AuW Exam Notes

Ruben Schenk

August 2020

1 Wahrscheinlichkeitstheorie

1.1 Grundbegriffe und Notationen

Definition
Ein **diskreter Wahrscheinlichkeitsraum** ist bestimmt durch *Ergebnismenge* $\Omega = \{\omega_1, \omega_2, ...\}$ von *Elementarereignissen*. Jedem Elementarereigniss ω_i ist eine **Wahrscheinlichkeit** $\Pr[\omega_i]$ zugeordnet, wobei wir fordern, dass

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

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 $\overline{E} := \Omega \setminus E$ das **Komplementärereignis** zu E.

Lemma: Für Ereignisse $\mathcal{A}$, $\mathcal{B}$ gilt:

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

Additionssatz
Wenn die Ereignisse $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n$ *paarweise disjunkt* sind, so gilt

$$\Pr\Big[\bigcup_{i=1}^{n} \mathcal{A}_i\Big] = \sum_{i=1}^{n} \Pr[\mathcal{A}_i]$$

Siebformel
Für Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ gilt:

$$\Pr\Big[\bigcup_{i=1}^{n} \mathcal{A}_i\Big] = \sum_{l=1}^{n} (-1)^{l+1} \sum_{1 \leq i_1 \leq ... \leq i_l \leq n} \Pr[\mathcal{A}_{i_1} \cap ... \cap \mathcal{A}_{i_l}].$$

Korollar: Union Bound
Für Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ gilt

$$\Pr\Big[\bigcup_{i=1}^{n}\Big] \leq \sum_{i=1}^{n} \Pr[\mathcal{A}_i]$$

Prinzip von Laplace: Wenn nichts dagegen spricht, gehen wir davon aus, dass alle Elementarerignisse gleich wahrscheinlich sind.

Bei der Anwendung des Prinzips von Laplace gilt für alle Elementarerignisse $\Pr[\omega] = \frac{1}{|\Omega|}$. Daraus erhalten wir für ein beliebiges Ereignis E die Formel

$$\Pr[E] = \frac{|E|}{|\Omega|}$$

1.2 Bedingte Wahrscheinlichkeit

Mit $\mathcal{A}|\mathcal{B}$ bezeichnen wir das Ereignis, dass $\mathcal{A}$ eintritt, wenn wir bereits wissen, dass das Ereignis $\mathcal{B}$ auf jeden Fall eintritt.

Definition
Seien $\mathcal{A}$ und $\mathcal{B}$ Ereignisse mit $\Pr[\mathcal{B}] > 0$. Die **bedingte Wahrscheinlichkeit** $\Pr[\mathcal{A}|\mathcal{B}]$ ist definiert durch

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

Multiplikationssatz
Seien die Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ gegeben. Falls $\Pr[\mathcal{A}_1 \cap ... \cap \mathcal{A}_n] > 0$ ist, gilt

$$\Pr[\mathcal{A}_1 \cap ... \cap \mathcal{A}_n] = \Pr[\mathcal{A}_1] \cdot \Pr[\mathcal{A}_2|\mathcal{A}_1] \cdot \Pr[\mathcal{A}_3|\mathcal{A}_1 \cap \mathcal{A}_2] \cdot \cdots \Pr[\mathcal{A}_n|\mathcal{A}_1 \cap ... \cap \mathcal{A}_{n-1}]$$

Satz von Bayes
Seien die Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ paarweise disjunkt. Ferner sei $\mathcal{B} \subseteq \mathcal{A}_1 \cup ... \cup \mathcal{A}_n$ ein Ereignis mit $\Pr[\mathcal{B}] > 0$. Dann gilt für ein beliebiges $i = 1, ..., n$

$$\Pr[\mathcal{A}_i|\mathcal{B}] = \frac{\Pr[\mathcal{A}_i \cap \mathcal{B}]}{\Pr[\mathcal{B}]} = \frac{\Pr[\mathcal{B}|\mathcal{A}_i] \cdot \Pr[\mathcal{A}_i]}{\sum_{j=1}^{n} \Pr[\mathcal{B}|\mathcal{A}_|] \cdot \Pr[\mathcal{A}_j]}.$$

1.3 Unabhängigkeit

Definition
Die Ereignisse $\mathcal{A}$ und $\mathcal{B}$ heissen **unabhängig**, wenn gilt

$$\Pr[\mathcal{A} \cap \mathcal{B}] = \Pr[\mathcal{A}] \cdot \Pr[\mathcal{B}].$$

Definition: Die Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ heissen *unabhängig*, wenn für alle Teilmengen $I \subseteq \{1, ..., n\}$ mit $I = \{i_1, ..., i_k\}$ gilt, dass

$$\Pr[\mathcal{A}_{i_1} \cap ... \cap \mathcal{A}_{i_k}] = \Pr[\mathcal{A}_{i_1}] \cdot \cdots \Pr[\mathcal{A}_{i_k}].$$

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

1.4 Zufallsvariablen

Definition Eine **Zufallsvariable** ist eine Abbildung $\mathcal{X} : \Omega \to \mathbb{R}$, wobei Ω die Ergebnismenge eines Wahrscheinlichkeitsraumes ist.

Definition: Der *Wertebereich* einer Zufallsvariable ist definiert durch

$$\mathcal{W}_\mathcal{X} := \mathcal{X}(\Omega) = \{x \in \mathbb{R} \mid \exists \omega \in \Omega \;\; mit \;\; \mathcal{X}(\omega) = x\}.$$

Definition
Die *Dichtefunktion* ist definiert durch

$$f_\mathcal{X} : \mathbb{R} \to [0,1], \quad x \to \Pr[\mathcal{X} = x].$$

Die *Verteilungsfunktion* ist definiert durch

$$F_\mathcal{X} : \mathbb{R} \to [0,1], \quad x \to \Pr[\mathcal{X} \leq x] = \sum_{x' \in \mathcal{W}_\mathcal{X} : x' \leq x} \Pr[\mathcal{X} = x].$$

1.4.1 Erwartungswert

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

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

sofern die Summe absolut konvergiert. Ansonsten sagen wir, dass der Erwartungswert undefiniert ist.

