ETH Zürich
Algorithmen, Wahrscheinlichkeit und Statistik

Lukas Wolf

Sommer 2019

Contents

1 Graphentheorie

1.1 Graphentheorie - Wiederholung

Satz 1.2
Für jeden Graphen G=(V,E) gilt: $\sum_{v \in V} deg(v) = 2|E|$.

Korollar 1.3
Für jeden Graphen G=(V,E) gilt: Die Anzahl der Knoten mit ungeradem Grad ist gerade.

Korollar 1.4
In jedem Graph G=(V,E) ist der durchschnittliche Knotengrad gleich $2|E|/|V|$. Insbesondere gibt es daher Knoten $x, y \in V$ mit $deg(x) \leq 2|E|/|V|$ und $deg(x) \geq 2|E|/|V|$.

Lemma 1.5
Ist T=(V,E) ein Baum mit $|V| \geq 2$ Knoten, so gilt:
a) T enthält mindestens zwei Blätter,
b) ist $v \in V$ ein Blatt, so ist der Graph T-v ebenfalls ein Baum.

Satz 1.6
Ist G=(V,E) ein Graph auf $|V| \geq 1$ Knoten, so sind die folgenden Aussagen äquivalent:
a) G ist ein Baum,
b) G ist zusammenhängend und kreisfrei,
c) G ist zusammenhängend und $|E| = |V| - 1$,
d) G ist kreisfrei und $|E| = |V| - 1$,
e) für alle $x, y \in V$ gilt: G enthält genau einen x-y-Pfad.

Lemma 1.7
Ein Wald G=(V,E) enthält genau $|V| - |E|$ viele Zusammenhangskomponenten.

Prim's Algorithm
Erweitert von einem Startknoten aus (n-1)-mal mit an die Zusammenhangskomponente des Startknoten anschliessenden Kanten mit dem jeweils kleinsten Gewicht.

Kruskal's Algorithm
Wählt (n-1)-mal die Kante mit kleinstem Gewicht, so dass die beiden Endpunkte der Kanten nicht in der gleichen Zusammenhangskomponente sind.

Kürzeste Pfade
Dijkstra(one-to-all, pos. weights), Floyd-Warshall(all-to-all), Bellman-Ford(one-to-all, can detect neg. cycles), Johnson (all to all, neg. weights)

1.2 Zusammenhang

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

Definition: Ein Graph G = (V,E) heisst *k-kanten-zusammenhängend*, falls $|E| \geq k+1$ und für alle Teilmengen $X \subseteq E$ mit $|X| < k$ gilt: Der Graph $(V, E \backslash X)$ ist zusammenhängend.

Satz (Menger):
Sei G = (V,E) ein Graph. Dann gilt:
a) G ist genau dann k-zusammenhängend, wenn es für alle Paare von Knoten $u, v \in V, u \neq v$, mindestens k intern-kanten-disjunkte u-v-Pfade gibt.
b) G ist genau dann k-kanten-zusammenhängend, wenn es für alle Paare von Knoten $u, v \in V, u \neq v$, mindestens k intern-kantendisjunkte u-v-Pfade gibt.
Es gilt immer:
Knotenzusammenhang $\leq$ Kantenzusammenhand $\leq$ minimaler Knotengrad

Artikulationsknoten
Spezialfall k = 1:
Sei G = (V,E) ein zusammenhängender Graph.
Definition: Ein Knoten $v \in V$ heißt Artikulationsknoten (engl. cut vertex) g.d.w. $G[V \backslash \{v\}]$ nicht zusammenhängend ist.

Brücke
Definition: Eine Kante $e \in E$ heißt Brücke (engl. cut edge) g.d.w. $G \backslash e$ nicht zusammenhängend ist.
Lemma: Sei G = (V,E) ein zusammenhängender Graph. Ist $\{x, y\} \in E$ eine Brücke, so gilt: $deg(x) = 1$ oder x ist ein Artikulationsknoten.
Achtung: Die Umkehrung gilt im Allgemeinen nicht!

Effiziente Bestimmung von Artikulationsknoten / Brücken:
1. Führe DFS auf Graph aus. Speichere die dfs-Zahl (Reihenfolge, mit welcher die Knoten besucht werden).
2. Berechne "on the way" die low-Werte. Ein low-Wert low[v] wird mit dem Wert dfs[v] initialisiert. Dann wird er ggf. geupdated und der kleinste Wert retourniert. Update, wenn Rückwärtskanten gefunden werden oder wenn zu sich selbst zurückgekehrt.
low-Wert: kleinste dfs-Nr., die man von v aus durch einen Pfad aus (bel. vielen) Vorwärtskanten und max. einer Rückwärtskanten erreichen kann.

Ein Knoten v ist ein Artikulationsknoten, genau dann wenn v = s (Startknoten) und s hat in T (DFS tree) Grad mindestens zwei, oder $v \neq s$ und es gibt $w \in V$ mit $\{v, w\} \in E(T)$ und $low[w] \leq dfs[v]$.

Satz: Die um die von low[] ergänzte DFS berechnet in einem zusammenhängenden Graphen alle Artikulationsknoten und Brücken in Zeit O(m). Dies gilt auch für zusammenhängende Graphen, die als Adjazenzlisten gespeichert sind.

1.3 Kreise

Eulertour
Definition: Eine Eulertour in einem Graphen G = (V,E) ist ein geschlossener Weg (Zyklus), der jede Kante des Graphen genau einmal enthält. Enthält ein Graph eine Eulertour, so nennt man ihn eulersch.
Satz:
a) Ein zusammenhängender Graph G= (V,E) ist genau dann eulersch, wenn der Grad aller Knoten gerade ist.
b) In einem zusammenhängenden, eulerschen Graphen kann man eine Eulertour in Zeit $O(\mid E \mid)$ finden
Bemerkung: Sind in einem zusammenhängenden Graphen G = (V,E) alle bis auf zwei Knotengrade gerade, so enthält der Graph einen Eulerweg.

Algorithmus zur Bestimmung Eulertour: langsamer / schneller Läufer:
Die Idee ist basiert auf zwei Annahmen: Wir wissen der Graph ist zusammenhängend und alle Knoten haben einen geraden Grad. Deshalb wählen wir einen beliebigen Startknoten v und laufen einen belibigen Weg W und löschen dabei jede benutzte Kante. Der Weg W ist ein Zyklus und endet wieder bei v (Wiederspruchsbeweis S.41). Daher haben wir nun entweder eine Eulertour gefunden, oder es existiert ein Knoten v' (mit unbenutzten Kanten), von dem eine weitere Eulertour W' gelaufen werden kann, welche dann mit der ersten Tour W verschmolzen werden kann, indem man von v' aus zuerst W' läuft und dann W zurück zu v.
Um jeweils einen Knoten v' zu finden, verwenden wir einen "langsamen" Läufer, der sich nur bewegt, wenn der "schnelle" Läufer feststeckt. Er startet genauso im Knoten v. Steckt der schnelle Läufer fest, bewegt sich der langsame Läufer so lange auf W und prüft bei jedem Knoten ob es noch ausgehende Kanten gibt. Somit findet er v' und der schnelle Läufer startet hier erneut.
Dies setzen wir fort, bis der langsame Läufer den Weg W einmal komplett abgelaufen hat. Dann gibt es keine Kanten mehr und W ist die finale Eulertour.

Hamiltonkreis
Definition: Ein Hamiltonkreis in einem Graphen G = (V,E) ist ein Kreis, der alle Knoten von V genau einmal durchläuft. Enthält ein Graph einen Hamiltonkreis, so nennt man ihn hamiltonsch.

Satz:
Ein n * m Gitter enthält einen Hamiltonkreis, genau dann wenn n * m gerade ist.

Algorithm 1: Eulertour(G,v_{start})

```
1  // schneller Läufer
2  W ← v_start
3  // langsamer Läufer
4  v^langsam ← StartknotenvonW
5  while v^langsam ist nicht letzter Knoten in W do
6    v ← Nachfolger von v^langsam in W
7    if(N_G(v) ≠ ∅ do
8      W' ← Randomtour(v)
9      // ergänze W = W_1+ < v > +W_2 um die Tour W'
10     W ← W_1 + W' + W_2
11   v^langsam ← Nachfolger von v^langsam in W
12 return W
```

Algorithm 2: Randomtour(v_{start})

```
1  v ← v_start
2  W ←< v >
3  while N_G(v) ≠ ∅ do
4    wähle v_next aus N_G(v) beliebig
5    hänge v_next an die Tour W an
6    e ← {v, v_next}
7    lösche e aus G
8    v ← v_next
9  return W
```

Hamiltonkreis - Beispiele:
d-dimensionaler Hyperwürfel H_d:
Knotenmenge: $\{0,1\}^d$, Kantenmenge: alle Knotenpaare, die sich in genau einer Koordinate unterscheiden.
Klassisches Anwendungsbeispiel: Travelling Salesman Problem

Wie entscheidet man (effizient), ob ein Graph einen Hamiltonkreis enthält? Dieses Problem ist NP-vollständig (Karp 1972).
→ dynamic programming Ansatz

Hamiltonkreis - Dynamic Programming
Zunächst überlegen wir uns, dass es genügt zu testen, ob es ein $x \in N(1)$ (Nachfolger von Knoten 1) gibt, für dass es einen 1-x-Pfad gibt, der alle Knoten aus $V = [n]$ enthält.
Wenn es einen Hamiltonkreis in G gibt, so enthält dieser Kreis den Knoten 1 und eine Kante von 1 zu einem Nachbar $x \in N(1)$. Der Hamiltonkreis ohne diese Kante ist dann der gesuchte 1-x-Pfad.
Um zu testen ob es solch ein $x \in N(1)$ gibt, definieren wir für alle Teilmengen $S \subseteq [n]$ mit $1 \in S$ und für alle $x \in S$ mit $x \neq 1$:
$P_{S,x} = 1$, falls es in G einen 1-x-Pfad gibt, der genau die Knoten aus S enthält
$P_{S,x} = 0$, sonst.
Dann gilt:
G enthält einen Hamiltonkreis $\iff \exists x \in N(1)$ mit $P_{[n],x} = 1$
und wir können die Werte $P_{S,x}$ mit Hilfe der dynamischen Programmierung berechnen.
Dazu überlegen wir uns zunächst, dass für die Mengen $S = \{1, x\}$ offenbar gilt: $P_{S,x} = 1$, genau dann wenn $\{1, x\} \in E$. Damit haben wir die initialisierung geschafft.
Für die Rekursion von Mengen der Grösse s auf Mengen der Grösse s+1, für $s \geq 2$, überlegen wir uns zunächst folgendes:
$P_{S,x} = 1$, genau dann wenn es ein $x' \in N(x)$ gibt mit $x' \neq 1$ und $P_{S\setminus[x],x'} = 1$, denn ein 1-x Pfad mit Knoten aus S muss ja einen Nachbarn x' von x als vorletzten Knoten enthalten. Und der Pfad von 1 bis zu diesem Knoten x' ist dann ein 1-x'-Pfad, der genau die Knoten aus $S \setminus [x]$ enhält.
Es gilt also:
$P_{S,x} = max\{P_{S\setminus[x],x'} | x' \in S \cap N(x), x' \neq 1\}$.

Laufzeit: $O(n^2 \cdot 2^n)$
Speicherplatz: $O(n \cdot 2^n)$

Lemma 1.35
Ist $G = (A \cup B, E)$ ein bipartiger Graph mit $|A| \neq |B|$, so kann G keinen Hamiltonkreis enthalten. (Paritätsargument)

Gray-Code:
Der Gray-Code ist ein stetiger Code, bei dem sich benachbarte Codewörter nur in einer einzigen binären Ziffer unterscheiden, die Hamming-Distanz benachbarter Codewörter ist 1.

Satz 1.37 (Dirac):
Jeder Graph G = (V,E) mit $|V| \geq 3$ und Minimalgrad $\delta(G) \geq \frac{|V|}{2}$ enthält einen Hamiltonkreis.

Traveling Salesman Problem (TSP):
Gegeben: vollständiger Graph und Funktion l(e) die jeder Kante des Graphen eine Länge zuordnet
Gesucht: Hamiltonkreis mit minimaler Länge
Die Funktion l(x) erfüllt die Dreiecksungleichung. Aus dem Minimum Spanning Tree (MST) kann man eine 2-Approximation ableiten.

Das TSP ist allgemeiner als das Hamiltonkreisproblem und damit sicherlich nicht einfacher zu lösen. Allerdings erlaubt uns die Formulierung als Optimierungsproblem eine differenziertere Antwort: wir können jetzt auch die Güte einer nicht optimalen Lösung bewerten. Entsprechend spricht man von einem α-Approximationsalgorithmus, wenn der Algorithmus immer einen Hamiltonkreis C findet, dessen Lösung um den Faktor α von der optimalen abweicht.

Satz 1.39
Gibt es für ein $\alpha > 1$ einen α -Approximationsalgorithmus für das TSP mit Laufzeit O(f(n)), so gibt es auch einen Algorithmus, der für alle Graphen auf n Knoten in Laufzeit O(f(n)) entscheidet, ob sie hamiltonsch sind.

