Algorithmen und Wahrscheinlichkeiten

Danny Camenisch (dcamenisch)

March 15, 2026

Contents

List of Algorithms

Chapter 1

Template

1.1 tcolorbox

Definition

Definition...

Lemma

Lemma...

Satz von Danny

Satz...

My Heading

This is a **tcolorbox**.

Here, you see the lower part of the box.

subtitle

1.2 minted

Part I

Graphentheorie

Chapter 2

Zusammenhang

Wir erinnern uns an die Definition eines **zusammenhängenden** Graphen:

Definition

Ein Graph $G = (V, E)$ heisst **zusammenhängend**, wenn $\forall u, v \in V, u \neq v$ ein Pfad von u nach v in G existiert.

X heisst **u-v-Seperator**, wenn u und v in verschiedenen Zusammenhangskomponenten von $G[V \backslash X]$ liegen.

Diese Definition besagt zwar ob ein Graph zusammenhängend ist oder nicht, man kann aber nichts darüber sagen, wie stark ein Graph zusammenhängend ist. Dafür definieren wir sowohl Knoten- als auch Kantenzusammenhang:

Definition

Ein Graph $G = (V, E)$ heisst **k-zusammenhängend**, falls $|V| \geq k + 1$ und $\forall X \subseteq V$ mit $|X| < k$ gilt: $G[V \backslash X]$ ist zusammenhängend.

Ein Graph $G = (V, E)$ heisst **k-kanten-zusammenhängend**, falls $\forall X \subseteq E$ mit $|X| < k$ gilt: $(V, E \backslash X)$ ist zusammenhängend.

Ein Graph der 3-zusammenhängend ist, besitzt keine Seperator der Grösse 2 und ist dadurch auch 2-zusammenhängend.

Anschaulich zu merken: Wie viele Knoten oder Kanten muss man mindestens entfernen, bis der Graph nicht mehr zusammenhängend ist.

Lemma

Knoten-Zusammenhang $\leq$ Kanten-Zusammenhang $\leq$ minimaler Knotengrad

Ein (teils schwer zu beweisender) Satz erlaubt uns, eine äquivalente Definition von Zusammenhang verwenden:

Satz von Menger

Sei $G = (V, E)$ ein Graph. Dann gilt:

1. G ist k-zusammenhängend $\Leftrightarrow \forall u, v \in V, u \neq v$ gibt es mindestens k intern-knotendisjunkte u-v-Pfade.
2. G ist k-kanten-zusammenhängend $\Leftrightarrow \forall u, v \in V, u \neq v$ gibt es mindestens k intern-kantendisjunkte u-v-Pfade.

Chapter 3

Artikulationsknoten und Brücken

Definition

Sei $G = (V, E)$ ein zusammenhängender Graph. Ein Knoten $v \in V$ heisst **Artikulationsknoten** (eng. cut vertex) genau dann wenn $G[V \backslash \{v\}]$ nicht zusammenhängend ist.

Definition

Sei $G = (V, E)$ ein zusammenhängender Graph. Eine Kante $e \in E$ heisst **Brücke** (eng. cut edge) genau dann wenn $G[E \backslash \{e\}]$ nicht zusammenhängend ist.

Lemma

Sei $G = (V, E)$ ein zusammenhängender Graph. Ist $\{x, y\} \in E$ eine Brücke so gilt für x (und analog auch für y):

$$deg(x) = 1 \text{ oder } x \text{ ist ein Artikulationsknoten}$$

Die Umkehrung ist hier aber nicht immer wahr!

Wie können wir nun solche Artikulationsknoten und Brücken finden? Dazu können wir eine leicht modifizierte Version der bereits bekannten Tiefensuche verwenden.

Dazu müssen wir einige Beobachtungen machen. Zuerst, der von uns gewählte Startknoten s der Tiefensuche, und, für uns noch viel wichtiger, die Wurzel des Tiefensuchbaums T. Es wird schnell klar: Wenn s im Tiefensuchbaum eine Grad von mindestens 2 hat, so ist s ein Artikulationsknoten.

Für jeden Knoten $v \neq s$ ist allerdings nicht so offensichtlich, ob v ein Artikulationsknoten ist oder nicht. Wieder können wir uns den Tiefensuchbaum zunutzen machen: v ist ein Artikulationsknoten genau dann, wenn v im Baum Children (oder ganze Subtrees) besitzt, welche keine nicht-Baum-Kante zu einem Vorgänger von v besitzen.

Um diese Voraussetzung zu überprüfen, definieren wir für unseren Graphen alle Kanten des Tiefensuchbaums als *Vorwärtskanten*, alle anderen als *Rückwärtskanten*. Zusätzlich richten wir diese Kanten wie im Baum (für Vorwärtskanten) bzw. vom höheren zum kleineren DFS Wert für Rückwärtskanten. Weiter definieren wir für jeden Knoten $v \in V$:

low[v] := kleinste DFS-Nummer, die von v aus durch einen Pfad mit beliebig vielen Vorwärtskanten und maximale einer Rückwärtskante erreicht werden kann.

$$\text{low}[v] = \min\left(\text{dfs}[v], \min_{(v,w)\in E} \begin{cases} \text{dfs}[w], \text{ falls } (v,w) \text{ Restkante} \\ \text{low}[w], \text{ falls } (v,w) \text{ Baumkante} \end{cases}\right)$$

Dann erhalten wir als Bedingung für Artikulationsknoten:

$$\text{v ist Artikulationsknoten} \Leftrightarrow \begin{array}{l} v = sv \text{ hat in } T \text{ Grad } \geq 2 \\ v \neq s \text{ und } \exists w \in V \text{ mit } \{v,w\} \in E(T) \wedge \text{ LOW[w]} \geq \text{DFS[v]} \end{array}$$

Wenden wir unser Wissen über Brücken an, so können wir diese im gleichen Zug berechnen: Eine gerichtete Kante v, w des Tiefensuchbaums (nur diese kommen in Frage) ist eine Brücke genau dann wenn $\text{low}[w] > \text{dfs}[v]$. Restkanten sind niemals Brücken.

Unser angepasster DFS-Algorithmus sieht dann wie folgt aus, wobei als Input der Graph G und der Startknoten s verlangt wird:

Algorithm 1 DFS(G,s)

```
1: ∀v ∈ V : dfs[v] ← 0
2: num ← 0                                   ▷ Anzahl besuchter Knoten
3: T ← ∅                                     ▷ Menge der Kanten im Tiefensuchbaum
4: DFS-Visit(G,s)
5: if s hat in T mindestens Grad 2 then
6:     isArtVert[s] ← TRUE
7: else
8:     isArtVert[s] ← FALSE
9: end if
```

Algorithm 2 DFS-Visit(G,v)

```
1:  num ← num +1
2:  dfs[v] ← num
3:  low[v] ← dfs[v]
4:  isArtVert[v] ← FALSE
5:  for (v, w) ∈ E do
6:      if dfs[w] = 0 then
7:          T ← T + (v, w)
8:          val ← DFS-Visit(G,w)              ▷ low-Wert des direkten Nachfolgers
9:          if val ≥ dfs[v] then
10:             isArtVert[v] ← TRUE
11:             if val > dfs[v] then
12:                 isBridge[v][w] ← TRUE
13:             end if
14:         end if
15:         low[v] ← min{low[v], val}
16:     else                                  ▷ Also ist dfs[w] ≠ 0
17:         low[v] ← min{low[v], dfs[w]}
18:     end if
19: end for
20: return low[v]
```

Satz

In einem zusammenhängenden Graphen $G = (V, E)$, der als Adjazenzlist gespeichert ist, lassen sich alle Artikulationsknoten und Brücken in Zeit $\mathcal{O}(|E|)$ berechnen.