Lemma: Ist $\mathcal{X}$ eine Zufallsvariable, so gilt:

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

Satz: Sei $\mathcal{X}$ eine Zufallsvariable. Für paarweise disjunkte Ereignisse $\mathcal{A}_1, ..., \mathcal{A}_n$ mit $\mathcal{A}_1 \cup ... \cup \mathcal{A}_n = \Omega$ und $\Pr[\mathcal{A}_1], ..., \Pr[\mathcal{A}_n] > 0$ gilt

$$\mathbb{E}[\mathcal{X}] = \sum_{i=1}^{n} \mathbb{E}[\mathcal{X}|\mathcal{A}_i] \cdot \Pr[\mathcal{A}_i].$$

1.4.2 Linearität des Erwartungswerts

Der Erwartungswert einer Summe von Zufallsvariablen ist die Summe der Erwartungswerte der Zufallsvariablen.

Linearität des Erwartungswerts
Für Zufallsvariablen $\mathcal{X}_1, ..., \mathcal{X}_n$ und $\mathcal{X} := a_1\mathcal{X}_1 + ... + a_n\mathcal{X}_n + b$ mit $a_1, ..., a_n, b \in \mathbb{R}$ gilt

$$\mathbb{E}[\mathcal{X}] = a_1\mathbb{E}[\mathcal{X}_1] + ... + a_n\mathbb{E}[\mathcal{X}_n] + b.$$

Definition
Für ein Ereignis $\mathcal{A} \subseteq \Omega$ ist die zugehörige **Indikatorvariable** $\mathcal{X}_\mathcal{A}$ definiert durch:

$$\mathcal{X}_\mathcal{A}(\omega) := \begin{cases} 1, & falls\ \omega \in \mathcal{A} \\ 0, & sonst. \end{cases}$$

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

1.4.3 Varianz

Definition: Für eine Zufallsvariable $\mathcal{X}$ mit $\mu = \mathbb{E}[\mathcal{X}]$ definieren wir die **Varianz** durch

$$\mathrm{Var}[\mathcal{X}] := \mathbb{E}[(\mathcal{X} - \mu)^2] = \sum_{x \in \mathbb{W}_\mathcal{X}} (x - \mu)^2 \cdot \Pr[\mathcal{X} = x].$$

Die Grösse $\sigma := \sqrt{\mathrm{Var}[\mathcal{X}]}$ heisst **Standardabweichung** von $\mathcal{X}$.

Satz: Für eine beliebige Zufallsvariable $\mathcal{X}$ gilt

$$\mathrm{Var}[\mathcal{X}] = \mathbb{E}[\mathcal{X}^2] - \mathbb{E}[\mathcal{X}]^2.$$

Satz: Für eine beliebige Zufallsvariable $\mathcal{X}$ und $a, b \in \mathbb{R}$ gilt

$$\mathrm{Var}[a \cdot \mathcal{X} + b] = a^2 \cdot \mathrm{Var}[\mathcal{X}].$$

1.5 Wichtige diskrete Verteilungen

1.5.1 Bernoulli Verteilung

Eine Zufallsvariable $\mathcal{X}$ mit $\mathcal{W}_\mathcal{X} = \{0, 1\}$ und der Dichte

$$f_\mathcal{X}(x) = \begin{cases} p & \text{für } x = 1, \\ 1 - p & \text{für } x = 0, \\ 0 & \text{sonst} \end{cases}$$

heisst **Bernoulli-verteilt**. Den Parameter p nennt man die *Erfolgswahrscheinlichkeit* der Bernoulli-Verteilung. Ist eine Zufallsvariable Bernoulli-verteilt mit Parameter p, so schreibt man dies auch als

$$\mathcal{X} \sim \mathrm{Bernoulli}(p).$$

Für eine solche Bernoulli-verteilte Zufallsvariable $\mathcal{X}$ gilt

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

1.5.2 Binomialverteilt

Werfen wir eine Münze n mal und fragen wie oft wir Kopf erhalten haben, so ist die entsprechende Zufallsvariable $\mathcal{X}$ **binomialverteilt**. Für die Dichte von $\mathcal{X}$ gilt zudem:

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

Eine Zufallsvariable $\mathcal{X}$ mit $\mathcal{W}_{\mathcal{X}} = \{0,1,...,n\}$ und der Dichte wie oben beschrieben heisst **binomialverteilt** mit Parameter n und p. Man schreibt dies auch als

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

Für eine solche binomialverteilte Zufallsvariable $\mathcal{X}$ gilt

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

1.5.3 Geometrische Verteilung

Wenn ein einzelner Versuch mit Wahrscheinlichkeit p gelingt, so ist die Anzahl der Versuche bis zum Erfolg **geometrisch verteilt**. Wir schreiben hierfür auch

$$\mathcal{X} \sim \text{Geo}(p).$$

Für die Dichte einer geometrisch verteilten Zufallsvariable gilt

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

Erwartungswert und Varianz sind

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

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

$$\Pr[\mathcal{X} \geq s+t | \mathcal{X} > s] = \Pr[\mathcal{X} \geq t].$$

1.5.4 Poisson-Verrteilung

Die **Poisson-Verteilung** beschreibt den Grenzwert einer Binomialverteilung und modelliert unwahrscheinliche Ereignisse. Für eine Poisson-Verteilung schreiben wir:

$$\mathcal{X} \sim \text{Po}(\lambda).$$

Für die Dichte einer Poisson-Verteilung mit Parameter λ gilt

$$f_{\mathcal{X}}(i) = \begin{cases} \frac{e^{-\lambda}\lambda^i}{i!} & \text{für } i \in \mathbb{N}_0 \\ 0 & \text{sonst.} \end{cases}$$