Metrisches TSP
Annahme: Vollständiger Graph K_n und Gewichtsfunktion l erfüllt die Dreiecksungleichung.
Wir starten mit einem MST des Graphen. Dabei wird jede Kante zwei Mal durchlaufen. Die Gesamtlänge des entsprechenden (geschlossenen) Weges ist daher $2 \cdot mst(K_n, l)$. Anschliessend laufen wir den weg nochmals ab und lassen Knoten aus, die wir bereits durchlaufen haben. Da die Dreiecksungleichung gilt, wird damit das Gesamtgewicht vielleicht kleiner, aber sicher nicht grösser.

Satz 1.40
Für das metrische TSP gibt es einen 2-Approximationsalgorithmus mit Laufzeit $O(n^2)$.

1.4 Matchings

Eine Kantenmenge $M \in E$ in einem Graphen G = (V,E) heisst Matching, falls kein Knoten des Graphen zu mehr als einer Kante aus M inzident ist.
Formal : $e \cap f = \emptyset$ für alle $e, f \in M$ mit $e \neq f$.
Ein Knoten v wird von M überdeckt, falls es 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 $|M| = \frac{|V|}{2}$.

Ein Matching $M \subseteq E$ ist ein inklusionsmaximales Matching, wenn es kein Matching M' gibt mit $M \subseteq M'$ und $|M'| > |M|$. Ein inklusionsmaximales Matching hat also die Eigenschaft, dass man keine Kante mehr hinzufügen kann, ohne die Matching-Eigenschaft zu zerstören. Ein solches Matching muss nicht unbedingt kardinalitätsmaximal sein.
Ein Matching $M \subseteq E$ ist ein (kardinalitäts-) maximales Matching, wenn es kein Matching M' gibt mit $|M'| > |M|$. Ein kardinalitätsmaximales Matching ist immer auch inklusionsmaximal.

Satz 1.44:
Mit dem Greedy-Algorithmus kann man in Zeit $O(|E|)$ ein inklusionsmaximales Matching M_{Greedy} bestimmen mit $|M_{Greedy}| \geq \frac{|M_{max}|}{2}$, wobei M_{max} ein kardinalitätsmaximales Matching ist.

Algorithm 3: Greedy-Matching(G)

```
1 M = ∅
2 while (E ≠ ∅) do
3         wähle eine beliebige Kante e ∈ E
4         M ⟵ M ∪ {e}
5         lösche e und alle inzidenten Kanten in G
```

M-augmentierender Pfad P :
Die beiden Endknoten des Pfades P werden nicht durch das Matching M überdeckt (haben also in M Grad 0) und P besteht abwechselnd aus Kanten die nicht zu M gehören und Kanten aus M.

Damit erhalten wir einen Algorithmus zur Bestimmung eines kardinalitätsmaximalen Matchings:
Wir beginnen mir einem Matching, das auf einer einziges (beliebigen Kante besteht). Solange das Matching noch nicht kardinalitätsmaximal ist, gibt es einen augmentierenden Pfad, der es uns erlaubt, das Matching zu vergrössern. Spätestens nach —V— / 2 - 1 solchen Schritten ist das Matching dann maximal. Augmentierende Pfade kann man für bipartite Graphen einfach mit einer modifizierten Breitensuche bestimmen.

Die Vereinigung zweier Matchings besteht aus Kreisen gerader Länge und alternierenden Pfaden (gerader und ungerader Länge).
Beobachtung: Vertauschen wir innerhalb eines Kreises oder Pfades die Farben, erhalten wir wiederum Matchings.
Konzept der alternierenden Pfade:
$O(\mid V \mid \cdot \mid E \mid)$ für bipartite Graphen
$O(\mid V \mid^{\frac{1}{2}} \cdot \mid E \mid)$ für ungewichtete Graphen
$O(\mid V \mid^3)$ für gewichtete Graphen

Satz 1.45:
Ist n gerade und l(e) eine Gewichtsfunktion des vollständigen Graphen K_n, so kann man in Zeit $O(n^3)$ ein minimales Matching finden, also ein perfektes Matching M mit:
$\sum_{e \in M} l(e) = min\{\sum_{e \in M'} l(e) | \text{M' perfektes Matching in } K_n\}$.

Satz 1.46:
Für das metrische TSP-Problem gibt es einen 3/2-Approximationsalgorithmus mit Laufzeit $O(n^3)$.

Satz 1.47: Hall (Heiratssatz):
Für einen bipartiten Graphen $G = (A \cup B, E)$ gibt es genau dann ein Matching M der Kardinalität $|M| = |A|$, wenn gilt:
$|N(X)| \geq |X|$, für alle $X \subseteq A$.
$N(X)$: Nachbarschat der Knotenmenge $X \subseteq V$.

Korollar (Frobenius):
Für alle k gilt: jeder k-reguläre bipartite Graph enthält ein perfektes Matching.

Satz 1.48
Sei $G = (A \cup B, E)$ ein k-regulärer bipartiter Graph. Dann gibt es $M_1, ..., M_k$ so dass $E = M_1 \cup ... \cup M_k$ und alle $M_i, 1 \leq i \leq k$, perfekte Matchings in G sind.

Satz 1.49:
Ist G = (V,E) ein 2^k-regulärer bipartiter Graph, so kann man in Zeit $O(|E|)$ ein pefektes Matching bestimmen.
2^k-regulär: alle Knoten haben den Grad 2^k. Tatsächlich ist 2^k nicht notwendig. Auch k ist ausreichend, aber der Algorithmus ist dann komplizierter.

1.5 Färbungen

Anwendung: Im Compilerbau, um eine Zuordnung von Variablen auf die Register des Prozessors zu finden, so dass gleichzeitig verwendete Variablen in verschiedenen Registern gespeichert werden.

Eine (Knoten-)Färbung eines Graphen G = (V,E) mit k Farben ist eine Abbildung $c : V \to [k]$, so dass gilt:
$c(u) \neq c(v)$ für alle Kanten u,v ∈ E.
Die chromatische Zahl χ_G ist die minimale Anzahl Farben, die für eine Knotenfärbung von G benötigt wird.

Für jeden Graphen G=(V,E) gilt: $\chi_G \leq k$ gdw. G k-partit
Inbesondere gilt für $\chi_G = 2$ gdw. G ist bipartit (und G nicht der leere Graph)
Ob G bipartit ist, lässt sich in $O(|E|)$ Zeit mit einer Breiten- oder Tiefensuche feststellen.

Satz 1.53:
Ein Graph G = (V,E) ist genau dann bipartit, wenn er keinen Kreis ungerader Länge als Teilgraphen enthält.

Satz:
Das Problem "Gegeben ein Graph G = (V,E), gilt $\chi(G) \leq 3$?" ist NP-vollständig.

Satz:
Einen 3-färbbaren Graphen kann man in Zeit $O(\mid E \mid)$ mit $O(n^{\frac{1}{2}})$ Farben färben.

Satz:
$\forall k \in N, \forall r \in N$: Es gibt Graphen ohne einen Kreis mit Länge ≤ k, aber mit chromatischer Zahl ≥ r. (Beweis im Skript für k=3)

Algorithm 4: Greedy-Färbung(G)

```
1 wähle eine beliebige Reihenfolge der Knoten: V = {v_1, ..., v_n}
2 c[v_i] = 1
3 for i = 2 to i = n do
4         c[v_i] = min{k ∈ ℕ | k ≠ c(u) für alle u ∈ N(v_i) ⋂ {v_1, ...v_{i-1}}}
```

Beobachtung:
Für jede Reihenfolge $V = \{v_1, ..., v_k\}$ der Knoten benötigt der Greedy-Algorithmus höchstens $\Delta(G) + 1$ viele Farben. Notation: $\Delta(G)$:= maximaler Grad in G.

Beobachtung:
Es gibt eine Reihenfolge der Knoten für die der Greedy-Algorithmus nur $\chi(G)$ viele Farben benötigt.

Beobachtung:
Es gibt bipartite Graphen und eine Reihenfolge der Knoten für die der Greedy-Algorithmus $\frac{|V|}{2}$ viele Faben benötigt.
Jeder Graph kann in Zeit $O(|E|)$ mit $\Delta(G)+1$ vielen Farben gefärbt werden (Greedy-Färbung).

Satz 1.59 (Brooks):
$G \neq K_n, G \neq C_{2n+1}$, d.h. nicht vollständig, kein ungerader Kreis, G zusammenhängend:
$\rightarrow$ G kann in Zeit $O(|E|)$ mit $\Delta(G)$ Farben gefärbt werden.

Satz 1.60:
Wenn jeder Subgraph einen Knoten vom Grad $\leq k$ enthlt, dann kann der Graph in Zeit $O(|E|)$ mit (k+1) Farben gefärbt werden und es gilt $\chi \leq k+1$.

2 Wahrscheinlichkeitstheorie (diskret)

2.1 Grundbegriffe und Notationen

Anwendungsgebiete von Wahrscheinlichkeitstheorie in der Informatik:
Erzeugen von elektronischem Geld:
Bank bekommt Schlüssel für 20 Kisten vom Kunden mit je zwei Schlössern. Sie bittet den Kunden die Kisten aufzusperren und prüft ob in jeder Kiste Name und Kontonummer verhanden bzw. korrekt sind und gibt ihm dafür einen "elektronischen Geldschein". Zahlt der Kunde damit in einem Geschäft, so verlangt das Geschäft zufällig einen der beiden Schlüssel (links / rechts) vom Kunden. Das Geschäft geht damit zur Bank. Zahlt der Kunde mit dem gleichen Geldschein, so erhält die Bank im Mittel für die Hälfte der Schlösser beide Schlüssel und detektiert den Betrug.
Verteiltes Rechnen:
Das Ziel ist es Knoten (Rechner) zu verbinden um eine gemeinsame Aufgabe zu lösen. Dafür gesucht ist eine *stabile Menge*. Eine stabile Menge ist eine Menge von Knoten, welche nicht durch Kanten verbunden sind. Es gibt keinen schnellen Algorithmus der eine "große" stabile Menge findet. Deshalb wird folgendes gemacht: in Runde 1 löschen sich alle Knoten mit Wahrscheinlichkeit p. In der zweiten Runde löschen alle noch vorhandenen Kanten einen Knoten zufällig. Dies führt zu einer stabilen Menge.
Bestimmung einer stabilen Menge:
$\mathbb{E}[S] \geq np - mp^2$ mit $p = \frac{1}{d}$ wobei $d := 2\frac{m}{n}$ der Durchschnittsgrad erhalten wir:
$\mathbb{E}[S] \geq \frac{n}{2d}$.

Wahrscheinlichkeitsraum:
Ein diskreter Wahrscheinlichkeitsraum ist bestimmt durch eine Ergebnismenge/Ereignisraum $\Omega = \{\omega_1, \omega_2, ...\}$ von Elementarereignissen.
Jedem Elementarereignis ω_i ist eine (Elementar-) Wahrscheinlichkeit $Pr[\omega_i]$ zu-

geordnet, wobei wir fordern, dass $0 \leq Pr[\omega_i] \leq 1$ und $\sum_{\omega \in \Omega} Pr[\omega] = 1$.
Eine Menge $E \subseteq \Omega$ heisst Ereignis. Die Wahrscheinlichkeit $Pr[E]$ eines Ereignisses ist definiert durch $Pr[E] := \sum_{\omega \in E} Pr[\omega]$.
Ist E ein Ereignis, so bezeichnen wir mit $\overline{E} := \Omega \backslash E$ das Komplementärereignis zu E.
Die Klasse aller beobachtbaren Ereignisse ist F, wobei F eine Teilmenge von Ω ist. Ist Ω endlich, so gilt $F = 2^{\Omega}$, das heisst das jede Teilmenge von Ω beobachtbar ist.

Elementare und nützliche Konsequenzen:
Für Ereignisse A, B gilt:

1. $Pr[\emptyset] = 0, Pr[\Omega] = 1$
2. $0 \leq Pr[A] \leq 1$
3. $Pr[\overline{A}] = 1 - Pr[A]$
4. Wenn $A \subseteq B$, so folgt $Pr[A] \leq Pr[B]$

Additionssatz:
Wenn die Ereignisse $A_1, ..., A_n$ paarweise disjunkt sind (also wenn für alle Paare $i \neq j$ gilt, dass $A_i \cap A_j = \emptyset$), so gilt:
$Pr[\bigcup_{i=1}^{n} A_i] = \sum_{i=1}^{n} Pr[A_i]$.
Für eine unendliche Menge von disjunkten Ereignissen gilt analog:
$Pr[\bigcup_{i=1}^{\infty} A_i] = \sum_{i=1}^{\infty} Pr[A_i]$.

Vereinigung von Ereignissen - Siebformel:
Regel: ungerade Anzahl Vereinigungen werden addiert, gerade Anzahl Vereinigungen subtrahiert, (Allgemeine Definition siehe Skript S. 76)
Bsp.:
$Pr[A \cup B \cup C] = Pr[A] + Pr[B] + Pr[C]$
$- Pr[A \cap B] - Pr[A \cap C] - Pr[B \cap C] + Pr[A \cap B \cap C]$

Boolesche Ungleichung / Union Bound:
Für Ereignisse $A_1, ..., A_n$ gilt:
$Pr[\bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} Pr[A_i]$.
Analog gilt für eine unendliche Folge von Ereignissen, dass
$Pr[\bigcup_{i=1}^{\infty} A_i] \leq \sum_{i=1}^{\infty} Pr[A_i]$.