Chapter 4

Block-Graphen

4.1 Blöcke

Definition

Sei $G = (V, E)$. Wir definieren eine Äquivalenzrelation auf E wie folgt:

$$e \sim f :\Leftrightarrow e = f \text{ oder } \exists \text{ Kreis durch } e \text{ und } f$$

Diese Äquivalenzklassen nennen wir auch **Blöcke**. Eine alternative Definition lautet wiefolgt:

Definition

Sei $G = (V, E)$ ein zusammenhängender Graph. Ein **Block** ist eine maximale Menge von Kanten, so dass je zwei dieser Kanten auf einem gemeinsamen Kreis liegen.

Merke: Ein Block ist ein Subgraph, der 2-zusammenhängend ist.

Lemma

Zwei Blöcke schneiden sich - wenn überhaupt - immer in einem Artikulationsknoten.

Wir erinnern uns an die Definition von bipartiten Graphen.

Definition

Ein Graph ist **bipartit**, wenn sich die Knotenmenge in zwei disjunkte Mengen A und B zerlegen lässt, sodass Kanten von G nur zwischen A und B verlaufen. Wir verwenden dafür folgende Notation:

$$V = A \uplus B$$

4.2 Block-Graphen

Ein Block-Graph ist nun wiefolgt definitert:

Definition

Der Block-Graph von G ist der bipartite Graph $T = (A \uplus B, E_T)$ mit

- $A =$ Artikulationsknoten von G
- $B =$ Blöcke von G
- $\forall a \in A, b \in B : \{a, b\} \in E_T \Leftrightarrow a$ inzident zu einer Kante in b

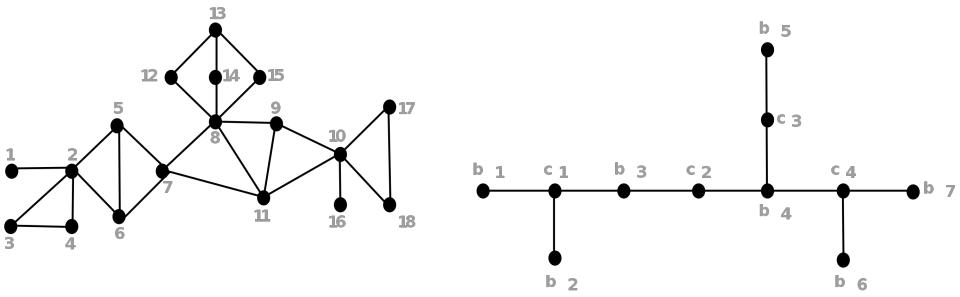

Figure 4.1: Beispiel eines Graphen (links) und dem dazugehörigen Block-Graphen (rechts)

Chapter 5

Kreise

5.1 Eulertour

Definition

Sei $G = (V, E)$ ein Graph. Eine **Eulertour** ist ein geschlossener Weg, der jede Kante genau einmal enthält. Enthält ein Graph eine Eulertour, so nennt man ihn **eulersch**.

Eine solche Tour lässt sich in $\mathcal{O}(|E|)$ finden - der Algorithmus ("schneller und langsamer Läufer") lässt sich ungefähr wie folgt beschreiben:

Algorithm 3 Eulertour(G,s)

```
1:  W ← RandomTour(s)                          ▷ "Läufer"
2:  v_slow ← Startknoten von W                 ▷ "Schildkröte"
3:  while v_slow ist nicht der letzte Knoten in W do
4:      v ← Nachfolger von v_slow in W
5:      if ∃(v, w) ∈ E, w ∉ W then             ▷ Ungenutze Kanten ab v
6:          W' ← RandomTour(s)
7:          W ← W_1 + W' + W_2
8:      end if
9:      v_slow ← Nachfolger von v_slow in W
10: end whilereturn W
```

Algorithm 4 Randomtour(s)

```
1:  v ← s
2:  W ← {v}
3:  while ∃(v, w) ∈ E, w ∉ W do                ▷ Ungenutze Kanten ab v
4:      Wähle beliebigen Nachfolger v_next
5:      Hänge v_next an W analog
6:      e ← {v, v_next}
7:      Lösche e aus G
8:      v ← v_next
9:  end while
10: return W
```

Die Laufzeit ist leicht erklärt: Wir betrachten jede Kante genau einmal und "löschen" sie danach.

Satz

Ein zusammenhängender Graph $G = (V, E)$ enthält eine Eulertour $\Leftrightarrow$ der Grad jedes Knoten $v \in V$ ist gerade.

5.2 Hamiltonkreis

Definition

Sei $G = (V, E)$ ein Graph. Ein **Hamiltonkreis** ist ein Kreis, der alle Knoten von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn **hamiltonisch**.

Ein klassisches Anwendungsbeispiel ist das Traveling Salesman Problem. Es wird vermutet, dass es keinen Algorithmus gibt, der in polynomieller Zeit bestimmt, ob es einen Hamiltonkreis in einem gegebenen Graphen gibt.

Es gibt aber einige Spezialfälle, für die es deutlich leichter ist, die Existenz eines Hamiltonkreises zu bestimmen:

- Ein $n \times m$ Gitter enthält einen Hamiltonkreis genau dann wenn $n * m$ gerade ist
- Ein d-dimensionaler Hyperwürfel H_d (Knotenmenge: $\{0, 1\}^d$, Kantenmenge: Alle Knotenpaare, welche sich in genau einer Koordinate unterscheiden) enthält einen Hamiltonkreis (auch für Dimensionen $d \geq 4$)

Satz von Dirac

Jeder Graph $G = (V, E)$ mit $|V| \geq 3$ und Minimalgrad $\delta(G) \geq |V|/2$ enthält einen Hamiltonkreis.

Wenn wir einen Graphen anschauen, der nicht in einer dieser Kategorien fällt, wird es schwieriger um zu entscheiden, ob der Graph einen Hamiltonkreis enthält. Wenn wir das Problem mit Brute Force angehen, dauert es $\mathcal{O}(n!)$. Nun wollen wir uns zwei Algorithmen anschauen, die etwas effizienter sind.

Algorithm 5 Hamiltonkreis(G)

for $v \in [n]$ mit $v \neq 1$ **do** ▷ Initialisierung
    $P_{(1,x),x} = \begin{cases} 1, & \text{falls } (1,x) \in E \\ 0, & \text{sonst} \end{cases}$
end for
for $s = 3$ bis n **do** ▷ Initialisierung
    for $S \subseteq [n]$ mit $1 \in S$ und $|S| = s$ **do**
        for $x \in S, x \neq 1$ **do**
            $P_{s,x} = \max\{P_{S\setminus x,y} : y \in S \cap N(x), y \neq 1\}$
        end for
    end for
end for
if $\exists x \in N(1)$ mit $P_{[n],x} = 1$ **then** ▷ Ausgabe
    return G enthält einen Hamiltonkreis
else
    return G enthält keinen Hamiltonkreis
end if

Dieser Algorithmus kann in $\mathcal{O}(|V|^2 \cdot 2^{|V|})$ berechnen ob ein Hamiltonkreis existiert. Jedoch braucht er dafür Speicherplatz in $\mathcal{O}(|V| \cdot 2^{|V|})$. Dies ist nicht optimal, daher gibt es einen zweiten Algorithmus, der nicht nur die Existenz von Hamiltonkreisen feststellt, sondern auch gleich die Anzahl der Hamiltonkreise berechnet.