Für den Erwartungswert und die Varianz gilt

$$\mathbb{E}[\mathcal{X}] = \text{Var}[\mathcal{X}] = \lambda.$$

Bemerkung: Gilt für eine Binomialverteilung $\text{Bin}(n, \frac{\lambda}{n})$, dass der erste Parameter n sehr gross ist und das Produkt der beiden Parameter np eine kleine Konstante ist, so kann man die Binomialverteilung durch die Poisson-Verteilung approximieren, wobei der Parameter der Poisson-Verteilung durch den Erwartungswert der Binomialverteilung gegeben ist.

1.5.5 Coupon-Collector-Problem

Gelegentlich werden Produkten Coupons beigelegt, um den Käufer zum Sammeln anzuregen. Wenn es insgesamt n verschiedene Coupons gibt, wie viele Produkte muss man im Mittel erwerben, bis man eine vollständige Sammlung (jeden der n Coupons mindestens einmal) besitzt?

Sei $\mathcal{X}$ die Anzahl der Käufe bis zur Komplettierung der Sammlung. Zudem teilen wir die Zeit in Phasen ein: Phase i bezeichnet die Schritte/Käufe vom Erwerb des $(i-1)$-ten Coupons (exklusiv) bis zum Erwerb des i-ten Coupons (inklusiv). Wäre zum Beispiel $n = 4$ und unsere Käufe sehen wie folgt aus: $2,2,1,2,2,3,1,3,2,3,1,4$. Dann wären die vier Phasen wie folgt: $P_1 = \{2\},\ P_2 = \{2,1\},\ P_3 = \{2,2,3\}$ und $P_4 = \{1,3,2,3,1,4\}$.

Sei nun $\mathcal{X}_i$ die Anzahl der Käufe in Phase i. $\mathcal{X}_i$ ist geometrisch verteilt mit Parameter $p = \frac{n-i+1}{n}$ und es gilt $\mathbb{E}[\mathcal{X}_i] = \frac{n}{n-i+1}$. Mit diesem Wissen können wir $\mathbb{E}[\mathcal{X}]$ ausrechnen, denn es folgt wegen der Linearität des Erwartungswerts, dass

$$\mathbb{E}[\mathcal{X}] = \sum_{i=1}^{n} \mathbb{E}[\mathcal{X}_i] = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \cdot \sum_{i=1}^{n} \frac{1}{i}.$$

Es folgt, dass $\mathbb{E}[\mathcal{X}] = n\ln(n) + \text{O}(n)$ ist.

1.6 Mehrere Zufallsvariablen

Wir untersuchen für zwei Zufallsvariablen $\mathcal{X}$ und $\mathcal{Y}$ Wahrscheinlichkeiten der Art

$$\Pr[\mathcal{X} = x,\ \mathcal{Y} = y] = \Pr[\{\omega \in \Omega \mid \mathcal{X}(\omega) = x,\ \mathcal{Y}(\omega) = y\}].$$

Die **gemeinsame Dichte** der Zufallsvariablen $\mathcal{X}$ und $\mathcal{Y}$ ist gegeben durch

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

Die **Randdichten**, also die Dichten der einzelnen Zufallsvariablen, sind gegeben durch

$$f_{\mathcal{X}}(x) = \sum_{y \in \mathcal{W}_{\mathcal{Y}}} f_{\mathcal{X},\mathcal{Y}}(x,y) \text{ bzw. } f_{\mathcal{Y}}(y) = \sum_{x \in \mathcal{W}_{\mathcal{X}}} f_{\mathcal{X},\mathcal{Y}}(x,y).$$

Bemerkung: Es gilt $\Pr[\mathcal{X} = x] = f_{\mathcal{X}}(x)$ und $\Pr[\mathcal{Y} = y] = f_{\mathcal{Y}}(y)$.

Die **gemeinsame Verteilung** ist gegeben als

$$F_{\mathcal{X},\mathcal{Y}}(x,y) := \Pr[\mathcal{X} \leq x, \mathcal{Y} \leq y] = \Pr[\{\omega \in \Omega \mid \mathcal{X}(\omega) \leq x,\ \mathcal{Y}(\omega) \leq y\}] = \sum_{x' \leq x} \sum_{y' \leq y} f_{\mathcal{X},\mathcal{Y}}(x',y').$$

Auch hier können wir wieder die **Randverteilung** beachten, die gegeben ist durch

$$F_{\mathcal{X}}(x) = \sum_{x' \leq x} f_{\mathcal{X}}(x') = \sum_{x' \leq x} \sum_{y \in \mathcal{W}_{\mathcal{Y}}} f_{\mathcal{X},\mathcal{Y}}(x', y)$$

und analog dazu die Randverteilung von $\mathcal{Y}$.

1.6.1 Unabhängigkeit von Zufallsvariablen

Definition: Zufallsvariablen $\mathcal{X}_1, ..., \mathcal{X}_n$ heissen *unabhängig*, genau dann wenn für alle $(x_1, ..., x_n) \in \mathcal{W}_{\mathcal{X}_1} \times ... \times \mathcal{W}_{\mathcal{X}_n}$ gilt

$$\Pr[\mathcal{X}_1 = x_1, ..., \mathcal{X}_n = x_n] = \Pr[\mathcal{X}_1 = x_1] \cdot \cdots \Pr[\mathcal{X}_n = x_n].$$

Satz: Seien $f_1, ...f_n$ reellwertige Funktionen ($f_i : \mathbb{R} \to \mathbb{R}$ für $i = 1, ..., n$). Wenn die Zufallsvariablen $\mathcal{X}_1, ..., \mathcal{X}_n$ unabhängig sind, dann gilt dies auch für $f_1(\mathcal{X}_1), ..., f_n(\mathcal{X}_n)$.