Laplace-Raum:
Ein Laplace-Raum ist ein endlicher Wahrscheinlichkeitsraum, in dem alle Elementarereignisse gleich wahrscheinlich sind.
Prinzip von Laplace: Wenn nichts dagegen spricht, gehen wir davon aus, dass alle Elementarereignisse gleich wahrscheinlich, d.h. uniform verteilt / gleichverteilt sind.
In einem Laplace-Raum gilt für jedes Ereignis E: $Pr[E] = \frac{|E|}{|\Omega|}$.

2.2 Bedingte Wahrscheinlichkeiten

Bedinge Wahrscheinlichkeit:
A und B seien Ereignisse mit $Pr[B] > 0$. Die bedingte Wahrscheinlichkeit $Pr[A|B]$ von A gegeben B ist definiert durch:
$Pr[A|B] := \frac{Pr[A \cap B]}{Pr[B]}$

Die bedingten Wahrscheinlichkeiten der Form $Pr[\cdot|B]$ bilden für ein beliebiges Ereignis $B \subseteq \Omega$ mit $Pr[B] > 0$ einen neuen Wahrscheinlichkeitsraum über (Ω, F).
Damit gelten alle Rechenregeln für Wahrscheinlichkeiten auch für bedingte Wahrscheinlichkeiten.

Satz von der totalen Wahrscheinlichkeit
Die Ereignisse $A_1, ..., A_n$ seien paarweise disjunkt und es gelte $B \subseteq A_1 \subseteq ... \subseteq A_n$. Dann folgt:
$Pr[B] = \sum_{i=1}^{n} Pr[B|A_i] \cdot Pr[A_i]$
Analog gilt dies auch für unendlich viele paarweise disjunkte Ereignisse.

Multiplikationssatz
Bedingte Wahrscheinlichkeit für mehr als zwei Ereignisse.
Seien die Ereignisse $A_1, ..., A_n$ gegeben.
Falls $Pr[A_1 \cap ... \cap A_n] > 0$ ist, gilt:
$Pr[A_1 \cap ... \cap A_n] = Pr[A_1] \cdot Pr[A_2|A_1] \cdot Pr[A_3|A_1 \cap A_2] \cdot ... \cdot \cap \mathrm{Pr}[\mathrm{A}_n|A_1 \cap ... \cap A_{n-1}]$

Satz von Bayes
Die Ereignisse $A_1, ..., A_n$ seien paarweise disjunkt. Ferner sei $B \subseteq A_1 \cup ... \cup A_n$ ein Ereignis mit $Pr[B] > 0$.
Dann gilt für ein beliebiges $i = 1, ..., n$:
$Pr[A_i|B] = \frac{Pr[A_i \cap B]}{Pr[B]} = \frac{Pr[B|A_i] \cdot Pr[A_i]}{\sum_{j=1}^{n} Pr[B|A_j] \cdot Pr[A_j]}$.
Analog gilt dies auch für unendlich viele paarweise disjunkte Ereignisse.
Mit dem Satz von Bayes ann man gewissermassen die Reihenfolge der Bedingung umdrehen.

2.3 Unabhängigkeit

Intuitive Definition: Zwei Ereignisse A und B sind unabhängig, genau dann wenn der Eintritt von B nicht beeinflusst, ob A eintritt.

Unabhängigkeit
Die Ereignisse A und B heissen unabhängig, wenn gilt:
$Pr[A \cap B] = Pr[A] \cdot Pr[B]$.
Wichtig: Ereignisse können (stochastisch) unabhängig sein, ohne dass sie "physikalisch" unabhängig sind.
Wenn $Pr[B] \neq 0$ gilt, so können wir diese Definition umformen zu:
$Pr[A] = \frac{Pr[A \cap B]}{Pr[B]} = Pr[A|B]$.

Unabhängigkeit - viele Ereignisse
Die Ereignisse $A_1, ..., A_n$ heissen unabhängig, wenn für alle Teilmengen $I \subseteq \{1, ..., n\}$ mit $I = \{i_1, ..., i_k\}$ gilt, dass:
$Pr[A_{i_1} \cap ... \cap A_{i_k}] = Pr[A_{i_1}] \cdot ... \cdot Pr[A_{i_k}]$.
Eine unendliche Familie von Ereginissen A_i mit $i \in \mathbb{N}$ heisst unabhängig, wenn obige Formel für jede endliche Teilmenge $I \subseteq \mathbb{N}$ erfüllt ist.
Offensichtlich erfüllt, wenn die Ereignisse "physikalisch" unabhängig sind (z.B. wenn jedes A_i einem unabhängigem Münzwurf entspricht). Dies ist aber nicht unbeding erforderlich (z.B. Zufallszahlengenerator).

Lemma
(äquivalente Defintion, manchmal leichter zu prüfen)
Die Ereignisse $A_1, ..., A_n$ sind genau dann unabhängig, wenn für alle $(s_1, ..., s_n) \in \{0, 1\}^n$ gilt, dass:
$Pr[A_1^{s_1} \cap ... \cap A_n^{s_n}] = Pr[A_1^{s_1}] \cdot ... \cdot Pr[A_n^{s_n}]$,
wobei $A_i^0 = \overline{A_i}$ und $A_i^1 = A_i$

Die beiden Lemmas besagen anschaulich: Um zu überprüfen, ob n Ereignisse unabhängig sind, muss man entweder alle Teilmengen untersuchen oder den Schnitt aller Ereignisse betrachten, wobei an beliebiger Stelle Ereignisse komplementiert werden können.
Aus der Darstellung des zweiten Lemmas folgt die wichtige Beobachtung, dass für zwei unabhängige Ereignisse A und B auch die Ereignisse $\overline{A}$ und B, $\overline{A}, \overline{B}$ unabhängig sind.

Lemma
Seien A, B und C unabhängige Ereignisse. Dann sind auch $A \cap B$ und C bzw. $A \cup B$ und C unabhängig.

Eigenschaften einer zufälligen Folge
Bsp: Kopf-Zahl werfen:
Wir erwarten:
Kopf und Zahl treten ungefähr gleich häufig auf, also Anzahl Kopf dividiert durch Anzahl Zahl (ungefähr 1).
Wahrscheinlichkeit dass zwei aufeinanderfolgende Buchstaben gleich sind ist 50%.

2.4 Diskrete Zufallsvariablen

Definition
Eine Zufallsvariable ist eine Abbildung $X : \Omega \to \mathbb{R}$, wobei Ω die Ergebnismenge eines Wahrscheinlichkeitsraumes ist.
W_X = Wertebereich der Zufallsvariablen X (bei diskreten Wahrscheinlichkeitsräumen abzählbar (unendlich))
$W_X := X(\Omega) = \{x \in \mathbb{R} | \exists \omega \in \Omega \text{ mit } X(\omega) = x\}$

Dichte- und Verteilungsfunktion
Dichtefunktion / Gewichtsfunktion:
$f_X : \mathbb{R} \to [0,1], x \mapsto Pr[X = x]$
Die Dichte einer Zufallsvariablen genügt, um die Zufallsvariable genau zu beschreiben.
Verteilungsfunktion:
$F_X : \mathbb{R} \to [0,1], x \mapsto Pr[X \leq x] = \sum_{x' \in W_X : x' \leq x} Pr[X = x']$

Indikatorvariable / -funktion
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: $\mathbb{E}[X_A] = Pr[A]$.
Indikatorvariablen sind immer Bernoulliverteilt.

Erwartungswert
Zu einer Zufallsvariablen X definieren wir den Erwartungswert $\mathbb{E}[X]$ durch:
$\mathbb{E}[X] := \sum_{x \in W_X} x \cdot Pr[X = x]$
sofern die Summe konvergiert. Ansonsten sagen wir, dass der Erwartungswert undefiniert ist.

Lemma:
Ist X eine Zufallsvariable, so gilt mit $X(\omega) = x, Pr[\omega] = Pr[X = x]$:
$\mathbb{E}[X] = \sum_{\omega \in \Omega} X(\omega) \cdot Pr[\omega]$

Satz:
Sei X eine Zufallsvariable mit $W_X \subseteq \mathbb{N}_0$. Dann gilt:
$\mathbb{E}[X] = \sum_{i=1}^{\infty} Pr[X \geq i]$.

Satz zum Erwartungswert paarweise disjunkter Ereignisse:
Sei X eine Zufallsvariable. Für paarweise disjunkte Ereignisse $A_1, ..., A_n$ mit $A_1 \cup ... \cup A_n = \Omega$ und $Pr[A_1], ..., Pr[A_n] > 0$ gilt:
$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X|A_i] \cdot Pr[A_i]$.
Analog gilt dies für unendlich viele paarweise disjunkte Ereignisse.

Satz zur Linearität des Erwartungswertes
Für Zufallsvariablen $X_1, ..., X_n$ und $X := a_1X_1 + ... + a_nX_n + b$ mit $a_1, ..., a_n, b \in \mathbb{R}$ gilt:
$\mathbb{E}[X] = a_1\mathbb{E}[X_1] + ... + \mathbb{E}[X_n] + b$.

Monotonie des Erwartungswertes
Ist $X \leq Y$ (d.h. $X(\omega) \leq Y(\omega) \forall \omega$), so gilt auch $E[X] \leq E[Y]$.

Satz zur Abbildung im Erwartungswert
Sei X eine diskrete Zufallsvariable mit Gewichtsfunktion $p_X(x)$, und sei $Y = g(X)$ für eine Funktion $g : \mathbb{R} \to \mathbb{R}$. Dann ist
$E[Y] = E[g(X)] = \sum_{x_k \in W(X)} g(x_k) \cdot p_X(x_k)$,
sofern die Reihe absolut konvergiert.

Satz 2.39: Definition Varianz
Für eine beliebige Zufallsvariable gilt:
$Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.
Allgemeiner: Für eine Zufallsvariable X mit $\mu = \mathbb{E}[X]$ definieren wir die Varianz Var[X] durch:
$Var[X] := \mathbb{E}[(X - \mu)^2] = \sum_{x \in W_X} (x - \mu)^2 \cdot Pr[X = x]$.
Die Grösse $\sigma := \sqrt{Var[X]}$ heisst Standardabweichung von X.

Varianz und Covarianz für mehrere Zufallsvariablen
$Var[X_1 + X_2] = E[(X_1 + X_2)^2] - (E[X_1 + X_2])^2$
$= Var[X_1] + Var[X_2] + 2(E[X_1X_2] - E[X_1]E[X_2])$.
Die folgende Grösse heisst Covarianz von X_1 und X_2:
$Cov(X_1, X_2) := E[X_1X_2] - E[X_1]E[X_2]$
$= E[(X_1 - E[X_1])(X_2 - E[X_2])$.
Damit bekommen wir die Formel:
$Var[X_1 + X_2] = Var[X_1] + Var[X_2] + 2Cov[X_1, X_2]$.
Wir bemerken, dass gilt: $Cov(X, X) = Var[X]$.
Die allgemeine Summenformel für Varianzen lautet:
$Var[\sum_{i=1} .X_i] = \sum_{i=1}^{n} Var[X_i] + 2 \sum_{i<j} Cov(X_i, X_j)$.
Falls $Cov(X_1, X_2) = 0$ gilt, so heissen die ZV unkorreliert. Damit fällt der zweite Term für paarweise unkorrelierte $X_1, ..., X_n$ in der vorherigen Formel weg.

Korrelation
Die Korrelation ist definiert als:
Corr(X,Y) = $\frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}}$.

Rechenregeln für zusammengesetzte Zufallsvariablen
Satz (Multiplikativität des Erwartungswerts)
Für unabhängige Zufallsvariablen $X_1, ..., X_n$ gilt:
$\mathbb{E}[X_1 \cdot ... \cdot X_n] = \mathbb{E}[X_1]...\mathbb{E}[X_n]$.

Satz (Linearität Varianz)
Für unabhängige Zufallsvariablen $X_1, ..., X_n$ und $X := X_1 + ... + X_n$ gilt:
$Var[X] = Var[X_1] + ... + Var[X_n]$.

Satz 2.40
Für eine beliebige Zufallsvariable X und $a, b \in \mathbb{R}$ gilt:
$Var[a \cdot X + b] = a^2 \cdot Var[X]$.

Satz (Waldsche Identität)
N und X seien zwei unabhängige Zufallsvariablen, wobei für den Wertebereich von N gelte: $W_N \subseteq \mathbb{N}$. Weiter sei:
$Z := \sum_{i=1}^{N} X_i$,
wobei $X_1, X_2, ...$ unabhängige Kopien von X seien. Dann gilt:
$\mathbb{E}[Z] = \mathbb{E}[N] \cdot \mathbb{E}[X]$.