Definition

W_s = Anzahl der Wege der Länge n von s nach s die keinen Knoten in $S \subset V$ besuchen.

Algorithm 6 Zähle-Hamiltonkreis(G)

$s = 1$ ▷ Wahl des Startknoten (willkürlich)
$Z = |W_\emptyset|$
for **do**
    Berechne $|W_S|$ ▷ mit der Adjazenzmatrix von $G[V\setminus S]$
    $Z = Z + (-1)^{|S|} \cdot |W_S|$
end for
$Z = Z/2$
return Z ▷ Anzahl der Hamiltonkreise in G

Dieser zweite Algorithmus hat eine Laufzeit von $\mathcal{O}(|V|^{2.81} \cdot \log(|V|) \cdot 2^{|V|})$, braucht dafür aber nur Speicherplatz in $\mathcal{O}(|V|^2)$, weshalb dieser Algorithmus in der Praxis meist besser ist.

Wir bemerken den drastischen Laufzeitunterschied gegenüber der Eulertour: Dort benötigten wir gerade einmal $\mathcal{O}(|E|)$ Zeit, um eine solche zu finden, während das Hamiltonkreisproblem nicht in polynomieller Zeit lösbar ist, da es NP-vollständig ist.

5.3 NP-Probleme

Definition

Ein NP-Problem, ist ein Problem, dass nicht in polynomieller Laufzeit gelöst werden kann (man nimmt es mindestens an). Das Gegenteil dazu sind P-Probleme.

5.4 Traveling Salesman

Das Traveling Salesman Problem ist ein sehr berühmtes Problem:

Definition

Gegeben sei ein vollständiger Graph mit n Knoten, berechne die kürzeste Rundreise.

Dies bedeutet effektiv, dass wir den minimalen Hamiltonkreis suchen. Das TSP ist stark mit dem Hamiltonkreisproblem verwandt, aber wir haben nun statt einer binären Antwort, viele Antwortmöglichkeiten. Wir können aber aus dem TSP ein Optimierungsproblem machen. Bezeichned $\text{opt}(K_n, l)$ die optimale Lösung für einen Graphen K_n mit Gewichtsfunktion l, so sprechen wir von einem α-Approximationsalgorithmus, falls dieser immer eine Lösung C mit

$$\sum_{e \in C} l(e) \leq \alpha \cdot \text{opt}(K_n, l)$$

findet. Allerdings sind wir so immernoch sehr eingeschränkt, denn wenn wir einen Graphen so konstruieren, dass wir das Hamiltonkreisproblem lösen : Alle Kanten, die in einem anderen Graph (der zu untersuchen ist) vorkommen, haben Gewicht 0. Da $\alpha \cdot 0 = 0$ müsste ein α-Approximation immer einen Hamiltonkreis in seiner Laufzeit finden.

Mithilfe einer zusätzlichen Annahme, können wir das TSP in aktzeptabler Laufzeit approximieren: Wir erwarten, dass die Kantenlänge die *Dreiecksungleichung* erfüllen, dass sich also Umwege nie lohnen würden. Wir nenne dies, dass metrische TSP.

Satz

Für das metrische TSP gibt es einen 2-Approximationsalgorithmus mit Laufzeit $\mathcal{O}(n^2)$.

Wir erreichen dies wie folgt: Wir berechne zuerst den MST des Graphen in $\mathcal{O}(n^2)$. Nun verdoppeln wir alle Kanten - dadurch erhalten wir einen Weg von Länge 2· MST. Nun finden wir eine Eulertour, und durchlaufen diese erneut, wobei wir 'Abkürzungen' verwenden, also keinen bereits besuchten Knoten erneut besuchen. Aufgrund der Dreiecksungleichung wird unser Weg dadruch garantiert nicht länger. Da aus einem Hamiltonkreis durch entfernen einer Kante ein Spannbaum wird, muss offensichtlich gelten dass

$$\text{MST}(K_n, l) \leq \text{opt}(K_n, l)$$

Insbesondere muss also die Länge unseres gefundenen Hamiltonkreises kleiner sein als $2 \cdot \text{MST}$, und damit kleiner als $2 \cdot \text{opt}$, was ja genau zu zeigen war.

Chapter 6

Matching

Viele Zuordnungsprobleme lassen sich als ein Graphenproblem modellieren. Nehmen wir etwa an, wir müssen Jobs auf Rechner verteilen, aber nicht alle Rechner können alle Jobs ausführen. Modellieren wir dies durch einen Graphen, wobei Jobs und Rechner durch Knoten dargestellt sind, und Kanten zwischen Job und Rechner bedeuten das Erfüllen der Vorraussetzungen für den Job. Nun suchen wir eine Zuordnung, die möglichst jedem Job einen Rechner und jedem Rechner einen Job (der ausgeführt werden kann) zuordnet.

Definition

Eine Kantenmenge $M \subseteq E$ heisst **Matching** in einem Graphen $G = (V, E)$, falls kein Knoten des Graphen zu mehr als einer Kante aus M inzident ist, oder formal ausgedrückt, wenn

$$e \cap f = \emptyset \text{ für alle } e, f \in M \text{ mit } e \neq f$$

Ein Knoten v wir von M **überdeckt**, falls eine Kante $e \in M$ gibt, die v enthält.

Ein Matching M heisst **perfektes Matching**, wenn jeder Knoten durch genau eine Kante aus M überdeckt wird, oder, anders ausgedrückt, wenn $|M| = \frac{|V|}{2}$.

Wir intressieren uns oft dafür, ein möglichst grosses Matching zu finden. Hierfür müssen wir definieren, was ein möglichst grosses Matching ist:

Definition

Sei $G = (V, E)$ ein Graph mit einem Matching M.

- M heisst **inklusionsmaximal**, wenn es kein Matching M' mit $M \subseteq M'$ und $|M| < |M'|$ gibt.
- M heisst **kardinalitätsmaximal** (oder nur maximal), wenn es kein Matching mit $|M| < |M'|$ gibt.

Ein kardinalitätsmaximales Matching ist auch immer inklusionsmaximal, aber nicht umgekehrt. Weiter gilt, $|M_{inc}| \geq |M_{max}|/2$. Haben wir ein kardinalitätsmaximal Matching mit $M = |V|/2$, überdeckt es also alle Knoten, so nenne wir es ein **perfektes Matching**.

6.1 Matching-Algorithmen

Wir schauen uns nun ein paar Algorithmen an, mitwelchen wir Matchings finden können.

Algorithm 7 Greedy-Matching(G)

1: $M \leftarrow \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: wähle eine beliebige Kante $e \in E$
4: $M \leftarrow M \cup \{e\}$
5: lösche e und alle inzidenten Kanten in G
6: **end while**
7: **return** M

Mit dem Greedy-Algorithmus kann in $\mathcal{O}(|E|)$ ein Matching bestimmt werden. Es gilt $|M_{Greedy}| \geq |M_{max}|/2$, wobei M_{max} ein kardinalitätsmaximales Matching ist.