1.6.2 Zusammengesetzte Zufallsvariablen

Satz: Für zwei unabhängige Zufallsvariablen $\mathcal{X}$ und $\mathcal{Y}$ sei $\mathcal{Z} := \mathcal{X} + \mathcal{Y}$. Es gilt

$$f_{\mathcal{Z}}(z) = \sum_{x \in \mathcal{W}_{\mathcal{X}}} f_{\mathcal{X}}(x) \cdot f_{\mathcal{Y}}(z - y).$$

Multiplikativität des Erwartungswerts
Für unabhängige Zufallsvariablen $\mathcal{X}_1, ..., \mathcal{X}_n$ gilt

$$\mathbb{E}[X_1 \cdots \mathcal{X}_n] = \mathbb{E}[\mathcal{X}_1] \cdots \mathbb{E}[\mathcal{X}_n].$$

Satz: Für unabhängige Zufallsvariablen $\mathcal{X}_1, ..., \mathcal{X}_n$ und $\mathcal{X} := \mathcal{X}_1 + ... + \mathcal{X}_n$ gilt

$$\mathrm{Var}[\mathcal{X}] = \mathrm{Var}[\mathcal{X}_1] + ... + \mathrm{Var}[\mathcal{X}_n].$$

1.6.3 Waldsche Identität

Waldsche Identität
Seien $\mathcal{N}$ und $\mathcal{X}$ zwei unabhängige Zufallsvariablen, wobei für den Wertebreich von $\mathcal{N}$ gilt: $\mathcal{W}_{\mathcal{N}} \subseteq \mathbb{N}$. Weiter sei

$$\mathcal{Z} := \sum_{i=1}^{\mathcal{N}} \mathcal{X}_i,$$

wobei $\mathcal{X}_1, \mathcal{X}_2, ...$ unabhängige Kopien von $\mathcal{X}$ seien. Dann gilt:

$$\mathbb{E}[\mathcal{Z}] = \mathbb{E}[\mathcal{N}] \cdot \mathbb{E}[\mathcal{X}].$$

1.7 Abschätzen von Wahrscheinlichkeiten

1.7.1 Die Ungleichungen von Markov und Chebyshev

Ungleichung von Markov
Sei $\mathcal{X}$ eine Zufallsvariable, die nur nicht-negative Werte annimmt. Dann gilt für alle $t \in \mathbb{R}$ mit $t > 0$, dass

$$\Pr[\mathcal{X} \geq t] \leq \frac{\mathbb{E}[\mathcal{X}]}{t}.$$

Oder äquivalent dazu $\Pr[\mathcal{X} \geq t \cdot \mathbb{E}[\mathcal{X}]] \leq t^{-1}$.

Ungleichung von Chebyshev
Sei $\mathcal{X}$ eine Zufallsvariable mit $t \in \mathbb{R}$ mit $t > 0$. Dann gilt

$$\Pr[|\mathcal{X} - \mathbb{E}[\mathcal{X}]| \geq t] \leq \frac{\mathrm{Var}[\mathcal{X}]}{t^2}.$$

Oder äquivalent dazu $\Pr\left[|\mathcal{X} - \mathbb{E}[\mathcal{X}]| \geq t \cdot \sqrt{\mathrm{Var}[\mathcal{X}]}\right] \leq t^{-2}$.

1.7.2 Die Ungleichung von Chernoff

Chernoff-Schranken: Seien $\mathcal{X}_1, ..., \mathcal{X}_n$ unabhängige *Bernoulli-verteilte* Zufallsvariablen mit $\Pr[\mathcal{X}_i = 1] = p_i$ und $\Pr[\mathcal{X}_i = 0] = 1 - p_i$.
Dann gilt für $\mathcal{X} := \sum_{i=1}^{n} \mathcal{X}_i$:

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

1.8 Randomisierte Algorithmen

Las-Vegas Algorithmus
Las-Vegas Algorithmen werden charakterisiert durch:

- Geben nie eine falsche Antwort
- Geben zuweilen keine Antwort (Ausgabe = "???")
- Haben eine beschränkte Laufzeit, falls wir die Ausgabe "???" akzeptieren
- Haben eine unbeschränkte Laufzeit (Zufallsvariable), falls wir nur richtige Antworten akzeptieren
- **Ziel:** $\Pr[\text{Ausgabe} = \text{"???"}] = \text{"winzig"}$

Monte-Carlo Algorithmus Monte-Carlo Algorithmen werden charakterisiert durch:

- Geben zuweilen eine falsche Antwort
- Haben eine beschränkte (und meist schnelle) Laufzeit
- **Ziel:** Pr[Ausgabe falsch] = "winzig"

1.8.1 Reduktion der Fehlerwahrscheinlichkeit

Satz: Sein $\mathcal{A}$ ein **Las-Vegas Algorithmus** mit $\Pr[\mathcal{A}(I) \text{ korrekt}] \geq \epsilon$. Wir definieren für $\delta > 0$ den Algorithmus $\mathcal{A}_\delta$, der den Algorithmus $\mathcal{A}$ N mal aufruft, wie folgt:

- Gibt eine Antwort aus, sobald $\mathcal{A}$ ein Antwort ausgibt, die nicht "???" ist
- Gibt "???" aus, falls $\mathcal{A}$ N mal "???" ausgegeben hat