Bemerkung: Es gelten die Implikationen:
unabhängig $\rightarrow$ paarweise unabhängig $\rightarrow$ unkorreliert

2.5 Wichtige diskrete Verteilungen

Diskrete Gleichverteilung
ZV X mit Wertebereich $W_X = \{x_1, ..., x_N\}$ und Gewichtsfunktion:
$p_X(x_k) = P[X = x_k] = \frac{1}{N}$

Bernoulli-Verteilung
$X \sim Bernoulli(p)$
$W_X = \{0, 1\}$
Bsp: Werfen einer Münze, Indikator für Kopf

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

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

Binomialverteilung
$X \sim Bin(n, p)$
$W_X = \{0, 1, ..., n\}$
Bsp: Werfen einer Münze n mal, X = Anzahl Kopf

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

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

Negativbinomiale Verteilung
Betrachten wir eine unendliche Folge von unabhängigen 0-1-Experimenten mit Erfolgsparameter p, so können wir für $r \in \mathbb{N}$ auch die Wartezeit auf den r-ten Erfolg betrachten. Dann hat X eine negativbinomiale Verteilung mit Parametern r und p. Das ist eine Verallgemeinerung der geometrischen Verteilung, die man als Spezialfall für r = 1 erhält. X lässt sich schreiben als:
$X = inf\{k \in \mathbb{N} | \sum_{i=1}^{k} I_{A_i} = r\}$.
Der Wertebereich von X ist offenbar $W(X) = \{r, r + 1, r + 2, ...\}$ und die Gewichtsfunktion ist:
$p_X(k) = P[X = k] = \binom{k-1}{r-1} p^r (1 - p)^{k-r}$.

$E[X] = \frac{r}{p}$ und $Var[X] = \frac{r(1-p)}{p^2}$

Geometrische Verteilung
$X \sim Geo(p)$
Bsp: Wiederholtes Werfen einer Münze, X = Anzahl Würfe bis zum ersten Mal Kopf

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

$\mathbb{E}[X] = \frac{1}{p}$ und $Var[X] = \frac{1-p}{p^2}$

Hypergeometrische Verteilung
In einer Urne seien n Gegenstände, davon r vom Typ 1 und n-r vom Typ 2. Man zieht ohne Zurücklegen m der Gegenstände, die ZV X beschreibt die Anzahl der Gegenstände vom Typ 1. Der Wertebereich von X ist $W(X) = \{0, 1, ..., min(m, r)\}$ und ht die Gewichtsfunktion:
$p_X(k) = \frac{\binom{r}{k}\binom{n-r}{m-k}}{\binom{n}{m}}$

Satz 2.44
Eine wichtige und nützliche Eigenschaft der geometrischen Verteilung ist, dass sie gedächtnislos ist. Ist $X \sim Geo(p)$, so gilt für alle $s, t \in \mathbb{N}$:
$Pr[X \geq s + t | X > s] = Pr[X \geq t]$

Poisson-Verteilung
$X \sim Po(\lambda)$
Bsp: Modellierung seltener Ereignisse, zum Beispiel Anzahl Herzinfarkte in der Schweiz in der nächsten Stunde

$$f_X(i) = \begin{cases} \frac{e^{-\lambda}\lambda^i}{i!} \text{für i} \in \mathbb{N} \\ 0 \text{ sonst.} \end{cases} \quad (1)$$

$\mathbb{E}[X] = Var[X] = \lambda$
$Bin(n, \lambda/n)$ konvergiert für $n \to \infty$ gegen $Po(\lambda)$.

Wichtige Beispiele:
Coupon-Collector-Problem (S.114)

2.6 Mehrere Zufallsvariablen

Gemeinsame Dichte:
$f_{X,Y} := Pr[X = x, Y = y]$
Wenn man die gemeinsame Dichte gegeben hat, kann man auch wieder zu den Dichten der einzelnen Zufallsvariablen übergehen, indem man die Randdichten berechnet.

Randdichte:

Die Funktionen f_X, f_Y nennt man Randdichten: $f_X(x) = \sum_{y \in W_Y} f_{X,Y}(x,y)$
Analog dazu existieren auch gemeinsame Verteilung und Randverteilung (von der einzelnen ZV).

Gemeinsame Verteilung:
$F_{X,Y} := Pr[X \leq x, Y \leq y] = \sum_{x' \leq x} \sum_{y' \leq y} f_{X,Y}(x', y')$.

Unabhängigkeit mehrere ZV:
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:
$Pr[X_1 = x_1, ..., X_n = x_n] = Pr[X_1 = x_1] \cdot ... \cdot Pr[X_n = x_n]$.
Alternativ gilt dies auch für die gemeinsamen Dichtefunktionen. Bei unabhängigen ZV ist also die gemeinsame Dichte gleich dem Produkt der Randdichten.

Lemma:
Sind $X_1, ..., X_n$ unabhängige Zufallsvariablen und $S_1, ..., S_n$ beliebige Mengen mit $S_i \subseteq W_{X_i}$, dann gilt:
$Pr[X_1 \in S_1, ..., X_n \in S_n] = Pr[X_1 \in S_1] \cdot ... \cdot Pr[X_n \in S_n]$

Korollar (Teilmengen von unabh. ZV):
Sind $X_1, ..., X_n$ unabhängige Zufallsvariablen und ist $I = i_1, ..., i_k \subseteq [n]$, dann sind $X_{i_1}, ...X_{i_k}$ ebenfalls unabhängig.

Satz (Funktionen von ZV)
$f_1, ..., f_n$ seien reelwertige Funktoinen ($f_i : \mathbb{R} \to \mathbb{R}$ für $i = 1, ..., n$). Wenn die Zufallsvariablen $X_1, ..., X_n$ unabhängig sind, dann gilt dies auch für $f_1(X_1), ..., f_n(X_n)$.

Satz (Dichte von Z = X + Y)
Für zwei unabhängige ZV X, Y sei Z := X + Y. Es gilt:
$f_Z(z) = \sum_{x \in W_X} f_X(x) \cdot f_Y(z - x)$.
(Faltung / Konvolution der Dichten)

2.7 Ungleichungen: Abschätzen von Wahrscheinlichkeiten

Die folgenden Ungleichungen gelten für beliebige (diskrete oder allgemeine) Zufallsvariablen.

Satz: Ungleichung von Markov
Sei X eine Zufallsvariable, die nur nicht-negative Werte annimmt. Dann gilt für alle $t \in \mathbb{R}$ mit $t > 0$ und ferner $g : W(X) \to [0, \infty)$ eine wachsende Funktion mit $g(t) > 0$, dass:
$Pr[X \geq t] \leq \frac{\mathbb{E}[g(X)]}{g(t)}$.
Oder äquivalent dazu $Pr[X \geq t \cdot \mathbb{E}[g(X)]] \leq 1/g(t)$. Dabei ist $g(t)$ oftmals die Identitätsabbildung.

Satz: Ungleichung von Chebyshev
Sei X eine Zufallsvariable und $t \in \mathbb{R}$ mit $t > 0$. Dann gilt:
$Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{Var[X]}{t^2}$.
oder äquivalent dazu $Pr[|X - \mathbb{E}[X]| \geq t \cdot \sqrt{Var[X]}] \leq 1/t^2$.

Satz: Chernoff-Schranken
Seien $X_1, ..., X_n$ unabhängige Bernoulli-verteilte Zufallsvariablen mit $Pr[X_i = 1] = p_i$ und $Pr[X_i = 0] = 1 - p_i$. Dann gilt für $X := \sum_{i=1}^{n} X_i$:

1. $Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2 \mathbb{E}[X]}$ für alle $0 < \delta \leq 1$
2. $Pr[X \geq (1-\delta)\mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2 \mathbb{E}[X]}$ für alle $0 < \delta \leq 1$
3. $Pr[X \geq t] \leq 2^{-t}$ für $t \geq 2e\mathbb{E}[X]$

3 Wahrscheinlichkeit (allgemein) und Statistik

3.1 Bedingte Verteilungen

Def. bedingte Verteilung
Seien X und Y diskrete Zufallsvariablen mit gemeinsamer Dichtefunktion $p(x, y)$. Die bedingte Gewichtsfunkton von X, gegeben dass Y = y, ist definiert durch:
$p_{X|Y}(x|y) := P[X = x|Y = y] = \frac{P[X=x, Y=y]}{P[Y=y]} = \frac{p(x,y)}{p_Y(y)}$,
für $p_Y(y) > 0$ und 0 sonst.

3.2 Allgemeine Zufallsvariablen

Eine reellwertige ZV auf Ω ist eine messbare Funktion (messbar: Urbilder der Intervalle sind Ereignisse).

Verteilungsfunktion und Wahrscheinlichkeitsmass
Die Verteilungsfunktion von X ist nun eine nicht unbedingt diskrete Abbildung $F_X : \mathbb{R} \to [0, 1]$:
$F_X(t) := P[X \leq t]$.
Das Wahrscheinlichkeitsmass μ_X auf $\mathbb{R}$ ist definiert durch:
$\mu_X(B) := P[X \in B]$, wobei B ein Intervall ist.
Eigenschaften der Verteilungsfunktion: wachsend und rechtsseitig, GW gg. $-\infty$ ist 0, gg. ∞ ist der GW 1.

Dichtefunktion
Bei diskreten Zufallsvariablen haben wir meistens mit der Gewichtsfunktion gearbeitet, das Analogon im allgemeinen Fall ist die sogenannte Dichtefunktion, sofern eine existiert.
Eine ZV X mit Verteilungsfunktion $F_X(t) = P[X \leq t]$ heisst (absolut) stetig mit Dichte(funktion) $f_X : \mathbb{R} \to [0, \infty)$, falls gilt:

$F_X(t) = \int_{-\infty}^{t} f_X(s)ds$, für alle $t \in \mathbb{R}$.
Eigenschaften zu zeigen: $f_X(x) \geq 0$ und $\int f_X(x) = 1$.

3.2.1 Wichtige stetige Verteilungen

Gleichverteilung
Auf einem Intervall [a,b] mit selbigem Wertebereich ist die Dichtefunktion:
$f_X(t) = \begin{cases} \frac{1}{b-a}, a \leq t \leq b \\ 0, \text{sonst} \end{cases}$, und die Verteilungsfunktion:
$F_X(t) = \begin{cases} 0, t < a \\ \frac{t-a}{b-a}, a \leq t \leq b \\ 1, t > b \end{cases}$ $E[X] = \frac{a+b}{2}$

Exponentialverteilung
Die Exponentialverteilung mit Parameter $\lambda > 0$ ist das stetige Analogon der geometrischen Verteilung. Die zugehörige ZV X hat $W(X) = [0, \infty)$, Dichtefunktion:
$f_X(t) = \begin{cases} \lambda e^{-\lambda t}, t \geq 0 \\ 0, t < 0 \end{cases}$, und Verteilungsfunktion:
$F_X(t) = \begin{cases} 1 - e^{-\lambda t}, t \geq 0 \\ 0, t < 0. \end{cases}$

Normalverteilung
Die Normalverteilung (NV) oder auch Gauss-Verteilung genannt, spielt eine zentrale Rolle in vielen Anwendungen. Sie hat zwei Parameter Erwartungswert und Varianz $\mu \in \mathbb{R}, \sigma^2 > 0$. Die ZV X hat den WB $W(X) = \mathbb{R}$ und die Dichtefunktion:
$f_X(t) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(t-\mu)^2}{2\sigma^2})$.
Für $\mu = 0, \sigma^2 = 1$ erhalten wir die sog. Standardnormalverteilung.
$E[X] = \mu$

Cauchy-Verteilung
ZV X mit $W(X) = \mathbb{R}$ und der Dichtefunktion $f_X(x) = \frac{1}{\pi} \frac{1}{1+x^2}$ und der zugehörigen Verteilungsfunktion $F_X(x) = \frac{1}{2} + \frac{1}{\pi} \arctan(x)$.
Sind Y und Z unabhängige NV(0,1)-verteilte Zufallsvariablen, so kann man zeigen dass der Quotient $X = \frac{Y}{Z}$ Cauchy-verteilt ist.
Die Dichte der Cauchy-Verteilung geht für $|x| \to \infty$ nur sehr langsam gegen 0 (heavy-tailed distribution). Für die CV existiert kein EW, da die Dichte nicht absolut integrierbar ist.

3.2.2 Erwartungswerte

Ist X eine beliebige reellwertige Zufallsvariable, so kann man X immer durch eine Folge von diskreten Zufallsvariablen approximieren.
Ist X stetig mit Dichte $f_X(x)$, so liefert das Integral $E[X] = \int_{-\infty}^{\infty} x f_X(x)dx$,

den Erwartungswert. Ist das Integral nicht absolut konvergent, so existiert der Erwartungswert nicht (zumindest nicht in $\mathbb{R}$).
Damit ist der Erwartungswert einer allgemeinen Zufallsvariable als Grenzwert einer ZV mit endlich vielen Werten definiert.
Bsp: $E[XY] = \int\int xyf(x,y)$.

Funktion im Erwartungswert
Sei X eine ZV und Y = g(X) eine weitere ZV. Ist X stetig mit Dichte $f_X(x)$, so ist $E[Y] = E[g(X)] = \int_{-\infty}^{\infty} g(x)f_X(x)dx$, sofern das Integral absolut konvergiert.

Bedingter Erwartungswert
Für ein bestimmtes $x_i \in W_X$ gilt:
$E[Y|X = x_i] = \int_{W_Y} y \cdot f_Y(y|X = x_i)dy = \int_{W_Y} y \cdot \frac{f_{X,Y}(x_i,y}{f_X(x_i)}dy$.