Solch ein Greedy-Matching muss nicht umbedingt ein kardinalitätsmaximales Matching sein. Wollen wir solche eines finden, dann brauchen wir einen anderen Algorithmus. Dafür führen wir den Begriff des **augmentierenden Pfad** ein.

Definition

Für ein Matching M, nennen wir eine Pfad M**-augmentierender**, wenn er abwechselnd Kanten aus M und nicht aus M enthält, und in (unterschiedlichen) Knoten die nicht von M überdeckt werden startet und endet.

Wenn wir solch einen M-augmentierenden Pfad P haben, können wir durch tauschen der Zugehörigkeit aller Kanten im Pfad das Matching vergrössern. Wir notieren das tauschen/augmentieren als $M \oplus P$.

Satz von Berger

Jedes Matching, das nicht kardinalitätsmaximal ist, besitz einen augmentierenden Pfad.

Nun können wir einen simplen Algorithmus verwenden, um solch ein grösseres Matching zu erhalten. Wir schauen uns hier nur den Algorithmus für bipartite Graphen an.

Algorithm 8 Bipartite-Matching(G)

1: $M \leftarrow \emptyset$
2: **repeat**
3: $P \leftarrow$ Augmenting-Path(G,M)
4: $M \leftarrow M \oplus P$
5: **until** $P = \emptyset$
6: **return** M

Das Finden des augmentierenden Pfades erreichen wir durch modifizierte Breitensuche:

Algorithm 9 Augmenting-Path(G = $(A \uplus B, E)$, M)

1: $L_0 \leftarrow$ {unüberdeckte Knoten aus A} und markiere L_0 als besucht
2: **for** i = 1...n **do**
3: **if** i ungerade **then**
4: $L_i \leftarrow$ {unbesuchte Nachbaren von L_{i-1} via Kanten in $E \setminus M$ }
5: **else**
6: $L_i \leftarrow$ {unbesuchte Nachbaren von L_{i-1} via Kanten in M }
7: **end if**
8: markiere Knoten aus L_i als besucht
9: **if** ein Knoten v in L_i nicht überdeckt ist **then**
10: **return** Pfad von L_0 nach v (backtracking)
11: **end if**
12: **end for**

Da BFS in $\mathcal{O}(|V| + |E|)$ läuft, und unser Matching maximal eine Grösse von $\frac{n}{2} = \mathcal{O}(n)$ hat, läuft unser Algorithmus in $\mathcal{O}(|V| \cdot |E|)$. Es ist einfach zu sehen, dass der kürzeste Pfad von L_0 nach v die Länge i hat.

Wir können diesen Algorithmus aber noch etwas optimieren, so dass er in $\mathcal{O}(\sqrt{|V|} \cdot (|V| + |E|))$ läuft. Anstatt den erstbesten Pfad zurückzugeben, suchen wir knotendisjunkte Pfade alle von unüberdeckten v in L_i nach L_0.

Algorithm 10 Hopcroft-Karp(G)

1: $M \leftarrow \emptyset$
2: **repeat**
3: $k \leftarrow$ länge eines kürzeste augmentierenden Pfades
4: Finde alle disjunkte augmentierende Pfade S der Länge k
5: **for** Pfad P in S **do**
6: $M \leftarrow M \oplus P$
7: **end for**
8: **until** $P = \emptyset$
9: **return** M

(*Beweis siehe Vorlesung*)

Mit viel mehr Aufwand (randomisierte Algorithmen und lineare Programmierung) lassen sich andere effiziente Algorithmen finden, die schneller und allgemeiner sind:

- $\mathcal{O}(|V|^{1/2} \cdot |E|)$ für allgemeine Graphen
- $\mathcal{O}(|V|^3)$ für gewichtete Graphen

6.2 1.5-Approximation des TSP

Der letztere Algorithmus ist insofern interessant, da wir dadurch eine 1.5-Approximation für das metrische TSP finden können. Dies funktioniert ähnlich zur 2-Approximation:

1. Berechne einen MST T in G
2. $U :=$ {Knoten in T mit ungeradem Grad}, berechne ein perfektes Matching M von U mit minimalem Gewicht

3. Füge M zu T hinzu, es entsteht ein Multigraph T'
4. Berechne eine Eulertour E in T'
5. Kürze E ab: überspringe Knoten die zum wiederholten mal besucht werden.

Satz

Christofides' Algorithmus berechnet in Zeit $\mathcal{O}(|V|^3)$ eine 1.5-Approximation für das metrische TSP.

Seit September 2020 ist es möglich eine noch bessere Approximation zu finden (1.499999999999999999999999999999999990-Approximation). Dieser Durchbruch wird in den kommenden Jahren für viele Verbesserungen sorgen.

6.3 Satz von Hall

Definition

Ein Graph ist **regulär**, wenn alle Knoten den selben Grad haben.

Satz von Hall (Heiratssatz)

Ein bipartiter Graph $G = (A \uplus B, E)$ enthält ein Matching M der Kardinalität $|M| = |A|$, genau dann wenn $\forall X \subseteq A : |X| \leq |N(X)|$.

Als direktes Korollar gibt es den Satz von Frobenius:

Satz von Frobenius

Für alle k gilt: jeder k-reguläre, bipartite Graph enthält ein perfektes Matching.

Dieses perfekte Matching lässt sich in Zeit $\mathcal{O}(|E|)$ bestimmen. Für $k = 2^x$ ist dies leicht einzusehen: Wir bestimmen für jede Zusammenhangskomponente eine Eulertour. Dann entfernen wir jede zweite Kante davon, und erhalten einen 2^{x-1} regulären Graphen. Nach x Iterationen ist der Graph $2^0 = 1$ regulär, d.h. wir haben ein perfektes Matching gefunden.

Dies kann verallgemeinert werden, so dass es auch für nicht Zweierpotenzen funktioniert. Dies lassen wir hier aber aus.

Chapter 7

Färbungen

Viele Probleme in der Graphentheorie können wir durch eine Partition der Knotenmenge lösen. Ein solches Problem ist zum Beispiel, dass erstellen eines Prüfungsplans - es wir eine Färbung gesucht, so dass keine zwei benachbarten Knoten die gleiche Farbe haben.

Definition

Eine **(Knoten-)Färbung** eines Graphen $G = (V, E)$ mit k Farben ist eine Abbildung $c : V \to [k]$, so dass gilt

$$c(u) \neq c(v) \text{ für alle Kanten } \{u, v\} \in E$$

Die **chromatische Zahl** $\chi(G)$ ist die minimale Anzahl Farben, die für eine Färbung von G benötigt wird.

Ein klassisches Graphfärbungsproblem, welches einen herausragend komplizierten Beweis erfordert, ist das 4-Farben-Theorem, welches besagt, dass jede Landkarte mit 4-Farben gefärbt werden kann, so dass jeweils benachbarte Länder unterschiedlich gefärbt sind. Im Allgemeinen gilt, dass ein planarer Graph immer mit 4 Farben färbbar ist.

Graphen mit chromatischer Zahl k nennt man auch k-partit, denn es gilt:

$$\chi(G) \leq k \Leftrightarrow G \text{ ist } k\text{-partit}$$

Besonders Interessant ist hier der Fall $k = 2$, also ein bipartiter Graph. Ob ein Graph bipartit ist, lässt sich in $\mathcal{O}(|V| + |E|)$ bestimmen (DFS / BFS).