Für den Algorithmus $\mathcal{A}_\delta$ gilt, dass

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq (1 - \delta)$$

falls

$$N = \epsilon^{-1} \ln(\delta^- 1).$$

Satz: Sei $\mathcal{A}$ ein **Monte-Carlo Algorithmus** mit **einseitigem Fehler**, dass heisst, $\mathcal{A}$ gibt immer eine der beiden Antworten JA oder $NEIN$ aus, wobei

$$\Pr[\mathcal{A}(I) = JA] = 1 \quad \text{falls } I \text{ eine } JA - \text{Instanz ist,}$$

und

$$\Pr[\mathcal{A}(I) = NEIN] \geq \epsilon \quad \text{falls } I \text{ eine } NEIN - \text{Instanz ist.}$$

Wir definieren für $\delta > 0$ den Algorithmus $\mathcal{A}_\delta$, der den Algorithmus $\mathcal{A}$ N mal aufruft, wie folgt:

- Gibt $NEIN$ aus, sobald $\mathcal{A}$ einmal $NEIN$ ausgegeben hat
- Gibt JA aus, wenn $\mathcal{A}$ N mal JA ausgegeben hat

Für den Algorithmus $\mathcal{A}_\delta$ gilt, dass

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq (1 - \delta)$$

falls

$$N = e^{-1} \ln(\delta^{-1})$$

Satz: Sei $\mathcal{A}$ ein **Monte-Carlo Algorithmus** mit **zweiseitigem Fehler**, das heisst, $\mathcal{A}$ gibt immer eine der beiden Antworten JA oder $NEIN$ aus, wobei

$$\Pr[\mathcal{A}(I) \text{ korrekt}] \geq \frac{1}{2} + \epsilon.$$

Wir definieren für $\delta > 0$ den Algorithmus $\mathcal{A}_\delta$, der den Algorithmus $\mathcal{A}$ N mal aufruft, und danach die am *häufigst aufgetretene* Antwort ausgibt. Dann gilt:

$$\Pr[\mathcal{A}_\delta(I) \text{ korrekt}] \geq (1 - \delta)$$

falls

$$N = 4\epsilon^{-2} \ln(\delta^{-1}).$$

1.8.2 Sortieren und Selektieren

Ein klassisches Beispiel für einen Las-Vegas Algorithmus ist der *QuickSort* Algorithmus, bei dem wir in jedem Schritt das Pivot-Element zufällig wählen.

Algorithm 1 QuickSort

```
1: function QuickSort(A, l, r)
2:     if l < r then
3:         p ← Uniform({l, l + 1, ...r})          ▷ wähle Pivotelement zufällig
4:         t ← Partition(A, l, r, p)
5:         QuickSort(A, l, t − 1)
6:         QuickSort(A, t + 1, r)
```

Es gilt $\mathbb{E}[T_{1,n}] \leq 2(n+1)\ln(n) + \mathrm{O}(n) \in \mathrm{O}(n \ln(n))$, wobei $T_{l,r}$ die Anzahl der Vergleiche bezeichnet, die bei einem Aufruf von $QuickSort(A, l, r)$ ausgeführt werden.

Algorithm 2 QuickSelect

```
1: function QuickSelect(A, l, r, k)
2:     p ← Uniform({l, l + 1, ...r})              ▷ wähle Pivotelement zufällig
3:     t ← Partition(A, l, r, p)
4:     if k = t then
5:         return A[t]                            ▷ gesuchtes Element ist gefunden
6:     else if k < t then
7:         return QuickSelect(A, l, t − 1, k)     ▷ gesuchtes Element ist links
8:     else
9:         return QuickSelect(A, t + 1, r, k − t) ▷ gesuchtes Element ist rechts
```

Sei T die Anzahl von Vergleichen, dann gilt $\mathbb{E}[T] \leq 2n \sum_{j=1}^{\infty} (\frac{3}{4})^{j-1} = 8n \in \mathrm{O}(n)$.

1.8.3 Primzahltest

Satz: Kleiner fermatscher Satz
Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $0 < a < n$

$$a^{n-1} \equiv 1 \mod n.$$

Algorithm 3 Miller-Rabin-Primzahltest(n)

```
1:  function MR-PrimeTest(n)
2:      if n = 2 then
3:          return 'Primzahl'
4:      else if n gerade oder n = 1 then
5:          return 'keine Primzahl'
6:      Wähle a ∈ {2, 3, ..., n − 1} zufällig
7:      Berechne k, d ∈ ℤ mit n − 1 = d2^k und d ungerade
8:      x ← a^d mod n
9:      if x = 1 oder x = n − 1 then
10:         return 'Primzahl'
11:     for i = 1, ..., k − 1 do 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'
```

Die Laufzeit dieses Algorithmus ist O(ln(n)). Ist n prim, so gibt der Algorithmus immer die Antwort 'Primzahl'. Ist n zusammengesetzt, so gibt er die Antwort 'keine Primzahl' mit Wahrscheinlichkeit mindestens $(\frac{3}{4})^3$.

1.8.4 Target-Shooting

Gegeben eine Menge U und eine Untermenge $S \subseteq U$ *unbekannter* Grösse. Wie gross ist der Quotient $\frac{|S|}{|U|}$? Wir nehmen an, dass wir für S die Indikatorfunktion $I_S : U \to \{0, 1\}$ haben, sodass $I_S(u) = 1$ genau dann gilt wenn $u \in S$.