3.2.3 Gemeinsame Verteilungen, unabhängige Zufallsvariablen

Auch dieser Abschnitt ist fast völlig parallel zum diskreten Fall.

Gemeinsame Verteilungsfunktion
Die gemeinsame Verteilungsfunktion von n Zufallsvariablen $X_1, ..., X_n$ ist die Abbildung $F : \mathbb{R}^n \to [0,1]$, mit $F(x_1, ..., x_n) = P[X_1 \leq x_1, ..., X_n \leq x_n]$. Dies entspricht dem n-fachen Integral von $-\infty$ bis $x_1, ..., x_n$.
Weitere Eigenschaften:

- $f(x_1, ..., x_n) \geq 0$, und $= 0$ ausserhalb von $W(X_1, ..., X_n)$.
- Das n fache Integral von $-\infty$ bis ∞ über alle Variablen der Dichtefunktion $f(x_1, ..., x_n)$ ist gleich 1 (Dichte).
- $P[(X_1, ..., X_n) \in A] = \int_{(x_1,...,x_n)\in A} \int f(x_1, ..., x_n)dx_n...dx_1$, für $A \subseteq \mathbb{R}^n$ (und davon nur Borel Mengen, d.h. Mengen, denen man naiverweise ein Volumen zuordnet).

Randverteilungen
Haben X und Y die gemeinsame Verteilungsfunktion F, so ist die Funkion $F_X : \mathbb{R} \to [0,1]$, $F_X(x) := P[X \leq x] = P[X \leq x, Y < \infty] = \lim_{y\to\infty} F(x,y)$ die Verteilungsfunktion der Randverteilung (RV) von X. Analoges gilt für Y.
Falls X und Y die gemeinsame Dichte f(x,y) haben,, so haben auch die Randverteilungen X und Y Dichten $f_X : \mathbb{R} \to [0,\infty)$. Analog zur Gewichtsfunktion p_x ist $f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy$.

Unabhängigkeit
Die Zufallsvariablen $X_1, ...X_n$ heissen unabhängig, falls gilt: $F(x_1, ..., x_n) = F_{X_1}(x_1) \cdot ....{}_{X_n}(x_n)$, d.h. die gemeinsame Verteilungsfunktion ist das Produkt der Verteilungsfunktionen der Randverteilungen.
Hat man stetige Zufallsvariablen mit Dichten, so ist das (analog zum diskreten

Fall) äquivalent zu $f(x_1, ..., x_n) = f_{X_1} ... f_{X_n} \forall x_1, ..., x_n$, d.h. die gemeinsame Dichtefunktion ist das Produkt der einzelnen Randdichten.

3.2.4 Funktionen und Transformationen von Zufallsvariablen

Wir untersuchen hier analog zum diskreten Fall wie sich die Summe von Zufallsvariablen verhält. Sei Z = X + Y, so gilt für die Verteilungsfunktion: $F_Z(z) = \int_{-\infty}^{\infty} \int_{-\infty}^{z-x} f(x,y) dy dx$.
Mit der Variablentransformation v = x + y und dy = dv gelangt man zur auch zu einer Dichte von Z, und zwar: $f_Z(z) = \frac{d}{dz} F_Z(z) = \int_{-\infty}^{\infty} f(x, z-x) dx = \int_{-\infty}^{\infty} f(z-y, y) dx$.
Sind zusätzlich X und Y unabhängig, so ist $f(x,y) = f_X(x) f_Y(y)$ und damit wieder wie im diskreten Fall:
$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z-x) dx = \int_{-\infty}^{\infty} f_X(z-y) f_Y(y) dy =: (f_X(x) * f_Y(y))(z)$, wobei dies die Faltung darstellt.

Affine und nichtlineare Transformationen
Affine Transformation, i.e. g(x) = ax + b, mit $a > 0, b \in \mathbb{R}$. Dann gilt für $Y = g(X) = aX + b$:
$F_Y(t) = P[Y \leq t] = P[aX + b \leq t] = P[X \leq \frac{t-b}{a}] = F_X(\frac{t-b}{a})$ und damit nach der Kettenregel: $f_Y(t) = \frac{d}{dt} F_Y(t) = \frac{1}{a} f_X(\frac{t-b}{a})$. Analog bis auf ein Vorzeichen geht das auch mit $a < 0$.
Nichtlineare Transformation, bsp. für $Y = X^2$ gilt $F_Y(t) = P[X^2 \leq t] = P[-\sqrt{t} \leq X \leq \sqrt{t} = F_X(\sqrt{t}) - F_X(-\sqrt{t})$. Mit Hilfe der Kettenregel können wir dann die Dichte berechnen.

Satz zur Umkehrfunktion
Sei F eine stetige und streng monoton wachsende Verteilungsfunktion, mit Umkehrfunktion F^{-1}. Ist $X \sim U(0,1)$ und $Y = F^{-1}(X)$, so hat Y gerade die Verteilungsfunktion F.
Der Satz erlaubt die explizite Konstruktion einer Zufallsvariablen Y mit einer gewünschten Verteilungsfunktion F, wenn man eine $U(0,1)$-verteilte ZV X hat. Das liefert einen Simulationsalgorithmus nach der Inversionsmethode (z.B. Zufallszahlengenerator, der eine Folge von Zahlen produziert, die sich in einem gewissen Sinn verhält wie die Realisierung einer Folge von unabhängigen $U(0,1)$-verteilten Zufallsvariablen).

3.3 Grenzwertsätze

3.3.1 Das Gesetz der großen Zahlen

Schwaches Gesetz der grossen Zahlen
Sei $X_1, X_2, ...$ eine Folge von unabhängigen (tatsächlich reicht paarweise unkorreliert, d.h. $Cov(X_i, X_k) = 0$ für $i \neq k$) Zufallsvariablen, die alle den gleichen Erwartungswert $E[X_i] = \mu$ und die gleiche Varianz $Var[X_i] = \sigma^2$ haben. Sei $\bar{X}_n = \frac{1}{n} S_n = \frac{1}{n} \sum_{i=1}^{n} X_i$. Dann konvergiert $\bar{X}_n$ für $n \to \infty$ in Wahrschein-

lichkeit/stochastisch gegen $\mu = E[X_i]$, d.h.
$P[|\bar{X_n} - \mu| > \epsilon] \to_{n\to\infty} 0$, für jedes $\epsilon > 0$.

Starkes Gesetz der grossen Zahlen
Sei $X_1, X_2, ...$ eine Folge von unabhängigen Zufallsvariablen, die alle dieselbe Verteilung haben, und ihr Erwartungswert $\mu = E[X_i]$ sei endlich. Für $\bar{X_n} = \frac{1}{n}S_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ gilt dann:
$\bar{X_n} \to_{n\to\infty} \mu$ P-fastsicher (P-f.s.), d.h.
$P[\{\omega \in \Omega \bar{X_n}(\omega) \to_{n\to\infty}\}] = 1$.
Das heisst, dass z.B. in der Monte Carlo-Integration unsere Simulation nicht nur mit grosser Wahrscheinlichkeit, sondern mit Wahrscheinlichkeit 1 nahe bei I (Integralwert) liegt. Es kann uns also im Prinzip immer noch passieren, dass wir eine schlechte Realisierung erwischen, aber das passiert mit Wahrscheinlichkeit 0.

3.3.2 Der Zentrale Grenzwertsatz

Hier behandeln wir die Frage nach der Asymptotik für die Verteilung einer Summe von vielen "gleichartigen" Zufallsvariablen.

Zentraler Grenzwertsatz (ZGS)
Sei $X_1, X_2, ...$ eine Folge von i.i.d. Zufallsvariablen mit $E[X_i] = \mu$ und $Var[X_i] = \sigma^2$. Für die Summe $S_n = \sum_{i=1}^{n} X_i$ gilt dann:
$\lim_{n\to\infty} P[\frac{S_n - n\mu}{\sigma\sqrt{n}} \leq x] = \Phi(x)$, für alle $x \in \mathbb{R}$, wobei Φ die Verteilungsfunktion der $N(0,1)$-Verteilung ist. $\frac{S_n - n\mu}{\sigma\sqrt{n}}$ entspricht der Standardisierung von S_n.
Dies entspricht auch der Verteilung der Fehler bei der Monte-Carlo Integration.

4 Randomisierte Algorithmen

Klassischer (determinisitscher) Algorithmus:
Eingabe I $\to$ Alg $\mathbb{A} \to$ Ausgabe $\mathbb{A}(I)$

Wir beweisen:
1. Korrekthet für alle Eingaben I
2. Laufzeit für alle Eingaben I mit Länge $|I| = n$: Laufzeit O(f(n))

Ein randomisierter Algorithmus hat (in dieser Vorlesung) Zugriff auf sogenannte Zufallsbits. Diese sind das Ergebnis einer Bernoulli-verteilten Zufallsvariablen mit Parameter 1/2. Dadurch erhalten wir Zufallszahlen $\mathbb{R}$.
Eigenschaften:
Ausgabe $\mathbb{A}(I, \mathbb{R})$ hängt von der Eingabe I und den Zufallszahlen $\mathbb{R}$ ab. Insbesondere lässt sich ein Ergebnis i.A. nicht reproduzieren.

Wir beweisen:
1. Korrekthet für alle Eingaben I gilt: $Pr[\mathbb{A}(I, \mathbb{R})]$ ist korrekt $\geq ...$

2. Laufzeit für alle Eingaben I mit Länge $|I| = n$: Laufzeit O(f(n)) und / oder $Pr_{\mathbb{R}}[Laufzeit \leq O(f(n))] \geq ...$
Idealerweise sind die Wahrscheinlichkeiten "praktisch" Eins.

Las-Vegas Algorithmen
Geben nie eine falsche Antwort, aber zuweilen keine Antwort (Ausgabe = ???). Die Laufzeit ist dabei eine Zufallsvariable und da wir nach einer bestimmten Zeit abbrechen, geben wir ??? aus.
Ziel: Pr[Antwort = ???] = "winzig"

In der Praxis ruft man Las-Vegas Alg. meist so lange auf, bis sie eine Antwort geben.
$Pr[Antwort =???] \leq 1 - \epsilon$
$\rightarrow \epsilon^{-1} ln\delta^{-1}$ Wiederholungen reduzieren Fehler auf $Pr[Antwort =???] \leq \delta$

Bsp.: QuickSort, QuickSelect

Monte-Carlo Algorithmen
Geben immer eine Anwort aber zuweilen eine falsche.
Ziel: Pr[Antwort falsch] = "winzig"

Monte-Carlo-Algorithmen für Entscheidungsprobleme (binäre Antwort):
Einseitiger Fehler:
$Pr[Antwortfalsch|Antwort = Ja] = 0$,
$Pr[Antwortfalsch|Antwort = Nein] \leq 1 - \epsilon$
$\rightarrow \epsilon^{-1} ln\delta^{-1}$ Wiederholungen reduzieren Fehler auf $Pr[Antwortfalsch] \leq \delta$
Antwort Ja: wenn ein Aufruf Ja ausgibt, Antwort Nein: wenn alle Wdh. Nein ausgeben.

Zweiseitiger Fehler:
Bedingung: Fehlerwahrscheinlichkeit des Alg. von Anfang an strikt kleiner als 1/2.
$Pr[Antwortfalsch] \leq 1/2 - \epsilon$
$\rightarrow 4\epsilon^{-2} ln\delta^{-1}$ unabhängige Aufrufe, nach denen man die Mehrzeit der erhaltenen Antworten ausgibt, so gilt: $Pr[Antwortfalsch] \leq \delta$.

Siehe auch Bsp. Primzahltest

Satz 2.75 (Maximierungsproblem)
Sei $\epsilon > 0$ und $\mathbb{A}$ ein randomisierter Algorithmus für ein Maximierungsproblem, wobei gelte:
$Pr[\mathbb{A}(I) \geq f(I)] \geq \epsilon$.
Dann gilt für alle $\delta > 0$: bezeichnet man mit $\mathbb{A}_\delta$ den Algorithmus, der
$N = \epsilon^{-1} ln\delta^{-1}$ unabhängige Aufrufe von $\mathbb{A}$ macht und die beste der erhaltenen

Antworten ausgibt, so gilt für den Algorithmus $\mathbb{A}_\delta$, dass:
$Pr[\mathbb{A}_\delta \geq f(I)] \geq 1 - \delta$.
Für Minimierungsprobleme gilt eine analoge Aussage vonn wir $\geq f(I)$ mit $\leq f(I)$ ersetzen.

Quickselect
Finde k-kleinstes Element in O(n) Zeit. Beweis im Skript.