Satz

Für alle $k \geq 3$ ist das Problem,

$$\text{Gegeben ein Graph } G = (V, E), \text{ gilt } \chi(G) \leq k?$$

NP-vollständig.

Färbbarkeit ist so schwer, da gilt $\forall k, r \in \mathbb{N}$ gibt es einen Graphen **ohne** einen Kreis mit Länge $\leq k$, aber mit chromatischer Zahl $\geq r$.

Um eine Färbung zu finden, können wir wieder einen Greedy-Algorithmus benutzen.

Algorithm 11 Greedy-Färbung(G)

1: wähle eine beliebige Reihenfolge der Knoten in V
2: $c[v_1] \leftarrow 1$
3: **for** $i = 2$ to $i = n$ **do**
4: $\quad c[v_i] \leftarrow \min\{k \in \mathbb{N} | k \notin c(u)$ für alle $u \in N(v_i) \cap \{v_1, ..., v_{i-1}\}\}$
5: **end for**

Für jede Reihenfolge der Knoten, benötigt der Greedy-Algorithmus höchstens $\Delta(G)+1$ viele Farben. Jedoch gibt es eine Reihenfolge, so dass der Greedy-Algorithmus nur $\chi(G)$ viele Farben benötigt. Mit etwas Heuristik kommen wir darauf, dass wenn wir die Knoten zuerst absteigen nach dem Knotengrad ordnen, wir höchstens $k+1$ viele Farben brauchen. Dafür brauchen wir auch nur $\mathcal{O}(|E|)$ Zeit. Wir wissen aber, dass wir dies nochmal auf k viele Farben verbessern können.

Satz von Brooks

Einen zusammenhängenden Graphen G für den gilt $G \neq K_n, G \neq C_{2n+1}$, kann in $\mathcal{O}(|E|)$ Zeit eine Färbung mit $\Delta(G)$ vielen Farben gefunden werden.

Für den genauen Algorithmus siehe Slides 8a

Damit können wir nun zeigen, wie wir in $\mathcal{O}(|E|)$ eine Färbung mit $\mathcal{O}(\sqrt{n})$ für einen 3-färbbaren Graphen finden können: Wir färben zuerst alle Knoten mit $\deg(v) \leq \sqrt{n}$ und deren Nachbarn. Dafür benötigen wir maximal drei Farben in jedem Schritt, also maximal $3\sqrt{n}$ Farben insgesamt. Wir entfernen daraufhin alle gefärbten Knoten. Sobald wir alle Knoten dementsprechend abgearbeitet haben, färben wir den Rest unseres Graphen mit dem Greedy-Algorithmus. Dafür benötigen wir $\sqrt{n}$ Farben, also total $\mathcal{O}(\sqrt{n})$.

Part II

Wahrscheinlichkeiten

Chapter 8

Grundlagen

Wahrscheinlichkeiten spielen eine wichtige Rolle in der Informatik. Viele Algorithmen beruhen auf Zufall (z.Bsp. QuickSort oder Hashing) und jegliche Art von Kryptographie währe ohne Stochastik unmöglich. In diesem ersten Kapitel der Wahrscheinlichkeiten definieren wir ein paar grundlegende Begriffe und Notationen.

Definition

Ein **diskreter Wahrscheinlichkeitsraum** ist durch eine endliche oder zumindest abzählbare **Ereignismenge** $\Omega = \{\omega_1, \omega_2, ...\}$ von **Elementarereignissen**. Jedem Elementarereignis ω_i ist eine (Elementar)-Wahrscheinlichkeiten $\Pr[\omega_i]$ zugeordnet. Dabei gilt $0 \leq \Pr[\omega_i] \leq 1$ und

$$\sum_{\omega \in \Omega} \Pr[\omega] = 1$$

Eine Menge $E \subseteq \Omega$ heisst **Ereignis**. Die Wahrscheinlichkeiten eines Ereignisses ist definiert durch die Summe seiner Elementarereignissen. Weiter bezeichnen wir mit $\bar{E} = \Omega \backslash E$ das **Komplementärereignis** zu E.

Aus der Definition des Wahrscheinlichkeitsraums folgen einige triviale, aber sehr nützliche Konsequenzen:

Satz

Für Ereignisse $A_1, B_1, A_2, B_2, ...$ gilt:

- $\Pr[\emptyset] = 0$, $\Pr[\Omega] = 1$
- $0 \leq \Pr[A] \leq 1$
- $\Pr[\bar{A}] = 1 - \Pr[A]$
- $A \subseteq B \Rightarrow \Pr[A] \leq \Pr[B]$

Ausserdem lässt sich der erste wichtige Satz für den Umgang mit mehreren Ereignissen formulieren:

Additionssatz

Falls die Ereignisse $A_1, A_2, ..., A_n$ paarweise disjunkt sind, so folgt:

$$\Pr[\bigcup_{i=1}^{n} A_i] = \sum_{i=1}^{n} \Pr[A_i]$$

Dies gilt nur für disjunkte Ereignisse. Im Allgemeinen gilt:

Boolesche Ungleichung / Union Bound

Für Ereignisse Ereignisse $A_1, A_2, ..., A_n$ gilt:

$$\Pr[\bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i]$$

Wir können für die genaue Berechnung der Vereinigung von Ereignissen die **Siebformel** anwenden. Wie legen wir die Wahrscheinlichkeiten für Elementarereignissen sinnvoll fest? Generell gilt das **Prinzip von Laplace**, d.h. wenn nichts dagegen spricht, gehen wir davon aus, dass alle Elementarereignissen die gleich Wahrscheinlichkeit haben. Ein Wahrscheinlichkeitsraum, indem alle Elementarereignissen die gleiche Wahrscheinlichkeit haben, wird deshalb auch **Laplace-Raum** genannt. Als Beispiel für einen Laplace-Raum ist der Würfelwurf oder das Ziehen einer Karte aus einem Kartenspiel zu betrachten. An dieser Stelle lohnt es sich, den **Binomialkoeffizienten** einzuführen. Dies ist die Kurzfassung für:

> Wie viele Möglichkeiten gibt es, aus n Elementen k Elemente (ohne zurücklegen) auszuwählen?

Den Binomialkoeffizienten schreiben wir als

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Im Allgemeinen sind folgende vier Formeln sehr hilfreich zu wissen:

	geordnet	ungeordnet
mit Zurücklegen	n^k	$\binom{n+k-1}{k}$
ohne Zurücklegen	$n^{\underline{k}}$	$\binom{n}{k}$

Chapter 9

Die Siebformel

Siebformel / Prinzip der Inklusion/Exklusion

Für endliche Mengen $A_1, A_2, ..., A_n$ mit $n \geq 2$ gilt:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{l=1}^{n} (-1)^{l+1} \sum_{1 \leq i_1 < ... < i_l \leq n} |A_{i_1} \cap ... \cap A_{i_l}|$$

In Worten: Alle Schnitte von einer *ungeraden* Anzahl von Ereignissen *addiert* man, alle Schnitte von einer *geraden* Anzahl von Ereignissen *substrahiert* man.