Algorithm 4 Target-Shooting

```
1: Wähle U_1, ..., U_n ∈ U zufällig, gleichverteilt und unabhängig
2: return N^{-1} · Σ_{i=1}^{N} I_S(u_i)
```

Sei $\mathcal{Y}_i := I_S(U_i)$. Dann gilt für die Zufallsvariable $\mathcal{Y} := \frac{1}{N}\sum_{i=1}^{N} \mathcal{Y}_i$. Demnach erhalten wir für den Erwartungswert

$$\mathbb{E}[Y] = \frac{|S|}{|U|}.$$

Für die Varianz von $\mathcal{Y}$ gilt

$$\mathrm{Var}[Y] = \frac{1}{N}\Big(\frac{|S|}{|U|} - \Big(\frac{|S|}{|U|}\Big)^2\Big).$$

Satz: Seien $\delta, \epsilon > 0$. Falls $N > 3\frac{|U|}{|S|} \cdot \log(\frac{2}{\delta})$, so ist die Ausgabe des Algorithmus Target-Shooting(m)it Wahrscheinlichkeit mindestens $(1 - \delta)$ im Intervall $\Big[(1-\epsilon)\frac{|S|}{|U|}, (1+\epsilon)\frac{|S|}{|U|}\Big]$.

1.8.5 Finden von Duplikaten

Problemstellung: Gegeben ein Datensatz $S := \{s_1, ..., s_n\}$, finde Duplikate, d.h. finde alle Paare $1 \le i < j \le n$ mit $s_i = s_j$.

Definition: Gilt $S \subseteq U$, d.h. unser Datensatz ist eine Teilmenge eines sog. *Universums* U, so ist eine **Hashfunktion** eine Abbildung $h : U \to [m]$, wobei $[m] = \{1, ..., m\}$ und m die Anzahl der zusätzlich verfügbaren Speicherzellen ist.

Wir gehen davon aus, dass die Elemente *zufällig verteilt* sind, also für jedes Element $u \in U$ gilt: $\Pr[h(u) = i] = \frac{1}{m}$, für alle $i \in [m]$.

Wir können unser Problem einerseits mit einer *Hashmap* lösen. In dieser speichern wir die Elemente $(h(s_i), i)$ für alle $i \in \{1, ..., n\}$ und sortieren die Elemente dann bezüglich der ersten Koordinate. Anschliessen müssen wir nur noch diejenigen Datensätze auf Gleichheit überprüfen, die identische Einträge in der ersten Koordinate haben.

Wir definieren für einen Datensatz s_i die Indikatorvariable $\mathcal{X}_i$ die genau dann 1 ist, wenn es eine Kollision mit einem Datensatz s_j mit $s_j \neq s_i$ gibt. Es gilt:

$$\Pr[\mathcal{X}_i = 1] = 1 - (1 - \frac{1}{m})^{n-1}$$

und somit

$$\mathbb{E}[\text{Anzahl Kollisionen}] \le \text{Anzahl Duplikate } + n \cdot (1 - (1 - \frac{1}{m})^{n-1}).$$

Eine Alternative zu Hashmaps sind sog. **Bloomfilter**. Diese kommen oft zum Einsatz, wenn n sehr gross ist. Die grundlegende Idee ist hier, nicht nur eine, sondern k viele Hashfunktionen $h_1, ..., h_k : U \to [m]$ zu verwenden. Als Speicher M verwenden wir m **Bits**, die wir zu beginn alle auf 0 setzten. Zusätzlich initialisieren wir eine Liste $\mathcal{L}$ von möglichen Duplikaten mit $\mathcal{L} := \emptyset$. Für jeden Datensatz s_i gehen wir dann wie folgt vor:

1. für jedes $1 \le j \le k$ berechnen wir $x_i := h(s_i)$,
2. gilt $M[x_j] = 1$ für alle $1 \le j \le k$, so fügen wir i zu $\mathcal{L}$ hinzu, ansonsten setzten wir $M[x_j] = 1$ für alle $1 \le j \le k$.

Nachdem wir alle Datensätze s_i auf diese Weise behandelt haben, gehen wir nochmals alle Datensätze durch s_i durch und prüfen, ob $s_i = s_j$ für ein $j \in \mathcal{L}$, $j \neq i$, gilt.

2 Algorithmen - Highlights

2.1 Graphenaglorithmen

2.1.1 Lange Pfade

Problemstellung: Gegeben (G, B), wobei G ein Graph und $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt. Wir nennen das das **Long-Path-Problem**.
Dieses Problem ist relativ schwierig und es gilt: Können wir das Long-Path-Problem in polynomieller Zeit lösen, dann können wir auch das Hamiltonkreis-Problem in polynomieller Zeit lösen.

Wir beschäftigen uns aber im Moment nur mit Pfaden der Länge $B = \log(n)$, für welches eine polynomieller Algorithmus gefunden: **Color Coding.** Als erstes führen wir einige Hilfsmittel ein:

- $[n] := \{1, 2, ..., n\}$, $[n]^k$ ist die Menge der Folgen der Länge k über $[n]$.
- Für jeden Graph $G = (V, E)$ gilt $\sum_{v \in V} \deg(v) = 2|E|$
- Für $n \in \mathbb{N}_0$ gilt $\sum_{i=0}^{n} \binom{n}{i} = 2^n$

Wir untersuchen eine Variation des Problems: Dazu betrachten wir einen mit k Farben gefärbten Graphen, d.h. $G = (V, E)$ mit einer Abbildung $\gamma : V \to [k]$.

Definition: Ein Pfad ist **bunt**, wenn alle Knoten auf dem Pfad verschiedene Farben haben.