Algorithm 5: Quickselect(A,l,r,k)

```
1 p = Uniform(l,l+1,...,r) // wähle Pivot zufällig
2 t = Partition(A,l,r,p)
3 if (k = t) then
4   return A[t] // gesuchtes Element gefunden
5 else if k < t then
6   return Quickselect(A,l,t-1,k)
7 else
8   return Quickselect(A,t+1,r,k-t)
```

Satz:
- Quickselect bestimmt immer das richtige Ergebnis
- $\mathbb{E}$[Laufzeit] = O(n)

Gedankenexperiment:
t_n := erwartete Laufzeit von Quickselect bei n Elementen.
Wir definieren einen neuen Alg. wie folgt:

Algorithm 6: SuperQuickSelect(A,1,n,k)

```
1 rufe Quickselect(A,1,n,k) auf
2 sobald Laufzeit grösser als 2 · t_n:
3   breche Ausführung ab und gebe ??? aus
```

Dies ist ein randomisierter Algorithmus mit Laufzeit $\leq 2t_n$ und Wahrscheinlichkeit für ??? $\leq 1/2$. Wir können dies weiterführen mit SuperSuperQuickSelect und rufen den Algorithmus SuperQuickSelect 100 mal auf. Dann haben wir eine Laufzeit $\leq 200t_n$ und eine Wahrscheinlichkeit für ??? $\leq 2^{-100}$

Target Shooting (Bsp. für Optimierungsalgorithmen)
Gegeben eine Menge U und eine Untermenge $S \subseteq U$. Uns interessiert die Größe $|S|/|U|$.
Annahmen: Wir können ein Element aus U effizient zufällig gleichverteilt wählen

und es gibt eine effizient berechenbare Funktion:

$$\mathbb{I}_S(u) := \begin{cases} 1 \text{ falls } u \in S \\ 0 \text{ sonst} \end{cases}$$

Algorithm 7: Target-Shooting

1 wähle $U_1, ..., U_N \in U$ zufällig, gleichverteilt und unabhängig
2 return $N^{-1} \cdot \sum_{i=1}^{N} I_S(u_i)$

Definiere $Y_i := I_S(U_i)$ für alle i = 1,...N
$Y_1, ...Y_N$ sind unabhängige Bernoulli-Variablen mit $Pr[Y_i = 1] = |S|/|U|$
$Y := \frac{1}{N} \sum_{i=1}^{N} Y_i$.
Dann gilt: $\mathbb{E}[Y] = |S|/|U|$ ... unabhängig von der Wahl von N
$Var[Y] = \frac{1}{N}(\frac{|S|}{|U|} - (\frac{|S|}{|U|})^2)$

Satz:
Seien $\delta, \epsilon > 0$. Falls N "gross" ist, $N \geq 3\frac{|U|}{|S|}\epsilon^{-2}log(2/d)$, so ist die Ausgabe des Algorithmus Target-Shooting mit Wahrscheinlichkeit mindestens $1 - \delta$ im Intervall $[(1 - \epsilon)\frac{|S|}{|U|}, (1 + \epsilon)\frac{|S|}{|U|}]$.

Hashing
Idee: eine Hashfunktion bildet eine (potentiell sehr grosse) Datenmenge auf eine (kleine) natürliche Zahl ab.
Gewünschte Eigenschaften:
- alle Hashwerte sollen "gleich oft" vorkommen
- "geringe" Wsk. von Kollisionen
- ähnliche Eingaben sollen zu verschiedenen Ergebnissen führen
- Hashfunktion f soll effizient berechenbar sein
- (Krpyto) f^{-1} soll nicht effizient berechenbar sein

Theorie: In der Analyse von Algorithmen geht man meist davon aus, dass Hashfunktionen die gegebenen Daten zufällig und unabhängig voneinander auf {0,...,m-1} abbilden.
Praxis: Man wählt eine Hashfunktion f aus einer vorgegebenen Menge von Funktionen (universelle Hashklasse) zufällig.
Man geht davon aus, dass die Funktion f die in der theoretischen Analyse gemachten Annahmen erfüllt.

Probleme die es zu lösen gilt:
- Konstruktion einer Hashfunktion (z.B. mod m)
- Umgang mit Kollisionen; notwendig: theoretisches Verständnis, "was überhaupt möglich" ist

Minimum Cut
Wir betrachten Multigraphen G=(V,E).
Frage: Wie viele Kanten müssen aus einem Graph entfernt werden, bis er nicht mehr zusammenhängend ist ?
Schnitt (Cut) : Partition der Knoten in zwei Klassen (nicht mehr zusammenhängend)
Grösse des Schnitts : Anzahl Kanten zwischen den Klassen
Anwendungen: Netzwerk-Robustheit, Parallelisierung, Clustering, Kombinatorische Optimierung

Kantenkontraktion:
- Kante {u,v}: Ersetze u,v durch Knoten $x_{u,v}$
- Behalte alle Kanten ausser loops

Algorithm 8: Kargers Algorithmus: Cut(G), G zusammenh. Multigraph

```
1 while |V(G)| > 2 do
2   e ← gleichverteilt zufällige Kante in G
3   G ← G / e
4 return Grösse des eindeutigen Schnitts in G
```

- Findet Min-Cut mit Wsk. mind. $1/n^2$
- Wiederhole Cn^2 mal, um Min-Cut zu finden → Monte-Carlo-Alg.

In einem Graph mit n Knoten gibt es 2^n Schnitte. Warum funktioniert es so gut?
Lemma:
Algorithmus findet Min-Cut, wenn er nur Kanten ausserhalb des Cuts kontrahiert.
Theorem:
Wiederhole Cut(G) Cn^2 mal, finde Min-Cut mit Wsk. mind. $1 - e^{-c}$

Implementierung:
- n Kontraktionen pro Cut(G)
- Cn^2 Wdh.
- Zeit pro Kontraktion: O(n) (z.B. mit Adjacency-List)
Laufzeit: $O(Cn^4)$

Bootstrapping:
Idee: In sich selbst einsetzen eines Algorithmus.
Beobachtung: Fehlerwsk. ist am Ende am grössten. Daher verwenden wir SuperSuper...Karger: Anstatt die Grösse zu returnen, rufen wir ab einer bestimmten Grösse t einen anderen Min-Cut Algorithmus auf (int O(z(t)), mit Karger Erfolgswsk. mind. 1/2.
Laufzeit $O(n^2 log^3(n))$

Primzahltest
Gegeben eine Zahl n, finde heraus ob n prim ist. Naiver Algorithmus (Teiler ausprobieren) ist zu langsam.

Kleiner fermat'scher Satz:
n prim und $a \in \{1, ..., n-1\} \rightarrow a^{n-1} \equiv 1 \pmod{n}$

Problem: es existieren (unendlich viele) sog. Carmichael-Zahlen, für die der kleine fermat'sche Satz kein korrektes Ergebnis liefert (für alle Zahlen falsch).

Lösung: Miller-Rabin-Primzahltest
Idee: wenn n eine Primzahl ist, dann bilden die Zahlen $0 \leq a < n$ bezüglich der Addition und Multiplikation mod n einen Körper. Das heisst insbesondere, dass die Kongruenz $x^2 \equiv 1 \pmod{n}$ für $0 \leq x < n$ genau die zwei Lösungen x = 1 und x = n - 1 hat.
Diese Idee können wir mit dem kleinen fermat'schen Satz kombinieren. Dazu schreiben wir zuerst $n - 1 = d2^k$, wobei d ungerade ist. Ist n prim, dann muss nach dem kleinen fermat'schen Satz gelten: $a^{n-1} = (a^d)^{2^k} \equiv 1 \pmod{n}$. Dann ist aber wegen der vorhergehenden Beobachtung entweder $(a^d)^{2^{k-1}} \equiv 1 \pmod{n}$ oder $(a^d)^{2^{k-1}} \equiv n - 1 \pmod{n}$. Durch iterieren sieht man leicht: entweder gilt für alle $0 \leq i \leq k$ die Kongruenz $(a^d)^{2^i} \equiv 1 \pmod{n}$, oder aber es gibt ein $(a^d)^{2^i} \equiv n - 1 \pmod{n}$. Sind beide Bedingungen verletzt, so sagen wir, dass a ein Zertifikat für die Zusammengesetztheit von n ist.

MILLER-RABIN-PRIMZAHLTEST(n)

```
 1: if n = 2 then
 2:     return 'Primzahl'
 3: else if n gerade oder n = 1 then
 4:     return 'keine Primzahl'
 5: Wähle a ∈ {2, 3, ..., n − 1} zufällig und
 6: berechne k, d ∈ ℤ mit n − 1 = d2^k und d ungerade.
 7: x ← a^d (mod n)
 8: if x = 1 or x = n − 1 then
 9:     return 'Primzahl'
10: repeat k − 1 mal
11:     x ← x^2 (mod n)
12:     if x = 1 then
13:         return 'keine Primzahl'
14:     if x = n − 1 then
15:         return 'Primzahl'
16: return 'keine Primzahl'
```

- hat Erfolg mit Wsk- mind. 3/4
- einfach implementierbar
- kurze Laufzeit
- Monte-Carlo (kann falsch liegen)

5 Algorithmen - Highlights

5.1 Graphenalgorithmen

Lange Pfade
Eingabe: G = (V,E), $k \in \mathbb{N}$, Färbung $c : V \to [k]$
Frage: Gibt es in G einen Pfad der Länge k

Um zu zeigen, dass das Problem schwer ist, konstruieren wir einen Graph G' mit $n' \leq 2n-2$ Knoten, so dass G einen Hamiltonkreis hat, gdw. G' einen Pfad der Länge n hat (Vorgehen Skript S. 148).

Satz
Falls wir LONG-PATH für Graphen mit n Knoten in t(n) Zeit entscheiden können, dann können wir in $t(2n-2) + O(n^2)$ Zeit entscheiden, ob ein Graph mit n Knoten einen Hamiltonkreis hat.

Kurze lange Pfade
Hierbei betrachten wir Pfade mit einer Pfadlänge $k = log(n)$. Für dieses problem existiert tatsächlich ein polynomieller Algorithmus mit einer sehr eleganten randomisierten Methode, dem sog. Color Coding.

Hilfsmittel:

1. Die k Knoten auf einem Pfad der Länge k-1 kann man mit [k] auf genau k^k Arten färben, k! dieser Färbungen nutzen jede Farbe genau einmal.
2. Für $c, n \in \mathbb{R}$ gilt: $c^{logn} = n^{logc}$. Also, z.B. $2^{logn} = n^{log2} = n$ und $2^{O(logn)} = n^{O(1)}$ ist immer polynomiell in n.
3. Für $n \in \mathbb{N}_0$ gilt: $\sum_{i=0}^{n} \binom{n}{i} = 2^n$, eine Anwendung des Binomialsatzes: $\sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} = (x+y)^n$.
4. Für $n \in \mathbb{N}_0$ gilt: $\frac{n!}{n^n} \geq e^{-n}$, was man leicht der Potenzreihenentwicklung der Exponentialfunktion ablesen kann.

Bunte Pfade - Color Coding
Ein Pfad in G heisst bunt, wenn alle Knoten auf dem pfad verschiedene Farben haben. Eine Färbung ist eine Abbildung $\gamma : V \to [k]$.
Lösungsidee: Dynamische Programmierung
Dazu definieren wir $\forall v \in V$ und $\forall i \in [k]$:
$P_i(v) := \{S \in \binom{[k]}{i+1} | \exists$ in v endender genau mit S gefärbter bunter Pfad$\}$
$P_i(v)$ enthält also eine Menge S von i+1 Farben gdw. es einen bunten Pfad mit Endknoten v gibt, dessen Farben genau die Farben in S sind. Beachte, ein solcher Pfad muss immer die Länge genau i haben.

Wir berechnen also ausgehend von den Mengen $P_0(v) = \{\gamma(v)\}\}$, welche mit den Nachbarschaftsfarben initialisiert werden, sukzessive in k-1 Runden die Mengen

Algorithm 9: Bunt(G,i)

```
1 for all v ∈ V do
2   for all x ∈ N(v) do
3     for all R ∈ P_{i-1} mit γ(v) ∉ R do
4       P_i(v) ← P_i(v) ∪ {R ∪ {γ(v)}}
```

$P_{k-1}(v)$.

Satz (Color Coding)
Sei G ein Graph mit einem Pfad der Länge k-1.
(1) Eine zufällige Färbung mit k Farben erzeugt einen bunten Pfad der Länge k-1 mit Wahrscheinlichkeit $p_{Erfolg} \geq e^{-k}$.
(2) Bei wiederholten Färbungen mit k Farben ist der Erwartungswert der Anzahl Versuche bis man einen bunten Pfad der Länge k-1 erhälrt $\frac{1}{p_{Erfolg}} \leq e^k$.

Daher können wir jetzt leicht einen Monte-Carlo Algorithmus bauen, der unser Problem in polynomieller Zeit löst. Dazu wählen wir ein $\lambda \in \mathbb{R}, \lambda > 1$, und wiederholen unseren Test höchstens λe^k mal, bis wir eine Bestätigung haben, dass es einen Pfad der Länge k-1 gibt. Gelingt dies, dann Antwort JA, scheitern wir, dann Antwort NEIN.

Satz 3.3 (Laufzeit und Erfolgswsk.)
(1) Der Algorithmus hat eine Laufzeit von $O(\lambda(2e)^k km)$.
(2) Antwortet der Algorithmus mit JA, dann hat der Graph einen Pfad der Länge k-1.
(3) Hat der Graph einen Pfad der Länge k-1, dann ist die Wahrscheinlichkeit, dass der Algorithmus mit NEIN antwortet, höchstens e^{-k}.