Insgesamt müssen wir $\mathcal{O}(2^n)$ Summanden addieren. Man kann sich diese Formel oft einfacher visualisieren als beschreiben. Auf folgender Grafik kann man sich vergewissern, dass jede der sieben Teilmengen genau einmal gezählt wird:

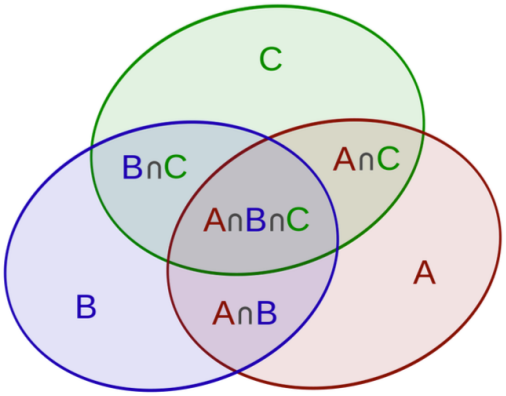

Figure 9.1: Beispiel Inklusion/Exklusion mit n = 3

$$\left|\bigcup_{i=1}^{3} A_i\right| = \sum_{l=1}^{3} |A_l| - |A_1 \cap A_2| - |A_2 \cap A_3| - |A_1 \cap A_3| + |A_1 \cap A_2 \cap A_3|$$

Chapter 10

Bedingte Wahrscheinlichkeiten

Durch das Bekanntwerden von zusätzlichen Informationen verändern sich die Wahrscheinlichkeiten. Manche Elementarereignisse werden wahrscheinlicher, andere unwahrscheinlicher oder sogar unmöglich.

Definition

A und B seien Ereignisse mit $\Pr[B] > 0$. Die **bedingte Wahrscheinlichkeit** $\Pr[A|B]$ (A gegeben B) ist definiert als:

$$\Pr[A|B] := \frac{\Pr[A \cap B]}{\Pr[B]}$$

Häufig wird die Definition der bedingten Wahrscheinlichkeit auch wie folgt benutzt:

$$\Pr[A \cap B] = \Pr[A|B] \cdot \Pr[B] = \Pr[B|A] \cdot \Pr[A]$$

Für mehrere Ereignisse führt dies zu folgender Rechenregel:

Multiplikationssatz

Gegeben Ereignisse $A_1, A_2, ..., A_n$. Falls $\Pr[A_1 \cap A_2 \cap ... \cap A_n] > 0$ ist, so gilt:

$$\Pr[A_1 \cap ... \cap A_n] = \Pr[A_1] \cdot \Pr[A_2|A_1] \cdot \Pr[A_3|A_1 \cap A_2] ... \Pr[A_n|A_1 \cap ... \cap A_{n-1}]$$

Dieser Satz kommt vorallem bei *Balls into Bins* Problemen zum gebrauch: Man werfe m Bälle in n Körbe, wie gross ist die Wahrscheinlichkeit, dass am Ende alle Bälle allein in einem Korb liegen?

Basierend auf der Definition der bedingten Wahrscheinlichkeit gibt es auch den folgenden Satz zu zeigen:

Satz der totalen Wahrscheinlichkeite

Die Ereignisse $A_1, A_2, ..., A_n$ seien paarweise disjunkt und es gelte $B \subseteq A_1 \cup A_2 \cup ... \cup A_n$. Dann gilt folgendes:

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B|A_i] \cdot \Pr[A_i]$$

Dieser Satz lässt sich relativ leicht begründen. Jeder einzelne Summand ist gleich $B \cup A_i$. Durch die Bedingungen von paarweiser Disjunktion und dass B Untermenge der Vereinigung von A sein muss, ergibt das Aufsummieren aller Summanden genau die Wahrscheinlichkeit der Menge B. Dieser Satz kann für das Monty-Hall Problem verwendet werden.

Als letzter Satz in diesem Kapitel schauen wir nun noch den Satz von Bayes an:

Satz von Bayes

Die Ereignisse $A_1, ..., A_n$ seien paarweise dijunkt und es gelte $B \subseteq A_1 \cup ... \cup A_n$ sowie $\Pr[B] > 0$. Dann gilt (für $1 \geq i \geq n$):

$$\Pr[A_i|B] = \frac{\Pr[A_i \cap B]}{\Pr[B]} = \frac{\Pr[B|\ A_i] \cdot \Pr[A_i]}{\sum_{j=1}^{n} \Pr[B|A_j] \cdot \Pr[A_j]}$$

Dieser Satz wird vorallem oft mit dem Beispiel von medizinischen Tests betrachtet. Definiert man die Ereignisse P = Test positiv, N = Test negativ, G = gesund und $K = krank$, so ist es oft einfach die Wahrscheinlichkeiten für $\Pr[P|K]$ und $\Pr[P|G]$ zu ermitteln. Was jedoch oftmals für die Patienten spannender ist, ist die Wahrscheinlichkeit $\Pr[K|P]$, hierfür können wir nun den Satz von Bayes verwenden.

10.1 Unabhängigkeit

Bei einer bedingten Wahrscheinlichkeit kann es vorkommen, dass die Bedingung keinen Einfluss auf das Ereignis hat.

Definition

Die Ereignisse A und B heissen **unabhängig**, falls gilt

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$

Falls $B \neq 0$, können wir zudem die Definition umformen zu:

$$\Pr[A] = \frac{\Pr[A \cap B]}{\Pr[B]} = \Pr[A|B]$$

Es ist wichtig zu bemerken, dass die stochastische Unabhängigkeit nicht umbedingt mit der physikalischen Unabhängigkeit gleichzusetzen ist.
Für die Unabhängigkeit von mehreren Ereignissen, gilt folgende Definition.

Definition

Die Ereignisse $A_1, A_2, ..., A_n$ heissen **unabhängig**, wenn für alle Teilmengen $I \subseteq \{1, ..., n\}$ mit $I = \{i_1, ..., i_k\}$ gilt, dass

$$\Pr[A_{i_1} \cap ... \cap A_{i_k}] = \Pr[A_{i_1}] \cdot ... \cdot \Pr[A_{i_k}]$$

Aus diesen Definitionen folgen diese Sätze:

Satz

Die Ereignisse $A_1, A_2, ..., A_n$ sind genau dann unabhängig, wenn für alle $(s_1, ..., s_n) \in \{0, 1\}^n$ gilt, dass

$$\Pr[A_1^{s_1}] \cap ... \cap \Pr[A_n^{s_n}] = \Pr[A_i^{s_1}] \cdot ... \cdot \Pr[A_n^{s_n}]$$

Wobei $A_i^0 = \bar{A}_i$ und $A_i^1 = A_i$.

In Worten: Bei der Unabhängigkeit von Ereignissen können wir stattdessen auch den Schnitt der Komplementärereignisse (und beliebige "Mischungen") betrachten.

Satz

Sind A, B und C unabhängige Ereignisse, so sind auch $A \cap B$ und C, bzw. $A \cup B$ und C unabhängig.

Chapter 11

Zufallsvariablen

Oft sind wir bei der Durchführung von Zufallsexperimenten an gewissen Ereignissen oder Merkmalen interessiert. Zum Beispiel an der Anzahl Kopf in drei Münzwürfen.

Definition

Eine **Zufallsvariable** ist eine Abbildung $X : \Omega \to \mathbb{R}$, wobei Ω die Ereignismenge eines Wahrscheinlichkeitsraums ist. Wir definieren die Kurzschreibweise für Zufallsvariablen wie folgt:

$$X \leq 5 := \{\omega \in \Omega | X(\omega) \leq 5\}$$

Diese Schreibweise steht für das Ereignis, dass die Zufallsvariable X einen Wert kleiner gleich 5 annimmt.