Wir beschreiben nun einen Algorithmus. der entscheidet, ob es in G einen bunten Pfad der Länge $k-1$ gibt (es müssen also alle k Farben im Pfad genau einmal vorkommen).
Wir definieren dazu für $v \in V$ und $i \in \mathbb{N}_0$ die Menge

$$P_i(v) := \left\{ S \in \binom{[k]}{i+1} \,\middle|\, \exists \text{ in } v \text{ endender, genau mit } S \text{ gefärbert bunter Pfad} \right\}.$$

$P_i(v)$ 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. Unser Problem reduziert sich also auf die Aussage

$$\exists \text{ bunter Pfad der Länge } k-1 \Leftrightarrow \bigcup_{v \in V} P_{k-1}(n) \neq \emptyset.$$

Wir definieren formal

Definition:

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(n) \notin R\}$$

Algorithm 5 Color Coding

```
1: function BUNT(G,i)
2:     for all v ∈ V do
3:         for all x ∈ N(v) do
4:             for all R ∈ P_{i-1}(x) mit γ(v) ∉ R do
5:                 P_i(v) ← P_i(u) ∪ {R ∪ {γ(v)}}
```

Wir können die i-te Runde (d.h. BUNT(G,i), wo wir die P_i's aus den P_{i-1}'s berechnen) in Zeit

$$O\Big(\sum_{v \in V} \deg(v) \cdot \binom{k}{i} \cdot i\Big) = O\Big(\binom{k}{i} \cdot i \cdot m\Big)$$

berechnen. Insgesamt, über alle $k-1$ Runden, ergibt das eine Laufzeit von

$$O\Big(\sum_{i=1}^{k-1} \binom{k}{i} \cdot i \cdot m\Big) = O(2^k km)$$

Satz: 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 $k-1$ erhält $\frac{1}{p_{Erfolg}} \leq e^k$.

Aus dem Algorithmus BUNT(G,i) können wir leicht einen Monte-Carlo Algorithmus machen, in dem wir den Algorithmus für $\lambda \in \mathbb{R},\ \lambda > 1$, $\lceil \lambda e^k \rceil$-mal laufen lassen. Sobald der Algorithmus einmal JA zurückgibt, geben wir dies zurück. Sollte der Algorithmus $\lceil \lambda e^k \rceil$-mal $NEIN$ zurückgeben, dann geben wir $NEIN$ zurück.

Satz:

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 $e^{-\lambda}$.

2.1.2 Flüsse in Netzwerken

Definition: Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt

1. (V, A) ist ein gerichteter Graph,
2. $s \in V$ ist die *Quelle* (source),
3. $t \in V \setminus \{s\}$ ist die *Senke* (sink/target),
4. $c : A \to \mathbb{R}_0^+$ ist die *Kapazitätsfunktion.*

Definition: Gegeben ein Netzwerk $N = (V, A, c, s, t)$. Ein **Fluss** in N ist eine Funktion $f : A \to \mathbb{R}$ mit den Bedingungen der

- *Zulässigkeit:* $0 \leq f(e) \leq c(e)$ für alle $e \in A$
- *Flusserhaltung:* $\sum_{u \in V:(u,v) \in A} f(u,v) = \sum_{u \in V:(v,u) \in A} f(v,u)$ für alle $v \in V \setminus \{s,t\}$

Der **Wert** eines Flusses f ist durch

$$\text{val}(f) := \text{netoutflow}(s) := \sum_{u \in V:(s,u) \in A} f(s,u) - \sum_{u \in V:(u,s) \in A} f(u,s)$$

definiert.

Lemma: Der **Nettozufluss** der Senke gleicht dem Wert des Flusses, d.h.

$$\text{netinflow}(t) := \sum_{u \in V:(u,t) \in A} f(u,t) - \sum_{u \in V:(t,u) \in A} f(t,u) = \text{val}(f).$$

Definition: Ein **s-t-Schnitt** für ein Netzwerk $N = (V, A, c, s, t)$ 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

$$\text{cap}(S,T) := \sum_{(u,w) \in (S \times T) \cap A} c(u,w).$$

Lemma: Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk $N = (V, A, c, s, t)$, so gilt

$$\text{val}(f) \leq \text{cap}(S,T).$$

Maxflow-Mincut-Theorem
Jedes Netzwerk $N = (V, A, c, s, t)$ erfüllt

$$\max_{f \text{ Fluss in } N} \text{val}(f) = \min_{(S,T) \text{ s-t-Schnitt in } N} \text{cap}(S,T).$$

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 $f(e) < c(e)$, 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} in 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: 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 $\text{val}(f) = \text{cap}(S, T)$.

Algorithm 6 Ford-Fulkerson

```
1: function FORD-FULKERSON(V,A,c,s,t)
2:     f ← 0                                   ▷ Fluss konstant 0
3:     while ∃ s-t-Pfad P ∈ (V, A_f) do        ▷ ∃ augmentierender Pfad
4:         Erhöhe den Fluss entlang P
5:     return f                                ▷ maximaler Fluss
```

Satz: 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, wobei $m = |A|$ und $n = |V|$.

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

2.1.3 Minimale Schnitte in Graphen

Wir betrachten in diesem Abschnitt ungerichtete ungewichtete Graphen $G = (V, E)$ ohne Schleifen, wobei mehrere Kanten zwischen denselben Knotenpaar erlauben - sogenannte **Multigraphen.**

Definition: Eine Menge $C \subseteq E$ für die $(V, E \setminus C)$ unzusammenhängend ist, nennen wir **Kantenschnitt** in G. Mit $\mu(G)$ bezeichnen wir die **Kardinalität** eines *kleinstmöglichen* Kantenschnitts in G.

Problemstellung: Gegeben einen Multigraphen G, bestimme $\mu(G)$. Wir nennen das das $MIN - CUT$-Problem.

Lemma: Sei G ein Graph und e eine Kante in G. Dann gilt $\mu(G \setminus e) \geq \mu(G)$ und falls es in G einen minimalen Schnitt C mit $e \notin C$ gibt, dann gilt $\mu(G \setminus e) = \mu(G)$.