Flüsse in Netzwerken
Anwendungen: Data Mining, Bilverarbeitung, etc.
Definiton Netzwerk
Ein Netzwerk ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt:
(V,A) ist ein gerichteter Graph, $s \in V$, die Quelle (source), $t \in V \setminus s$ die Senke (target), und $c : A \to \mathbb{R}_0^+$ die Kapazitätsfunktion.

Definition Fluss
Gegeben sei ein Netzwerk N. Ein Fluss ist eine Funtion $f : A \to \mathbb{R}$ mit den Bedingungen:
Zulässigkeit: $0 \leq f(e) \leq c(e)$ für alle $e \in A$
Flusserhaltung: $\sum_{u \in V:(u,v) \in A} f(u,v) = \sum_{u \in V:(v,u) \in A} f(v,u)$
Der Wert (value) eines Flusses f (Ausfluss in s) ist durch
$val(f) :=$ netoutflow(s) $:= \sum_{u \in V:(s,u) \in A} f(s,u) - \sum_{u \in V:(u,s) \in A} f(u,s)$ definiert.

Es gibt eine weitere Möglichkeit den Fluss zu definieren:

Lemma
Der Nettozufluss der Senke gleicht dem Wert des Flusses, d.h.
netinflow(t) := $\sum_{u \in V:(u,t) \in A} f(u,t) - \sum_{u \in V:(t,u) \in A} f(t,u) = val(f)$

Das algorithmische Problem ist nun, für ein Netzwerk den maximalen Fluss zu berechnen. Dabei ist Vorsicht geboten, da nicht klar ist ob solch ein maximaler Fluss existert. Ähnlich wie das offene Intervall (0,1) keine grösste Zahl hat, könnte es eine steigende Folge von Werten von Flüssen geben, deren Grenzwert aber nicht dem Wert eines Flusses entspricht. Offensichtlich gibt es im Allgemeinen unendlich viele Flüsse.

Schnitte
Definition s-t-Schnitt
Ein s-t-Schnitt für ein Netzwerk (V,A,c,s,t) ist eine Partition (S,T) von V (d.h. $S \cup T = V$ und $S \cap T = \emptyset$) mit $s \in S$ und $t \in T$.
Die Kapazität eines s-t-Schnitts (S,T) ist definiert durch:
$cap(S,T) := \sum_{(u,w) \in (SxT) \cap A} c(u,w)$

Lemma 3.8
Ist f ein Fluss und (S,T) ein s-t-Schnitt in einem Netzwerk (V,A,c,s,t), so gilt $val(f) \leq cap(S,T)$.

Satz 3.9 Maxflow-Mincut-Theorem
Jedes Netzwerk N erfüllt: $max\ val(f) = min\ cap(S,T)$
Wobei man das Maximum über alle Flüsse f in N und das Minimum über alle (S,T)-Schnitte in N wählt.
Beweis für Netzwerke ohne entgegen gesetzte Kanten und ganzzahligen Kapazitäten.

Augmentierende Pfade
Die Grundidee fast aller Flussmaximierungsalgorithmen ist, mit einem beliebigen Fluss zu beginen (f konstant 0 eignet sich immer) und diesen sukzessive zu verbessern. Ein erster Ansatz: finden wir einen gerichteten Pfad von Quelle zu Senke, wo der Fluss auf allen Kanten die Kapazität noch nicht erschöpft hat und $\delta := min_{e \in P} c(e) - f(e)$, dann können wir den Fluss auf allen Kanten um δ erhöhen.
Allerdings gibt es Flüsse, die sich nach diesem Schema nicht verbessern lassen, obwohl sie nicht optimal sind. In diesem Sinne kann man nun Pfade von Quelle zu Senke betrachten, auf denen Kanten vorwärts und rückwärts gerichtet sein können. Nehmen wir an, dass wir auf jeder vorwärts gerichteten Kante den Fluss um δ erhöhen können ohne die Zulässigkeit zu verletzen, und auf jeder rückwärts gerichteten Kante den Fluss um δ verkleinern können, ohne dass dieser negativ wird. Dann können wir entlang dieses sog. augmentierenden pfades den Fluss entsprechend modifizieren ohne die Flusserhaltung zu verlet-

zen. Verbessert man den Fluss mittels solcher aug. Pfade, so kann man nur in maximalen Flüssen stecken bleiben (was man beweisen muss). Das heisst aber nicht, dass man einen maximalen Fluss in endlich vielen Schritten erreicht (dies gilt interessanterweise nur, wenn die Kapazitäten rational sind).

Restnetzwerk
Wir machen vereinfachend folgende Annahme: Wir beschränken uns auf Netzwerke ohne entgegen gerichtete Kanten.
Definition
Sei N = (V,A,c,s,t) ein Netzwerk ohne entgegen gerichtete Kanten und sei f ein Fluss in N. Das Restnetzwerk $N_f := (V, A_f, r_f, s, t)$ ist wie folgt definiert:
(1) Ist $e \in A$ mit $f(e) \leq c(e)$, dann ist e auch eine Kante in A_f, mit $r_f(e) := c(e) - f(e)$.
(2) Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} A_f, mit $r_f(e^{opp}) = f(e)$.
(3) Nur Kanten wie in (1) und (2) beschrieben finden sich in A_f. $r_f(e), e \in A_f$, nennen wir die Restkapazität der Kante e.

Satz 3.11
Ein Fluss f in einem Netzwerk N ist ein maximaler Fluss gdw. es im Restnetzwerk N_f keinen gerichteten Pfad von der Quelle s zur Senke t gibt. Für jeden solchen maximalen Fluss gibt es einen s-t-Schnitt (S,T) mit val(f) = cap(S,T).

Algorithm 10: Ford-Fulkerson(V,A,c,s,t)

```
1 f ← 0
2 while ∃ s-t-Pfad P in (V, A_f) do (aug. Pfad)
3   Erhöhe den Fluss entlang P
4 return f
```

$$f'(e) = \begin{cases} f(e) + \epsilon, \text{ e auf P} \\ f(e) - \epsilon, \ e^{opp} \text{ auf P, und} \\ f(e), \text{ sonst.} \end{cases} \tag{2}$$

Suche mit Hilfe des Restnetzwerks augmentierende Pfade, solange es solche gibt. Erhöhe entlang P um diese Werte. Wahl von ϵ: $\epsilon := min_{e \in P} r_f(e)$

Der Ford-Fulkerson löst das Problem allerdings nur bedingt. Bei irrationalen Kapazitäten ist es möglich, dass er nicht terminiert. Bei ganzzahligen hingegen terminiert er immer.
Jede Iteration verändern wir die Flusswerte im Netzwerk und updaten das Restnetzwerk. D.h. Wenn eine Kante im Restnetzwerk Kapazität 10 hat und 1 subtrahiert wird, entsteht eine Kante mit Kapazität 9 und einer Gegenkante mit Kapazität 1 (siehe Slides).

Satz 3.12
Sind in einem Netzwerk ohne entgegen gerichtete Kanten alle Kapazitäten ganzzahlig und höchstens U, so gibt es einen ganzzahligen maximalen Fluss, der in Zeit O(mnU) berechnet werden kann (m ist die Anzahl Kanten, n die Anzahl Knoten im Netzwerk).

Algorithmus von Edmonds-Karp

Algorithm 11: Edmonds-Karp(V,A,c,s,t)

```
1 f ← 0
2 while ∃ s-t-Pfad P in (V, A_f) do (aug. Pfad)
3   Wähle einen kürzesten solchen Pfad P (z.B. mit BFS)
4   Erhöhe den Fluss entlang P
5 return f
```

Satz
Algorithmus von Edmonds-Karp hat Laufzeit $O(nm^2)$. Ausblick: Orlin (2013) löst das Problem in O(nm).

Bipartites Matching als Flussproblem
Sei G = (V,E) ein bipartiter Graph. Wir definieren ein Netzwerk
$N = (V \cup \{s,t\}, A, c, s, t)$. Die Knotenmenge besteht aus den Knoten V von G und zwei neuen Knoten s und t, die die Rolle von Quelle und Senke in N spielen. Die Kapazitätsfunktion ist konstant 1. Die Kanten aus E werden übernommen, immer von U nach W gerichtet. Ausserdem hat s Kanten zu allen Knoten in U; zu t fügen wir Kanten von allen Knoten in W ein. Wir beachten nun einen ganzzahligen Fluss f auf diesem Netzwerk und beachten, dass alle Kapazitäten 1 sind.

Lemma 3.15
Die maximale Grösse eines Matchings im bipartiten Graph G ist gleich dem Wert eines maximalen Flusses im Netzwerk N.

Bildsegmentierung als Schnittproblem
Wir wollen ein Bild bestehend aus Pixeln in Vordergrund und Hintergrund unterteilen (Segmentierung). Die Pixelmenge P ist mit einer Nachbarschaftsrelation assoziiert. Die Kantenmenge zwischen den Pixeln bildet einen Graphen G = (P, E).
Wir extrahieren für jedes Pixel p zwei nichtnegative Zahlen α_p, β_p. α_p drückt unsere Erwartung aus, dass p zum Vordergrund gehört, β_p analog zum Hintergrund. Mit diesen Zahlen scheint die Partition P in Vordergrund A und Hintergrund B einfach:
$A := \{p \in P | \alpha_p > \beta_p\}$ und $B := P \setminus A$.
Allerdings wollen wir keine zu feinkörnige Unterteilung, d.h. liegen viele Nach-

barn eines Pixels p im Vordergrund, sollte p auch eher im Vordergrund liegen. Diese Präferenz modellieren wir, indem wir jeder Kante $e \in E$ eine nichtnegative zahl γ_p zuordnen, mit der Interpretation, dass je grösser γ_p ist, wir umso mehr erwarten, dass die beiden beteiligten Pixel im gleichen Teil der Segmentierung landen.
Mit diesen Zahlen bewerten wir nun eine Partition (A,B) mitels der Qualitätsfunktion $q(A,B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E, |e \cap A| = 1} \gamma_p$.
Wir sind daran interessiert q zu maximieren.
Es gilt: $q'(A,B) = cap(S,T)$.
Um das Problem zu lösen, modellieren wir das Problem wie folgt. Wir führen neue Knoten s und t ein, wobei s gerichtete Kanten zu allen Pixeln in P mit Kap. α_p hat. Analog hat jedes Pixel eine mit β_p gewichtete gerichtete Kante zu t. Schliesslich ersetzen wir jede ungerichtete Kante $e = \{p, p'\}$ in E durch zwei gerichtete Kanten (p,p') und (p',p) mit je Kapazität γ_p.
Eine optimale Segmentierung entspricht somit einem minimalen s-t-Schnitt im generierten Netzwerk N (kann mit Fluss- oder Schnittalgorithmus berechnet werden).

Flüsse und konvexe Mengen

Lemma 3.16
Sind f_0 und f_1 Flüsse in einem Netzwerk N und $\lambda \in \mathbb{R}$, $0 < \lambda < 1$, dann ist der Fluss f_λ definiert durch:
$\forall e \in A : f_\lambda(e) := (1-\lambda) f_0(e) + \lambda f_1(e)$,
ebenfalls ein Fluss in N.
Es gilt:
$val(f_\lambda) = (1-\lambda) \cdot val(f_0) + \lambda \cdot val(f_1)$.

Korollar 3.17
Es gilt:
(i) Ein Netzwerk N hat entweder genau einen Fluss (den Fluss 0) oder unendlich viele Flüsse.
(ii) Ein Netzwerk hat entweder genau einen maximalen Fluss, oder unendlich viele maximale Flüsse.

Konvexe Mengen
Für eine geometrische Interpretation des Flusses wählen wir eine Nummerierung (Ordnung) $(e_1, e_2, ..., e_m)$, $m := |A|$, der Kanten in A.
Jede Funktion $f : A \to \mathbb{R}$ induziert so einen Vektor $v_f = (f(e_1), f(e_2), ..., f(e_m))$ in $\mathbb{R}^m$. Das heisst, die Menge der Flüsse (oder die Menge der maximalen Flüsse) kann man so als Teilmengen des $\mathbb{R}^m$ interpretieren.

Definition: Liniensegment, konvex
Sei $m \in \mathbb{N}$.
(i) Für $v_0, v_1 \in \mathbb{R}^m$ sei $\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 | \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\}$, das v_0, v_1 verbindende Liniensegment.
(ii) Eine Menge $C \subseteq \mathbb{R}^m$ heisst konvex, falls für alle $v_0, v_1 \in C$ das ganze Liniensegment $\overline{v_0 v_1}$ in C enthalten ist.
Beispiele konvexer Mengen sind Kugeln oder konvexe Polytope (z.B. Tetraeder oder Würfel im $\mathbb{R}^3$.

Satz 3.19
Die Menge der Flüsse eines Netzwerks mit m Kanten, interpretiert als Vektoren, ist eine konvexe Teilmenge des $\mathbb{R}^m$. Ebenso bildet die Menge der maximalen Flüsse eine konvexe Teilmenge.

5.2 Geometrische Algorithmen

Kleinster umschliessender Kreis
Zu einer gegebenen Menge P von n Punkten in der Ebene suchen wir einen Kreis C(P), so dass C(P) alle Punkte aus P enthält und der Radius von C(P) so klein wie möglich ist.