Wenn wir nun solch eine Zufallsvariable habe, können wir zwei wichtige Funktionen definieren:

Definition

Wir definieren die **Dichtefunktion** einer Zufallsvariablen X als

$$f_X : \mathbb{R} \to [0,1], x \mapsto \Pr[X = x]$$

Wir definieren die **Verteilungsfunktion** einer Zufallsvariablen X als

$$F_X : \mathbb{R} \to [0,1], x \mapsto \Pr[X \leq x'] = \sum_{x' \in W_x : x \leq x'} \Pr[X = x']$$

Wobei $W_x = \{x_1, ..., x_n\}$ der Wertebereich der Zufallsvariable ist.

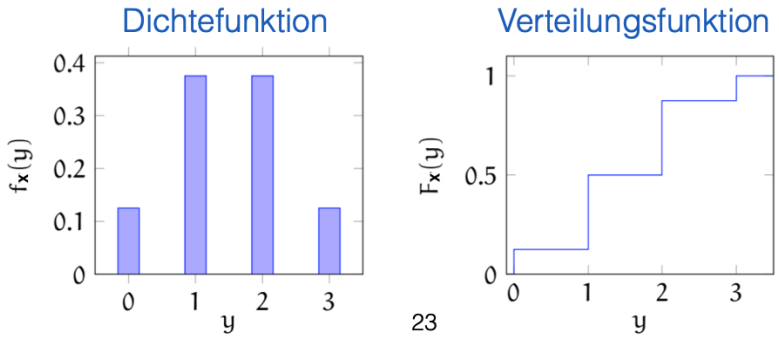

Figure 11.1: Beispiel von Dichte- und Verteilungsfunktion

11.1 Erwartungswert

Beim Betrachten von Zufallsvariablen ist es interessant zu wissen, welches Ergebnis man "im Mittel" erwarten kann. Wir definieren dazu:

Definition

Zu einer Zufallsvariablen X definieren wir den **Erwartungswert** $\mathbb{E}[X]$ durch

$$\mathbb{E}[X] := \sum_{x \in W_x} x \cdot \Pr[X = x]$$

oder als alternative Definition

$$\mathbb{E}[X] := \sum_{\omega \in \Omega} X(\omega) \cdot \Pr[\omega]$$

Wobei zu erwähnen ist, dass wir in der Vorlesung nur Zufallsvariablen betrachten, wo auch wirklich ein Erwartungswert existiert.

Für ganzzahlige, positive Wertebereiche, können wir den Erwartungswert wie folgt berechen:

Satz

Sei X eine Zufallsvariable mit $W_X \subseteq \mathbb{N}_0$, dann gilt

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i]$$

Eine wichtige Eigenschafft des Erwartungswertes ist seine Linearität.

Linearität des Erwartungswertes

Für Zufallsvariablen $X_1, ..., X_n$ und $X = \alpha_1 \cdot X_1 + ... + \alpha_n \cdot X_n + \beta$ mit $\alpha_1, ..., \alpha_n, \beta \in \mathbb{R}$ gilt

$$\mathbb{E}[X] = \alpha_1 \cdot \mathbb{E}[X_1] + ... + \alpha_n \cdot \mathbb{E}[X_n] + \beta$$

An dieser Stelle lohnt es sich auch den Begriff der Indikatorvariable einzuführen.

Definition

Für ein Ereignis $A \subseteq \Omega$ ist die zugehörige **Indikatorvariable** X_A definiert durch:

$$X_A(\omega) := \begin{cases} 1, & \text{falls } (\omega) \in A \\ 0, & \text{sonst} \end{cases}$$

Für den Erwartungswert von X_A gilt: $\mathbb{E}[X_A] = \Pr[A]$.

Chapter 12

Diskrete Verteilungen

Oft können wir eine Zufallsvariable einer gewissen Klasse zuordnen, welche eine ganze Familie von Zufallsvariablen parametrisiert beschreibt.

12.1 Bernoulli-Verteilung

Eine Zufallsvariable X mit $W_X = \{0, 1\}$ und der Dichte

$$f_X = \begin{cases} p, \text{ für } x = 1 \\ 1 - p, \text{ für } x = 0 \\ 0, \text{ sonst} \end{cases}$$

heisst Bernoulli-verteilt. p wird Erfolgswahrscheinlichkeit genannt. Oft sind diese Variablen Indikatorvariablen, da diese den entsprechenden Wertebereich und Dichtefunktion aufweisen. Oft schreibt man für eine Zufallsvariable, die Bernoulli-verteilt ist

$$X \sim \text{Bernoulli}(p)$$

Für solche eine Zufallsvariable X gilt

$$\mathbb{E}[X] = p \text{ und } \text{Var}[X] = p(1 - p)$$

12.2 Binomialverteilung

Als klassisches Beispiel für eine Bernoulli-verteilte Zufallsvariable gilt die Indikatorvariable für "Kopf" beim (unfairen) Münzwurf. Wenn wir dieses Experiment n-mal wiederholen und uns fragen, wie oft wir Kopf erhalten, so ist diese neue Variable binomialverteilt.

Eine Zufallsvariable X mit $W_X = \{0, 1, ..., n\}$ und der Dichte

$$f_X = \begin{cases} \binom{n}{x} p^x (1 - p)^{n-x}, \text{ für } x \in \{0, 1, ..., n\} \\ 0, \text{ sonst} \end{cases}$$

Kurz schreibt man auch

$$X \sim \text{Bin}(n, p)$$

Für solche eine Zufallsvariable X gilt

$$\mathbb{E}[X] = np \text{ und } \mathrm{Var}[X] = np(1-p)$$

Die Verteilung drückt folgendes aus: Wir wählen aus allen Versuche $\binom{n}{x}$, so dass x Versuche Erfolg haben p^x, und haben in den restlichen Versuche Misserfolg $(1-p)^{n-x}$.

12.3 Poisson-Verteilung

Die Poisson-Verteilung ist motiviert durch die Betrachtung von Ereignissen, die mit sehr geringer Wahrscheinlichkeit auftreten, zum Beispiel $X :=$ Anzahl Herzinfarkte in der Schweiz in der nächsten Stunde. Fuür einen einzelnen Bürger ist diese Wahrscheinlichkeit schwindend gering, aber schweizweit gibt es ca. drei bis vier pro Stunde.

Die Poisson-Verteilung ist als Grenzwert der Binomialverteilung zu verstehen: Lässt man die Verteilung $X \sim \mathrm{Bin}(n, \frac{\lambda}{n})$ gegen unednlich laufen ($n \to \infty$), so erhält man die Poisson-Verteilung, welche wie folgt aussieht:

$$X \sim \mathrm{Po}(\lambda)$$

Für die Dichte gilt:

$$f_X = \begin{cases} \frac{e^{-\lambda}\lambda^i}{i!}, \text{ für } i \in \mathbb{N}_0 \\ 0, \text{ sonst} \end{cases}$$

Erwartungswert und Varianz sehen wie folgt aus:

$$\mathbb{E}[X] = \mathrm{Var}[X] = \lambda$$

Gilt für eine Binomialverteilung, dass der erste Parameter n (Anzahl Versuche) sehr gross ist und dass der Erwartungswert $\mathbb{E}[X] = np$ eine (kleine) Konstante ist, so kann die Binomialverteilung durch die Poisson-Verteilung approximiert werden, wobei der Parameter λ der Poisson-Verteilung durch den Erwartungswert der Binomialverteilung gegeben ist.