Algorithm 7 CUT

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

Lemma: Sei $G = (V, E)$ ein Multigraph mit n Knoten. Falls e gleichverteilt zufällig unter den Kanten in G gewählt wird, dann gilt $\Pr[\mu(G) = \mu(G \setminus e)] \geq 1 - \frac{2}{n}$.

Satz: Für den Algorithmus der $\lambda \cdot \binom{n}{2}$-maligen Wiederholung von CUT(G) gilt:

- Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$
- Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$

2.2 Geometrische Algorithmen

2.2.1 Kleinster umschliessender Kreis

Problemstellung: Zu einer gegebenen Menge P von n Punkten in der Ebene, möchte man einen Kreis $C(P)$ bestimmen, so dass $C(P)$ alle Punkte aus P umschliesst und der *Radius* von $C(P)$ so klein wie möglich ist. Für einen Kreis C verwenden wir $C^\bullet$ für die von C umschlossene Kreisscheibe (inklusive C). Wir erlauben, dass auch die Punkte in P auch auf $C(P)$ liegen dürfen.

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

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

Algorithm 8 Complete Enumeration

```
1: function CompleteEnum(P)
2:     for all Q ⊆ P mit |Q| = 3 do
3:         bestimme C(Q)
4:         if P ⊆ C•(Q) then
5:             return C(Q)
```

Dieser Algorithmus berechnet $C(P)$ in $O(n^4)$.

Algorithm 9 Enclosing Circle Randomised

```
1: function RandomisedCleverVersion(P)
2:     wähle Q ⊆ P mit |Q| = 11 zufällig und gleichverteilt
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: Sei P eine Menge von n Punkten und für $r \in \mathbb{N}$, R zufällig gleichverteilt aus $\binom{P}{r}$. Dann ist die erwartete Anzahl Punkte von P, die ausserhalb von $C(R)$ liegen, höchstens $3\frac{n-r}{r+1} \leq 3\frac{n}{r+1}$.

Satz: Algorithmus RANDOMISEDCLEVERVERSION(P) berechnet den kleinsten umschliessenden Kreis von P in erwarteter Zeit $O(n \log(n))$.

2.2.2 Konvexe Hülle

Definition: Sei $S \subseteq \mathbb{R}^d$, $d \in \mathbb{N}$. Die **konvexe Hülle**, $\text{conv}(S)$, von S ist der Schnitt aller konvexen Mengen, die S enthalten, d.h.

$$\text{conv}(S) := \bigcap_{S \subseteq C \subseteq \mathbb{R}^d,\ C \text{ konvex}} C.$$

Definition: Ein geordnetes Paar qr, mit $q, r \in P$ und $q \neq r$, ist eine **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von der gerichteten Kante qr liegen.

Lemma: $(q_0, q_1, ..., q_{h-1})$ ist die **Eckfolge** des $\text{conv}(P)$ umschliessenden Polygons gegen den Uhrzeigersinn gdw. alle Paare (q_{i-1}, q_i), $i = 1, 2, ..., h$, *Randkanten* von P sind.

Algorithm 10 Jarvis Wrap

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

Algorithm 11 Jarvis Wrap Helper

```
1: function FindNext(q)
2:     Wähle p_0 ∈ P \ {q} beliebig
3:     q_next ← p_0
4:     for all p ∈ P \ {q, p_0} do
5:         if p rechts von qq_next then
6:             q_next ← p
7:     return q_next
```

Satz: Gegeben eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus JARVISWRAP(P) die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.

Ein anderer Ansatz zum Lösen des Problems ist **lokales Verbessern.** Als erstes sortieren wir P aufsteigend nach x-Koordinate. Sei $(p_1, p_2, ..., p_n)$ die sich ergebende Reihenfolge, dann betrachten wir das Polygon $(p_1, p_2, ..., p_{n-1}, p_n, p_{n-1,...,p_2,p_1})$.

Wir durchlaufen dieses Polygon von links nach rechts und führen immer für 3 sukzessive Punkte einen **Verbesserungsschritt** durch. Dieser ist definiert durch die Operation, bei der wir für die Punkte p, p', p'' feststellen, dass p' links von pp'' liegt, und wir deshalb p' aus der Folge entfernen. Am Ende dieses Prozesses erhalten wir die sog. **Triangulierung** der Punktemenge P.

Algorithm 12 Local Repair

```
1:  function LocalRepair((p_1, p_2, ..., p_n))          ▷ Folge sortiert nach x-Koordinate
2:      q_0 ← p_1
3:      h ← 0
4:      for i ← 2 to n do                    ▷ untere konvexe Hülle, von 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
9:      h' ← h
10:     for i ← n − 1 downto 1 do            ▷ obere konvexe Hülle, von 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
15:     return (q_0, q_1, ..., q_{h-1})
```

Satz: Gegeben eine Folge $p_1, p_2, ..., p_n$ nach x-Koordinate sortierter Punkte in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair$(p_1, p_2, ..., p_n)$ die konvexe Hülle von P in Zeit $O(n)$.

3 Runtime Overview

Bemerkung: Es gilt $n = |V|$ und E.

DFS()	$O(\|E\| + \|V\|)$
Eulertour()	$O(\|E\|)$
Hamiltonkreis()	$O(n^2 2^n)$
Kruskal()	$O(\|E\| \cdot \log \|V\|)$
Prim()	$O((\|V\| + \|E\|) \cdot \log \|V\|)$
Boruvka()	$O((\|V\| + \|E\|) \cdot \log \|V\|)$
Greedy-Matching()	$O(\|E\|)$
Greedy-Coloring()	$O(\|E\|)$
Bunte-Pfade()	$O(2^k km)$
Cut()	$O(n^2)$
Ford-Fulkerson()	$O(mnU)$
CleverRandomizedConvexCircle()	$O(n \cdot \log(n))$
JarvisWrap()	$O(nh)$
LocalRepair()	$O(n)$