Lemma 3.25
Für jede (endliche) Punktemenge P im $\mathbb{R}^2$ gibt es einen eindeutigen kleinsten umschliessenden Kreis C(P).

Lemma 3.26
Für jede (endliche) Punktemenge P im $\mathbb{R}^2$ mit $|P| \geq 3$ gibt es eine Teilmenge $Q \subseteq P$, so dass $|Q| = 3$ und $C(Q) = C(P)$.

Unser Ziel ist ein randomisierter Algorithmus mit (erwarteter) Laufzeit O(n ln(n)).
In einer primitiven Version des Algorithmus wählen wir uniform gleichverteilt immer eine 3 Punkte Menge Q und bestimmen C(Q). Sind alle Punkte enthalten, so geben wir C(Q) zurück, ansonsten wiederholen wir den Prozess. Dieser Algorithmus hat eine Laufzeit von $O(n^4)$
Eine erste Idee den Algorithmus zu beschleunigen ist es, mehr als 3 Punkte (z.B. 12) zu ziehen, da wir dann C(Q) immer noch in konstanter Zeit berechnen können aber eine höhere Chance haben, dass die Menge Q die drei definierenden Punkte von C(P) enthält. Dies ändert allerdings noch nichts an der Laufzeit.
Eine zusätzliche Idee verringert die Laufzeit: wir verdoppeln in jeder Iteration die Punkte ausserhalb von C(Q).
RandomizedCleverVersion(P):

```
1 repeat forever
2   wähle Q ⊆ P mit |Q| = 12 zufällig und gleichvereilt
3   bestimme C(Q)
4   if P ⊆ C(Q) then
5    return C(Q)
6   verdopple alle Punkte von P ausserhalb von C(Q)
```

Lemma 3.27 (Zufälliges Wählen von Punkten mit Gewichtung)
Seien $n_1, ..., n_t$ natürliche Zahlen und $N := \sum_{i=1}^{t} n_i$. Wir erzeugen $X \in \{1, ..., t\}$ zufällig wie folgt:
$k \leftarrow UNIFORMINT(1, N)$
$x \leftarrow 1$
while $\sum_{i=1}^{x} n_i < k$ do
$x \leftarrow x + 1$
return x

Dann gilt $Pr[X = i] = n_i/N$ für alle i = 1,...,t.

Lemma 3.28
Sei P eine Menge von n (nicht unbedingt verschiedenen) Punkten und $r \in \mathbb{N}$, R zufällig gleichverteilt aus $\binom{P}{r}$. Dann ist die erwartete Anzahl Punkte von P, die ausserhalb von C(R) liegen, höchstens $3\frac{n-r}{r+1}$.

Satz 3.29
Der Algorithmus RandomizedCleverVersion berechnet den kleinsten umschliessenden Kreis von P in erwarteter Zeit O(n log(n)).

Beweise / Zusammenhänge zu den Lemmas Skript S. 182f.

Das Sampling Lemma
Definition 3.30
Gegeben sei eine endliche Menge S, $n := |S|$, und Φ eine beliebige Funktion auf 2^s in einem beliebigen Wertebereich.
Wir definieren:
$V(R) = V_\Phi(R) := \{s \in S | \Phi(R \cup \{s\}) \neq \Phi(R)\}$
$X(R) = X_\Phi(R) := \{s \in R | \Phi(R \setminus \{s\}) \neq \Phi(R)\}$
Elemente in V(R) nennen wir Verletzter von R, Elemente in X(R) nennen wir extrem in R.
Unser Ziel ist zu verstehen wie gross V(R) ist, wenn R zufällig mit gegebener Grösse r gewählt wird.

Das Sampling Lemma setzt die erwartete Anzahl verletzender Elemente in Bezug zur Anzahl extremer Elemente.
Lemma 3.31 (Sampling Lemma)
Sei $k \in \mathbb{N}, 0 \leq k \leq n$. Wir setzen $v_k := \mathbb{E}[|V(R)|]$ und $x_k := \mathbb{E}[|X(R)|]$, wobei R eine k-elementige Teilmenge von S ist, zufällig gleichverteilt aus $\binom{S}{k}$. Dann gilt für $r \in \mathbb{N}$, $0 \leq r < n$,
$\frac{v_r}{n-r} = \frac{x_{r+1}}{r+1}$.

Korollar 3.32
Wählen wir r Elemente R aus einer Menge A von n Zahlen zufällig, dann ist der erwartete Rang des Minimums von R in A genau $\frac{n-r}{r+1} + 1 = \frac{n+1}{r+1}$.
Wählen wir r Punkte R aus einer Menge P von n Punkten in der Ebene zufällig, dann ist die erwartete Anzahl von Punkten aus P ausserhalb von C(R) höchstens $3\frac{n-r}{r+1}$.

Konvexe Hülle
Definition:
Sei $S \subseteq \mathbb{R}^d, d \in \mathbb{N}$. Die konvexe Hülle, conv(S), von S ist der Schnitt aller konvexen Mengen, die S enthalten, d.h.
$conv(S) := \bigcup_{S \subseteq C \subseteq \mathbb{R}^d, C konvex} C.$
Die konvexe Hülle selbst ist auch wieder konvex, sie ist die kleinste konvexe Menge, die S enthält.

Lemma 3.34
$(q_0, q_1, ..., q_{h-1})$ ist die Eckenfolge des conv(P) umeschliessenden Polygons gegen den Urzeigersinn, genau dann wenn alle Paare (q_{i-1}, q_i), i = 1,2,...,h, Randkanten von P sind.

Lemma 3.35
Seien $p = (p_x, p_y), q = (q_x, q_y)$, und $r = (r_x, r_y)$ Punkte in $\mathbb{R}^2$. Es gilt $q \neq r$ und p liegt links von qr, genau dann wenn:
$(q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)$. (Herleitung mittels Determinaten)

Wir beschreiben nun den JarvisWrap Alg. zum Finden der konvexen Hülle. Sei q_0 der Punkt mit kleinster x-Koordinate in P. Dann ist dieser sicher Teil der konvexen Hülle. Die folgende Funktion FindNext(q) findet den Punkt q_1, mit dem q_0 eine Randkante bildet.

Algorithm 13: FindNext(q)

```
1 wähle p_0 ∈ P \ {q} beliebig
2 q_next ← p_0
3 for all p ∈ P \ {q, p_0} do
4   if p rechts von qq_next then q_next ← p
5 return q_next
```

Warum funktioniert der Alg. ?
Lemma 3.36
Falls q eine Ecke der konvexen Hülle von P ist, so ist die Relation $\prec_q$ eine totale Ordnung auf $P \setminus \{q\}$. Für das Minimum p_{min} dieser Ordnung gilt, dass qp_{min} eine Randkante ist.

Satz 3.37
Gegeben eine Menge P von n Punkten in allgemeiner Lage (nicht aufeinander, nicht auf einer Gerade) in $\mathbb{R}^2$, berechnet der JarvisWrap die konvexe Hülle in Ziet O(nh), wobei h die Anzahl Ecken der Konvexen Hülle von P hat.

Algorithm 14: JarvisWrap(P)

```
1 h ← 0
2 p_now ← Punkt in P mit kleinster x-Koordinate
3 repeat
4   q_h ← p_now
5   p_now ← FindNext(q_h)
6   h ← h + 1
7 until p_now = q_0
8 return (q_0, q_1, ...q_{h-1})
```

Lokales Verbessern

Wir beginnen mit einem Polygon mit Ecken aus P und wollen dieses sukzessive korrigieren, wann immer wir ein Problem sehen.

Warnung: Ist in der Folge kein solcher Defekt zu entdecken, heisst das noch lange nicht, dass wir das richtige Polygon gefunden haben. Offensichtlich wissen wir nicht, dass das Polygon die Menge P umschliesst. Deshalb verwenden wir die Idee folgendermassen:

Als erstes sortieren wir P, aufsteigend nach x-Koordinate. Sei $(p_1, p_2, ..., p_n)$ die sich ergebende Reihenfolge.

Betrachten wir das Polygon $(p_1, ..., p_n, p_{n-1}, ..., p_2)$, d.h. wir durchlaufen P einmal von links nach rechts und dann wieder zurück von rechts nach links. Das Polygon hat 2(n-1) gerichtete Kanten. Nun beginnnen wir mit dem lokalen Verbessern dieses Polygons. Dabei wird p_1 wie auch p_n sicher nie entfernt und zu jedem Zeitpunkt teilt sich die Folge in einen Teil der x-mononton von p_1 nach p_n läuft und einen zweiten Teil der zurück läuft.

Ein Verbesserungsschritt sei definiert als die Operation, bei der wir für drei aufeinanderfolgende Punkte p, p', p" feststellen, dass p' links von pp" liegt und wir deshalb p' aus der Folge entfernen. Dabei vergrössert sich das vom Polygonzug umrandete Gebiet genau um das Dreieck pp'p". Behalten wir alte Kanten in der Zeichnung, so bekommen wir am Ende eine Triangulierung der Punktemenge P.

Ebene Graphen und Triangulierungen

Definition:

Ein Graph G=(P,E) auf P heisst eben, wenn sich die Segmente pq := conv({p,q}) der Kanten {p,q} $\in$ E höchstens in ihren jeweils gemeinsamen Endpunkten schneiden.

Entfernen wir die Segmente pq, {p,q} $\in$ E, aus $\mathbb{R}^2$ so nennen wir die entstehenden zusammenhängenden Gebiete die Gebiete von G. Die beschränkten Gebiete nennen wir innere Gebiete, das unbeschränkte (unendliche) Gebiet das äussere Gebiet.

Ein Graph T=(P,E) auf P heisst Triangulierung von P, falls T eben ist und maximal mit dieser Eigenschaft ist, d.h. das Hinzufügen jeder Kante $\binom{P}{2} \setminus E$ zu T verletzt die Eigenschaft "eben".

Wir beobachten, dass die Gebiete eines ebenen Graphen G auf P bis auf genau eines beschränkt sind.

Lemma 3.39
Sie h die Anzahl der Ecken der konvexen Hülle von P. Der lokale Verbesserungsprozess macht genau 2n-2-h Verbesserungsschritte und erzeugt eine Triangulierung mit 2n-2-h inneren Dreiecken und 3n-3-h Kanten.

Lemma 3.40 (Euler Relation)
Sei G=(P,E) ein ebener Graph auf P mit $v := |P|$ Knoten, $e := |E|$ Kanten, f Gebieten (inkl. des äusseren Gebiets). und c Zusammenhangskomponenten. Dann gilt:
v - e + f = 1 + c.

Korollar 3.41
Sei P eine Menge von $n \geq 3$ Punkten in allgemeiner Lage in $\mathbb{R}^2$ und sei h die Anzahl der Kanten von conv(P).
(i) Jede Triangulierung T von P hat genau 3n3h Kanten und genau 2n2h innere Gebiete.
(ii) Jeder ebene Graph G auf P hat höchstens $3n - 3 - h \leq 3n - 6$ Kanten und höchstens 2n2h $\leq$ 2n5 innere Gebiete.

Definition: planar
Ein Graph heisst planar, wenn man ihn in der Ebene so zeichnen kann, dass sich keine zwei Kanten kreuzen.

Satz (Einbettung planarer Graphen)
Sei G=(V,E) ein planarer Graph. Dann gibt es immer eine Einbettung $P : V \to \mathbb{R}^2$, so dass G=(P(V),E) ein ebener Graph ist.

Korollar
Für jeden planaren Graphen G=(V,E) gilt:
$|E| \leq 3|V| - 7$

Konvexe Hülle mit Local Repair
Frage: Wie finden wir die Ausbesserungen effizient?
Idee: Konstruiere "untere" Hülle von links nach rechts (sortierte Punkte), dann "obere" Hülle von rechts nach links.

Satz 3.42
Gegeben eine Folge $p_1, ..., p_n$ nach x-Koordinate sortierter Punkte in allgmeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair die konvexe Hülle von $\{p_1, ..., p_n\}$ in Zeit O(n)
D.h. inkl. Sortieren haben wir einen O(nlog(n)) Algorithmus, der schneller als JarvisWrap ist, es sei denn h = o(log(n)).

Algorithm 15: LocalRepair$(p_1, ..., p_n)$

```
1  setzt (p_1, ..., p_n), n ≥ 3, nach x-Koordinate sortiert voraus
2  q_0 ← p_1
3  h ← 0
4  for i ← 2 to n do // links nach rechts
5    while h > 0 und q_h links von q_{h-1}p_i do
6      h ← h − 1
7    h ← h + 1
8    q_h ← p_i // untere Hülle
9  h' ← h
10 for i ← n − 1 to 1 do // rechts nach links
11   while h > h' und q_h links von q_{h-1}p_i do
12     h ← h − 1
13   h ← h + 1
14   q_h ← p_i // untere Hülle
15 return (q_0, ..., q_{h-1}) // Ecken der konv. Hülle, gg. Uhrzeigersinn.
```

Das Problem der konvexen Hülle kann nicht schneller als Sortieren gelöst werden. Kann man konvexe Hüllen in t(n) berechnen, so kann man in t(n) + O(n) Zeit sortieren.