12.4 Geometrische-Verteilung

Wir haben bereits mehrere Experimente kennengelernt, die solange ausgeführt wurden bis ein Erfolg eintritt. Wenn ein solches Experiment mit Erfolgswahrscheinlichkeit p ausgeführt wird, so ist die Zufallsvariable für die Anzahl der Versuche, die ausgeführt wird geometrisch verteilt, anders geschrieben:

$$X \sim \mathrm{Geo}(p)$$

Die Dichte sieht wiefolgt aus:

$$f_X = \begin{cases} p(1-p)^{i-1}, \text{ für } i \in \mathbb{N} \\ 0, \text{ sonst} \end{cases}$$

Erwartungswert und Varianz sind:

$$\mathbb{E}[X] = \frac{1}{p} \text{ und } \mathrm{Var}[X] = \frac{1-p}{p^s}$$

Eine wichtige Eigenschaft der geometrischen Verteilung ist die **Gedächtnislosigkeit**:
Die Wahrscheinlichkeit, "Kopf" im ersten Versuch zu werfen ist identisch mit der Wahrscheinlichkeit, "Kopf" nach 1000 Fehlversuchen beim 1001. Versuch zu werfen.

Gedächtnislosigkeit

Ist $X \sim \text{Geo}(p)$, so gilt für alle $s, t \in \mathbb{N}$:

$$\Pr[X \geq s + t | X > s] = \Pr[X \geq t]$$

12.5 Negative Binomialverteilung

Statt wie bei der geometrischen Verteilung auf den ersten Erfolg zu warten, könnten wir ein Experiment auch bis zum Auftreten von n Erfolgen wiederholen. Für $n = 1$ ist also unsere Zufallsvariable "Anzahl Versuche" geometrisch verteilt, für $n \geq 2$ ist sie negativ binomialverteilt oder kurz:

$$X \sim \text{NegativeBinomial}(n, p)$$

Für die Dichte der Zufallsvariable gilt:

$$f_X(k) = \begin{cases} \binom{k-1}{n-1}(1-p)^{k-n}p^n, & \text{für } k = 1, 2, ... \\ 0, & \text{sonst} \end{cases}$$

Erwartungswert und Varianz sind:

$$\mathbb{E}[X] = \frac{n}{p} \text{ und } \text{Var}[X] = \frac{n(1-p)}{p^s}$$

12.6 Coupon Collector

Betrachten wir zum Abschluss dieses Abschnitts ein berühmtes Problem mit folgendem Szenario: Es gibt n verschiedene Bilder zu sammeln. In jeder Runde erhalten wir (mit gleicher Wahrscheinlichkeit) eines der Bilder. Sei X die Zufallsvariable die besagt, nach wie vielen Runden wir alle n Bilder besitzen. Was ist der Erwartungswert $\mathbb{E}[X]$?

Zur Lösung betrachten wir n verschiedene Phasen. In Phase i besitzen wir $i - 1$ verschiedene Bilder. Nun gilt, dass $X_i :=$ Anzahl Runden in Phase i geometrisch verteilt ist:

$$X_i \sim \text{Geo}(\frac{n - (i-1)}{n})$$

Jetzt nutzen wir die Linearität des Erwartungswertes und erhalten folgenden Term:

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \cdot \sum_{i=1}^{n} \frac{1}{i} = n \cdot H_n$$

Wobei H_n die harmonische Reihe bis zum Glied n ist. Diesen Term können wir asymptotisch angeben als $H_n = \ln(n) + \mathcal{O}(1)$

Chapter 13

Mehrere Zufallsvariablen

Bevor wir uns anschauen was passiert, wenn wir mehrere Zufallsvariablen haben, schauen wir uns zuerst bedingte Zufallsvariablen an.

Definition

Sei X eine Zufallsvariable auf dem Wahrscheinlichkeitsraum Ω mit $A \subseteq \Omega$ und $\Pr[A] > 0$. Die **bedingte Zufallsvariable** $X|A$ ist dieselbe Funktion wie X, aber der Definitionsbereich ist auf die Teilmenge A eingeschränkt.

Dichtefunktion

$$f_{X|A} : \mathbb{R} \to [0,1], x \mapsto \Pr[X = x|A]$$

Verteilungsfunktion

$$F_{X|A} : \mathbb{R} \to [0,1], x \mapsto \Pr[X \leq x|A]$$

Erwartungswert

$$\mathrm{E}[X|A] := \sum_{x \in W_X} x \cdot \Pr[X = x|A] = \frac{1}{\Pr[A]} \cdot \sum_{\omega \in A} X(\omega) \cdot \Pr[\omega]$$

Oftmals interessieren wir uns nun für mehrere Zufallsvariablen zugleich. In diesem Kapitel beschäftigen wir uns mit der Frage, wie wir mit mehreren Zufallsvariablen über demselbem Wahrscheinlichkeitsraum rechnen können - selbst wenn diese sich gegenseitig beeinflussen. Dazu definieren wir:

Definition

Für zwei Zufallsvariablen X, Y heisst die Funktion

$$f_{X,Y}(x,y) := \Pr[X = x, Y = y]$$

die **gemeinsame Dichte** der Zufallsvariable X und Y. Von der gemeinsamen Dichte kann man auch wieder zu der Dichte der jeweiligen Zufallsvariablen übergehen - den sogenannten Randdichten:

$$f_X(x) = \sum_{y \in W_y} f_{X,Y}(x,y) \text{ und } f_Y(y) = \sum_{x \in W_X} f_{X,Y}(x,y)$$

Für ein Beispiel siehe Skript. Analog dazu kann man auch gemeinsame Verteilung und Randverteilung definieren, was in der Vorlesung jedoch nicht verwendet wurde.

13.1 Unabhängigkeit von Zufallsvariablen

Analgo zu Ereignissen können wir auch für Zufallsvariablen Unabhängigkeit definieren. Dies ist offensichtlich der Fall, wenn Zufallsvariablen physikalisch voneinander unabhängig sind. Es kann aber auch nur implizit durch nachrechnen bestätigt werden.

Definition

Zufallsvariablen $X_1, ..., X_n$ heissen unabhängig genau dann, wenn für alle $(x_1, ..., x_n) \in W_{X_1} \times ... \times W_{X_n}$ gilt

$$\Pr[X_1 = x_1, ..., X_n = x_n] = \Pr[X_1 = x_1] \cdot ... \cdot \Pr[X_n = x_n]$$

Oder alternativ

$$f_{x_1,...,x_n}(x_1, ..., x_n) = f_{X_1}(x_1) \cdot ... \cdot f_{X_n}(x_n)$$

Aus dieser Definition können wir direkt folgern, dass zwei Indikatorvariablen unabhängig sind, wenn ihre Ereignissen unabhängig sind. Allgemein gilt für zwei Indikatorvariablen:

$$X \text{ und } Y \text{ sind unabhängig } \Leftrightarrow f_{X,Y}(1,1) = f_X(1) \cdot f_Y(1)$$

Aus der Definition folgen einige Lemmas: