisjunkter Wege: Suurballes Algorithmus (modifizierte Variante)

Input: Gerichteter Graph $G = (V, E)$ mit Kantengewichten c_e und Bedarfspaar (O, D).

Schritt 1. Schritte 1. und 2. analog zum Algorithmus von Suurballe für kürzeste kantendisjunkte Wege.

Schritt 2. Erstelle modifizierten Graphen G'': Für jeden Knoten i in $\bar{P}_1$ (mit Ausnahme O, D):

1. Ersetze i durch zwei neue Knoten i' und i''.
2. Verbinde i' und i'' mit einer Kante $e = (i'', i')$ mit Kantengewicht $c_e = 0$.
3. Ersetze die Kante $e = (k, i)$, die rückwärtsgerichtet entlang $\bar{P}_1$ verläuft durch $e' = (k, i'')$ mit Kantengewicht $c_{e'} = c_e$.
4. Ersetze die Kante $e = (i, k)$, die rückwärtsgerichtet entlang $\bar{P}_1$ verläuft durch $e' = (i', k)$ mit Kantengewicht $c_{e'} = c_e$.
5. Für alle restlichen eingehenden Kanten $e = (k, i)$ (die weder aus P_1 noch aus seiner

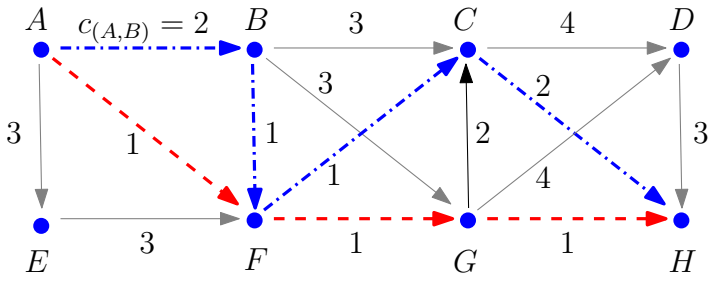


Abbildung 10.31: Kürzestes Paar kantendisjunkter Wege.

Umkehrung stammen): Ersetze e durch $e' = (k, i')$ mit Kantengewicht $c_{e'} = c_e$.
6. Für alle restlichen ausgehenden Kanten $e = (i, k)$ (die weder aus $\bar{P}_1$ noch aus seiner Umkehrung stammen): Ersetze e durch $e' = (i'', k)$ mit Kantengewicht $c_{e'} = c_e$.

Schritt 3. Finde einen zweiten kürzesten Weg $\bar{P}_2$ von O nach D im modifizierten Graphen G''.

Schritt 4. Falls es keinen Weg $\bar{P}_2$ gibt: Output: „Es existiert kein Paar knotendisjunkter Wege."

Schritt 5. Sonst: 1. Überführe G'' und $\bar{P}_2$ zurück nach G' .
2. Erzeuge kürzestes Paar knotendisjunkter Wege P_1, P_2 durch Vereinigung von $\bar{P}_1$ und $\bar{P}_2$.
3. Output: Lösung P_1, P_2

Beispiel:
Gegeben sei der Graph G aus Beispiel 10.4, siehe Abbildung 10.29. Wir verwenden den bereits ermittelten kürzesten Weg $\bar{P}_1 = (A, F, G, H)$ und den zugehörigen modifizierten Graphen G', siehe Abbildung 10.30 und fahren mit Schritt 2 des Algorithmus von Suurballe zur Erstellung knotendisjunkter Wege fort. Es ist also nun durch eine weitere Modifikation Graph G'' zu erstellen. Hierfür müssen alle Knoten entlang $\bar{P}_1$ (mit Ausnahme des Start- und Endeknotens) dupliziert und neu verbunden werde, siehe Abbildung 10.32. Ein kürzesten Weg $\bar{P}_2$ in G'' ist gegeben als $\bar{P}_2 = (A, B, G', F'', C, H)$ mit Kosten 7. Die Vereinigung der Wege $\bar{P}_1$ und $\bar{P}_2 = (A, B, G, F, C, H)$ ergibt ein kürzestes Paar knotendisjunkter Wege $P_1 = (A, F, C, H)$ und $P_2 = (A, B, G, H)$ mit Kosten 10, siehe Abbildung 10.33.

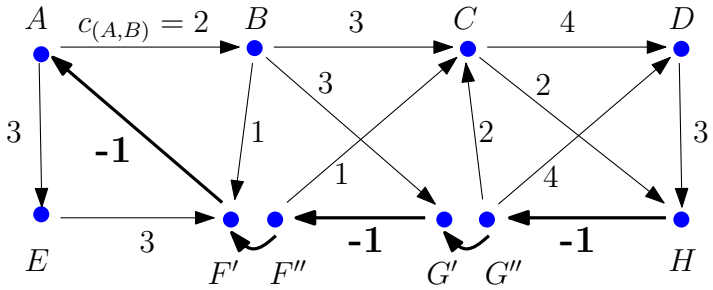


Abbildung 10.32: Modifizierter Graph G'.

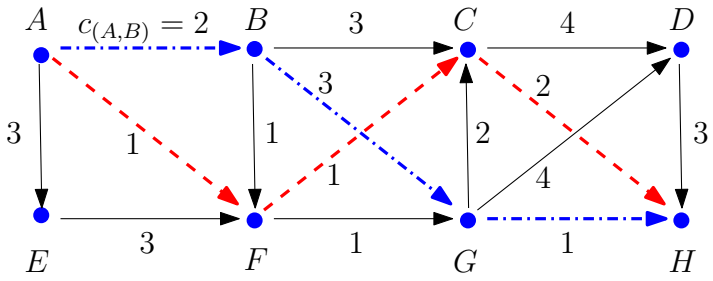


Abbildung 10.33: Kürzestes Paar knotendisjunkter Wege.

Der hier präsentierte Algorithmus stellt eine Modifikation des originalen Verfahrens von Suurballe dar. Der eigentliche Algorithmus von Suurballe beinhaltet das Dijkstra-Verfahren zur kürzeste-Wege-Findung, welches nichtnegative Kantengewichte fordert. In der hier vorgestellten, vereinfachten Variante, können bei der Umkehrung des kürzesten Weges $\bar{P}_1$ (Schritt 2.2 im Algorithmus negative Kantengewichte erzeugt werden. Suurballe reagiert auf diese Problematik durch eine geeignete Modifikation der Kantengewichte, siehe Übung.

10.5 Wellenlängenzuordnung

Ein Problem welches unmittelbar mit dem Routing verknüpft ist, ist das Problem der Wellenlängenzuordnung in Glasfasernetzen. Ein Glasfaserkabel ermöglicht das parallele Senden verschiedener Informationen durch Kodierung in verschiedenen Wellenlängen (Farben). Angenommen das Routing hat bereits stattgefunden, d.h. für jedes Bedarfspaar ist ein Weg fest vorgegeben. Diesem Weg muss nun eine Wellenlänge zugeordnet werden, d.h. das Bedarfspaar wird durch einen Lichtpfad verbunden. Entlang dieses Lichtpfades soll die Wellenlänge nicht geändert werden.[3] Um Konflikte auf Kanten zu vermeiden ist es nun wichtig, dass Lichtpfade, die gleiche Kanten nutzen, verschiedene Wellenlängen verwenden. Darüber hinaus ist die Anzahl der verschiedenen Wellenlängen in einem Netz technologisch begrenzt.
Die Aufgabe besteht nun darin, den Bedarspaaren Wellenlängen so zuzuordnen, dass

1. die maximale Zahl an Wellenlängen nicht überschritten wird und
2. auf keiner Kante Wellenlängenkonflikte entstehen.

Alternativ kann man das Problem mit der Zielvorgabe formulieren, die Anzahl der verschiedenen Wellenlängen im Netz zu minimieren. Eine erweiterte Variante des Problems bezieht das Routing in die Problematik mit ein (Routing- and Wavelength Assignement (RWA)).
In Kapitel 5 wurde das Knotenfärbungsproblem eingeführt. Die Aufgabe bestand darin, Knoten eines Graphen so zu färben, dass benachbarte Knoten verschiedene Farben tragen. Das Problem der Wellenlängenzuordnung lässt sich nun in das Knotenfärbungsproblem überführen, indem man einen Graphen generiert, der jedes Bedarfspaar durch einen Knoten repräsentiert und solche Knoten dann durch Kanten verbindet, wenn sich die entsprechenden Bedarfspaare

[3] Tatsächlich ist es möglich, Wellenlängen entlang eines Lichtpfades mittels Wellenlängenkonverter zu ändern. Solche Geräte verursachen Kosten, die Fragestellung der geeigneten Platzierung führt zum Problem der Wellenlängenkonverter-Standortplanung. Wir nehmen hier an, dass keine Wellenlängenkonverter einzusetzen sind.

Kanten teilen. Im folgenden Algorithmus wird die Verwandtschaft der Wellenlängenzuordnung und des Graphenfärbens genutzt:

Algorithmus: Wellenlängenzuordnung: Lösung mittels des Knotenfärbungsproblems

Voraussetzung: Jedem Bedarfspaar ist genau ein Weg zugeordnet.

Schritt 1. Erzeuge einen Hilfsgraphen:
1. Erzeuge einen Knoten für jedes Bedarfspaar.
2. Verbinde zwei Knoten durch eine Kante, wenn sich die Wege der zugeörigen Bedarfspaare mindestens eine Kante im Originalgraphen teilen.

Schritt 2. Finde eine minimale Knotenfärbung des Hilfsgraphen.

Schritt 3. Überführe die Lösung aus dem Hilfsgraphen in den Originalgraphen durch eineindeutige Zuordnung von Knotenfarben im Hilfsgraphen zu Wellenlängen der zugeörigen Bedarfspaare im Originalgraphen.

Ein Beispiel hierzu ist auf dem Übungsblatt zu finden.

Literaturverzeichnis

[Bha94] R. Bhandari. Optimal diverse routing in telecommunication fiber networks. IEEE INFOCOM, pages 1498–1508, 1994.

[CFR+09] M. Caserta, A. Fink, A. Raiconi, S. Schwarze, and S. Voss. Mathematical formulations and metaheuristics comparison for the push-tree problem. In J.W. Chinneck, B. Kristjansson, and M. Saltzman, editors, Operations Research and Cyber-infrastructure, pages 253–278, New York, 2009. Springer.

[Ham95] H.W. Hamacher. Mathematische Lösungsverfahren für planare Standortprobleme. Vieweg, Braunschweig, 1995.

[HN98] H. W. Hamacher and Nickel. Classification of location models. Locations Science, 6:229–242, 1998.

[HW04] F. Havet and M. Wennink. The push tree problem. Networks, 44:281–291, 2004.

[LMW88] R.F. Love, J.G. Morris, and G.O. Wesolowsky. Facilities Location, chapter 3.3, pages 51–60. North-Holland, Amsterdam, 1988.

[MM98] H. Müller-Merbach. Operations Research. Vahlens Handbücher der Wirtschafts- und Sozialwissenschaften. Verlag Vahlen, 1998.

[SKSKB06] D. Skorin-Kapov, J. Skorin-Kapov, and V. Boljunčić. Location problems in telecommunications. In M.G.C. Resende and P.M. Pardalos, editors, Handbook of Optimization in Telecommunications, pages 517–544. Springer, New York, 2006.

[Suu74] J.W. Suurballe. Disjoint paths in a network. Networks, 4:125–145, 1974.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the steinerproblem in graphs. Math. Japonica, 24:573–577, 1980.