

# Algorithms and Probability Summary

June 22, 2020

# Chapter 1

## Graphentheorie

### 1.1 Basics and Definitions

**Graph:** A graph is a tuple  $(V, E)$ , where  $V$  is a finite non empty set of vertices and  $E$  is a set of vertex pairs indicating the edges  $V \subseteq E$   
 $(E \subseteq \binom{V}{2} := \{(x, y) | x, y \in V, x \neq y\})$

**Complete (Vollständig):** There is an edge between each pair of vertices (den.  $K_n$ )

**Walk (Weg):** A sequence of vertices  $\langle v_1, v_2, \dots, v_n \rangle$  if  $\forall i$  there exists and edge from  $v_i$  to  $v_{i+1}$ . The length of the walk is given by the number of steps, i.e n-1.

**Path (Pfad):** A walk which doesn't contain any vertex more than once (den.  $P_n$ )

**Closed Walk (Zyklus):** A walk in which  $v_1 = v_n$  (den.  $C_n$ )

**Cycle (Kreis):** A closed walk with length of atleast three and the vertices  $v_1, \dots, v_{k-1}$  are pairwise distinct (in a directed graph it must have length of atleast two)

**Loops (Schlingen):** An edge from a vertex to itself

**Multiple edges (Mehrfachkanten):** When vertex pairs are connected by multiple edges

**Multigraph (Multigraph):** A Graph which contains loops and multiple edges (In this lecture we assume that a graph is not a multigraph unless stated otherwise)

**Neighbourhood (Nachbarschaft):** All outgoing and incoming edges to/from a vertex  $v$  denoted  $N_G(v) := \{u \in V | \{v, u\} \in E\}$

**Degree (Grad):** Indicates the size of the neighbourhood  $\deg_G(v) := |N_G(v)|$

**k-regular (k-regulär):** If every vertex  $v \in V$  has degree  $\deg(v) = k$   
o A complete graph  $K_n$  is n-1-regular

**Adjacent (Adjazent):** Two vertices  $u$  and  $v$  if there is an edge  $u, v$

**Satz 1.2** For any Graph  $G = (V, E)$  we have  $\sum_{v \in V} \deg(v) = 2|E|$

**Korollar 1.3** For any Graph  $G = (V, E)$  the number of vertices with uneven degree is even. (Direct Proof by splitting  $V$  into even and odd degree sets then use S1.2)

**Subgraph (Teilgraph):** A Graph  $H = (V_H, E_H)$  is a subgraph of a graph  $G = (V_G, E_G)$  if  $V_H \subseteq V_G$  and  $E_H \subseteq E_G$  denoted  $H \subseteq G$

**Induced Subgraph (Induzierte Teilgraph):** If  $E_H = E_G \cap \binom{V_H}{2}$  denoted  $H = G[V_H]$ . If there is an edge  $(u, v)$  in  $G$  and  $u, v$  are also vertices in  $H$  then there must be an edge  $(u, v)$  in  $H$  aswell

#### 1.1.1 Connectivity and Trees

**Connected (Zusammenhängend):** if for any Vertices  $s, t \in V$  there is a  $s-t$  path. A subgraph  $C \subseteq G$  for which this trait is maximal is called a connected component (hence for all subgraphs  $H \neq C$  with  $C \subseteq H \subseteq G$  is not connected)

**Cycle Free (Kreisfrei):** A Graph which doesn't contain a cycle

**Tree (Baum):** A Graph which is Cycle free and connected

**Leaf (Blatt):**  $T = (V, E)$  a tree and  $v \in V$  a vertice with  $\deg(v) = 1$

**Lemma 1.5:**  $T = (V, E)$  a tree with  $|V| \geq 2$ , it follows:

**a):**  $T$  contains atleast 2 leafs. (Proof: If there was only one leaf  $2|E| = \sum_{v \in V} \deg(v) \geq 1 + 2(|V| - 1)$  which is a contradiction to S1.6)

**b):** if  $v \in V$  is a leaf, the graph  $T-v$  is also a tree.

**Satz 1.6**  $G = (V, E)$  a Graph with  $|V| \geq 1$  vertices, the following is equivalent:

- $G$  is a tree
- $G$  is connected and cycle free
- $G$  is connected and  $|E| = |V| - 1$
- $G$  is cycle free and  $|E| = |V| - 1$
- for any  $x, y \in V$ ,  $G$  contains exactly one  $x-y$  path.

**Forrest (Wald):**  $W = (V, E)$  a graph which is cycle free, every component of a forrest is a tree

**Lemma 1.7** A forrest  $G = (V, E)$  contains  $|V| - |E|$  connected components (Proof by induction)

**Directed Graph (Gerichteter Graph):** A graph where the edges are represented by ordered pairs, i.e The directed graph  $D$  is given by the tuple  $(V, A)$  where  $V$  is the set of vertices and  $A \subseteq V \times V$  a set of directed edges. Compared to an undirected graph, between two vertices there can be two edges  $(x, y)$  and  $(y, x)$

**Out-Degree(Aus-Grad):**  $\deg^+(v) := |\{(x, y) \in A | x = v\}|$

**In-Degree(In-Grad):**  $\deg^-(v) := |\{(x, y) \in A | y = v\}|$

**Satz 1.8** For any directed graph  $D = (V, A)$  the following is true.

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |A|.$$

**Acyclic (Azyklisch):** A directed graph which deosn't contain a cycle (DAG). DAG's have a topological ordering.

**Satz 1.9** For any DAG  $D = (V, A)$  we can find a topological ordering in  $\mathcal{O}(|V| + |A|)$

**Strongly Connected (start zusammenhängend):** For a DAG  $D = (V, A)$ , if for every pair of vertices  $u, v \in V$  a directed  $u-v$ -Path exists

**Weakly Connected (schwach zusammenhängend):** When the underlying graph(i.e ignoring the direction of the edges) is connected

### 1.1.2 Datastructures

The two main ways of storing graphs, is with Adjacency matrices and Adjacency lists.

## 1.2 Trees

### 1.3 Paths

**Shortest Paths** Given: A connected Graph  $G = (V, E)$ , two vertices  $s, t \in V$  and a costfunction  $c : E \rightarrow \mathbb{R}$

Goal: Find an  $s-t$ -path  $P$  in  $G$  with  $\sum_{e \in P} c(e) = \min$

Algorithms which can solve shortest path problems:

1. Dijkstras Algorithm
2. Floyd-Warshall
3. Bellman-Ford
4. Johnson's Algorithm

## 1.4 Connection

**Definition 1.23** A Graph  $G = (V, E)$  is **k-connected (k-zusammenhängend)** if  $|V| \geq k + 1$  and for all subsets  $X \subseteq V$  with  $|X| < k$  the following is true: The Graph  $G[V \setminus X]$  is connected. (Hence you would need to remove atleast  $k$  vertices to destroy the connectivity of the graph, the only exception is a complete graph which is by definition  $k-1$  connected)

**Definition 1.24** A Graph  $G = (V, E)$  is **k-edge-connected (k-kanten-zusammenhängend)**, if for all subsets  $X \subseteq E$  with  $|X| < k$  the following is true:  $(V, E \setminus X)$  is connected. (Hence atleast  $k$  edges must be removed to destroy the connectivity of the Graph)

**Satz 1.25 Menger**  $G = (V, E)$  the following applies:

1.  $G$  is  $k$ -connected iff for all pairs of vertices  $u, v \in V$ ,  $u \neq v$ , atleast  $k$  internal-vertex disjoint  $u-v$  paths exist
2.  $G$  is  $k$ -edge-connected iff for all pairs of vertices  $u, v \in V$ ,  $u \neq v$ , atleast  $k$  edge-disjoint  $u-v$ -paths exist

#### 1.4.1 Articulation vertice (Artikulationsknoten)

**Articulation vertice (Artikulationsknoten):** If a graph is connected but not 2-connected, then there exists a vertice  $v$  with the attribute that  $G[V \setminus \{v\}]$  is not connected. Articulation vertices kann be detected using a modified DFS.

**Forward Edge (Vorwärtskante):** An edge starting from a vertice with a lower dfs number than the destination vertice

**Backwards Edge(Rückwärtsskante):** An edge starting from a vertice with a higher dfs number than the destination vertice

we assign  $\in V$  a number  $low[v] :=$  the smallest dfs-Number, that can be reached from the vertice  $v$  using any number of forward edges and atmost one backward edge. It follows for all  $v \in V$ :  $low[v] \leq dfs[v]$

$v$  is an Articulation vertice  $\Leftrightarrow v=s$  and  $s$  has degree of atleast 2 or  $v \neq s$  and there exists a  $w \in V$  with  $\{v, w\} \in E(T)$  and  $low[w] \geq dfs[v]$

**TODO:** Implement DFS-Visit which finds the articulation vertices for a given graph

**Satz 1.27** For a connected graph  $G = (V, E)$ , implemented with an adjacency list Articulation vertices can be found in  $\mathcal{O}(|E|)$

#### 1.4.2 Bridges (Brücken)

**Bridge (Brücke):** An edge  $e \in E$  such that  $(V, E \setminus \{e\})$  is not connected

From the definition of the bridge it follows that a spanning tree must contains all bridges of a graph and that the vertices at the end of the bridge are either Articulation vertices or vertices with degree 1.

An edge  $(v, w)$  of the depth-first-search tree is a bridge iff  $low[w] > dfs[v]$

**Satz 1.28** For a connected graph  $G = (V, E)$  implemented with an adjacency list, articulation vertices and bridges can be found in  $\mathcal{O}(|E|)$

### 1.5 Cycles

#### 1.5.1 Eulerwalk (Eulertour)

**Definition 1.29** A Eulerwalk in a graph  $G = (V, E)$  is a cycle which contains each edge exactly once. If  $G$  contains a Eulerwalk then  $\deg(v)$  of all  $v \in V$  is even. (Proof by contradiction assume  $G$  has vertices of even degrees pick a starting node  $v$  and an arbitrary node  $u$  and show that the path cannot end in  $u$  arguing with the parity of the degree)

In a connected graph eulerian graph a Eulerwalk can be found in  $\mathcal{O}(|E|)$

**TODO:** Implement an Algorithm which can find a Eulerwalk in  $\mathcal{O}(|E|)$

#### 1.5.2 Hamilton Cycles (Hamiltonkreise)

**Defintion 1.31** A Hamilton Cycle in a graph  $G = (V, E)$  is a cycle in which all vertices of  $V$  are visited exactly once. If a graph contains a Hamilton Cycle it is called hamiltonian (hamiltonisch). Wether or not a graph contains a hamilton cycle is NP-complete.

**Satz 1.33** The Algorithm HAMILTONKREIS to find a Hamilton cycle of a given graph  $G$  needs  $\mathcal{O}(n * 2^n)$  memory and has a runtime of  $\mathcal{O}(n^2 * 2^n)$ , where  $n = |V|$

**TODO:** Implement the Algorithm HAMILTONKREIS

#### 1.5.3 Special Cases

**Lattice (Gittergraph)** An  $m \times n$  lattice is hamiltonian if  $m$  or  $n$  is even (Proof using parity argument)

**Lemma 1.35**  $G = (A \uplus B, E)$  a bipartite Graph with  $|A| \neq |B|$ , then  $G$  cannot contain a Hamilton cycle

**Hypercube (Hyperwürfel):** The set of edges of a Hypercube  $H_d$  is  $\{0, 1\}^d$  hence the set of all 0-1 sequences of length d. Two vertices are connecten if their sequences differ at exactly one spot. A d-dimensional Hypercube contains a Hamilton cycle for all  $d \geq 2$  (Proof by induction)

**Satz 1.37(Dirac)** If  $G = (V, E)$  is a graph with  $|V| \geq 3$  vertices and each vertice has atleast  $|V|/2$  neighbours, then G is hamiltonian.

#### 1.5.4 The Travelling Salesman Problem

**Given:** A complete Graph  $K_n$  and a function  $l : \binom{[n]}{2} \rightarrow \mathbb{N}$  which gives each edge a length

**Goal:** Find a Hamilton cycle C in  $K_n$  with  $\sum_{e \in C'} l(e) = \min\{\sum_{e \in C'} l(e) | C' \text{ is a Hamilton cycle in } K_n\}$

For a graph  $G = (V, E)$  with  $V = [n]$  vertices we define a weight function  $l$  as  $l(\{u, v\}) = \begin{cases} 0, & \text{if } \{u, v\} \in E \\ 1, & \text{else} \end{cases}$

hence the length of a minimal Hamilton cycle in  $K_n$  when using  $l$  is 0 iff G contains a Hamitoncycle, hence this allows us evaluate the efficiency of a solution. We define the optimum solution as  $opt(K_n, l) := \min\{\sum_{e \in C'} l(e) | C' \text{ is a Hamilton cycle in } K_n\}$ . An algorithm which always finds a Hamilton cycle with  $\sum_{e \in C} l(e) \leq \alpha * opt(K_n, l)$  is known as an  $\alpha$ -Approximation algorithm.

**Satz 1.39** If there exists an  $\alpha$ -Approximation algorithm for an  $\alpha > 1$  for the Traveling Salesman Problem with a runtime of  $\mathcal{O}(f(n))$ , then an algorithm exits for all graphs with n vertices which decides wether it is hamiltonian or not in the same time complexity

#### Metric Travelling Salesman Problem (Metrisches TSP)

**Given:** A complete Graph  $K_n$  and a function  $l : \binom{[n]}{2} \rightarrow \mathbb{N}$  with the condition  $l(\{x, z\}) \leq l(\{x, y\}) + l(\{y, z\})$  for all  $x, y, z \in [n]$

**Goal:** Find a Hamilton cycle C in  $K_n$  with  $\sum_{e \in C'} l(e) = \min\{\sum_{e \in C'} l(e) | C' \text{ is a Hamilton cycle in } K_n\}$  The condition is called the triangle inequality (Dreiecksungleichung) and states that the direct connection between two vertices x and z cannot be longer than the detour over the vertice y.

**Satz 1.40** The Metric Travelling Salesman Problem has a 2-Approximation algorithm with a runtime of  $\mathcal{O}(n^2)$

### 1.6 Matchings

**Matching** A set of edges  $M \subseteq E$  is called a Matching of a graph  $G = (V, E)$ , if no vertice of the graph is incident to more than one edge from M i.e  $e \cap f = \emptyset$  for all  $e, f \in M$  with  $e \neq f$ . A vertice v is "covered" (überdeckt) if there is an edge  $e \in M$  which contains v.

**Perfect Matching** If each vertice is covered by exactly one edge of the Matching M i.e  $|M| = |V|/2$ . Not all graphs contain a perfect matching for example a star graph.

**Maximal matching (Inklusionsmaximal):**  $G = (V, E)$ , for a Matching M if  $M \cup \{e\}$  is not a Matching for all edges  $e \in E \setminus M$ .

**Maximum matching (Kardinalitätsmaximal):**  $G = (V, E)$  for a Matching M if  $|M| \geq |M'|$  for all Matchings M' in G

**Example:** A path consisting of three edges. Creating a Matching using the middle edge would creat a Maximal matching but not a Maximum matching. A Maximum and a Maximal Matching can be created by taking the two outer edges

#### 1.6.1 Algorithms

**GREEDY MATCHING** This algorithm picks random edges from E and adds it to the matching at the same time it deletes all incident edges from E. The algorithm stops when  $E = \emptyset$ . This algorithm can find a Maximal matching in time  $\mathcal{O}(|E|)$  for which the following applies:  $|M_{Greedy}| \geq \frac{1}{2}|M_{max}|$  where  $M_{max}$  is a Maximum matching

**Union of two Matchings:** Let  $M_1$  and  $M_2$  be arbitrary Matchings.  $G_M = (V; M_1 \cup M_2)$ . Every vertice in  $G_M$  has degree of atmost 2, hence all components of the graph are paths and/or cycles (cycles having even length). If we assume  $|M_1| < |M_2|$ , then every cycle/path of even length will have the same amount of edges from  $M_1$  as  $M_2$ . From our assumption we know that there must be a path P which contains more edges from  $M_2$  than  $M_1$  the two outer edges belonging to  $M_2$ . Hence we can create a new Matching  $M'_1$  using P which will contain one more edge. We achieve this by switching the edges in P i.e  $M'_1 := (M_1 \cup (P \cap M_2)) \setminus (P \cap M_1)$ . We say P is a  $M_1$ -augmented path.

**M-augmented Path (augmentierender Pfad):** Let M be an arbitrary Matching, an M augmented path is a path where the last two edges of P are not covered by M and P consists of edges alternating between edges belonging to M and not belonging to M

**AUGMENTED MATCHING** This algorithm finds a Maximum matching. We start by finding a Matching which consists of only one arbitrary edge. As long as the matching is not a maximum matching we repeat the following: We take an augmented path and we increase the size of the Matching. We know that after  $|V|/2 - 1$  times the matching will be maximum because a matching can't have more than  $|V|/2$  edges. We can easily find augmented paths in a bipartite Graph using a modified BFS. The total runtime of our algorithm is  $\mathcal{O}(|V| \cdot |E|)$

**Satz 1.45** If  $n$  is even and  $l : \binom{[n]}{2} \rightarrow \mathbb{N}$  a weight function of the complete graph  $K_n$  then we can find a minimal perfect Matching (i.e a matching where the sum of edge weights are minimal) in  $\mathcal{O}(n^3)$

**Satz 1.46** From S1.45 it follows that for the Metric Travelling Salesman Problem there is a  $3/2$ -Approximationsalgorithm with a runtime of  $\mathcal{O}(n^3)$

### 1.6.2 Der Satz von Hall

**Bipartite** A graph  $G = (V, E)$  is bipartite if we can partition the set of vertices  $V$  into two sets  $A$  and  $B$  such that all edges in  $E$  contain a vertex from  $A$  and a vertex from  $B$  (denoted:  $G = (A \sqcup B, E)$ )

**Satz von Hall/Heiratssatz:** For a bipartite graph  $G = (A \sqcup B, E)$  there is a Matching  $M$  of cardinality  $|M| = |A|$  iff  $|N(X)| \geq |X|$  for all  $X \subseteq A$ . From the satz von Hall it follows that a  $k$ -regular graph always has a perfect matching.

**Satz 1.48** Let  $G = (A \sqcup B, E)$  be a  $k$ -regular bipartite graph. There exists an  $M_1, \dots, M_k$  such that  $E = M_1 \sqcup \dots \sqcup M_k$  and all  $M_i, 1 \leq i \leq k$  are perfect matchings in  $G$ . The perfect matching can be found in  $\mathcal{O}(|E|)$ .

**Satz 1.49** Let  $G = (V, E)$  a  $2^k$ -regular bipartite graph. We can find a perfect matching in  $\mathcal{O}(|E|)$

## 1.7 Colouring (Färbungen)

**Vertex colouring (Knoten Färbung):** The vertex colouring of a graph  $G = (V, E)$  with  $k$  colours is a mapping  $c : V \rightarrow [k]$  such that the following holds:  $c(u) \neq c(v)$  for all edges  $u, v \in E$

**Chromatic number (chromatische Zahl):** denoted  $x(G)$  is the minimal number of colours needed to color the vertices of  $G$ . A complete graph has the chromatic number  $n$ . Cycles of even length have chromatic number 2, uneven length have a chromatic number 3. Trees with atleast two vertices have a chromatic number 2. Graphs with chromatic number  $k$  are also called  $k$ -partite. To decide wether or not a graph  $G$  is bipartite can be done in  $\mathcal{O}(|E|)$  with a DFS of BFS.

**Satz 1.53** A graph  $G = (V, E)$  is bipartite iff it does not contain a cycle of odd length as a subgraph

**Satz 1.54 (Vierfarbensatz):** Any map can be coloured using 4 colours.

**GREEDY FARBURGUNG** This algorithm calculates the colouring of a graph by picking vertices at random and giving it the lowest colour not used by its neighbours. There exists a order of vertices for which the GREEDY algorithm needs  $x(G)$  colors.

**Satz 1.55** Let  $G$  be a connected graph. For the number  $C(G)$  of colours needed by GREEDY FARBURGUNG to color the graph  $G$  the following applies:  $x(G) \leq C(G) \leq \Delta(G) + 1$  ( $\Delta(G) := \max_{v \in V} \deg(v)$ , hence the max degree of a vertice in  $G$ ). If the graph is saved in an adjacency list then we can find the colouring in  $\mathcal{O}(|E|)$

**Satz 1.59 (Satz von Brooks)** Let  $G = (V, E)$  be a connected graph which is not complete nor a cycle with odd degree i.e  $G \neq K_n$  and  $G \neq C_{2n+1}$  then the following holds  $x(G) \leq \Delta(G)$  and there exists an Algorithm which can colour the graph in  $\mathcal{O}(|E|)$  with  $\Delta(G)$  colours.

**Satz 1.60** Let  $G = (V, E)$  be a graph and  $k \in \mathbb{N}$  a natural number such that every induced subgraph of  $G$  has a vertice with degree atmost  $k$ . It follows that  $x(G) \leq k + 1$  and we can find a  $(k+1)$ -colouring in  $\mathcal{O}(|E|)$

**Satz 1.61 (Mycielski-Konstruktion):** For all  $k \geq 2$  there is a triangle free graph  $G_k$  with  $x(G_k) \geq k$  (Proof by induction)

**Satz 1.62** Every 3-colourable graph  $G = (V, E)$  can be coloured in  $\mathcal{O}(|E|)$  with  $\mathcal{O}(\sqrt{|V|})$  colours. Given a graph  $G = (V, E)$ , is  $x(G) \leq 3$  is NP-Complete.

## Chapter 2

# Probability Theory and Randomised Algorithms

### 2.1 Definitions and Notations

**Definition 2.1** A discrete Probabilityspace(diskreter Wahrscheinlichkeitsraum) is defined by a Set of **outcomes** (Ergebnismenge) denoted  $\Omega = \{\omega_1, \omega_2, \dots\}$  of elementary outcomes(Elementarereignissen). Each elementary outcome  $\omega_i$  is assigned a (elementary)probability(Elementar – Wahrscheinlichkeit) denoted  $Pr[\omega_i]$  where  $0 \leq Pr[\omega_i] \leq 1$  and  $\sum_{\omega \in \Omega} Pr[\omega] = 1$ . A set  $E \subseteq \Omega$  is called an **outcome(Ereignis)** The probability of  $Pr[E]$  of an outcome is defined by:  $Pr[E] := \sum_{\omega \in E} Pr[\omega]$   
If  $E$  is an outcome, we define  $\bar{E} := \Omega \setminus E$  the **compliment outcome(Komplementarereignis)**

**Finite probabilityspace (endlicher Wahrscheinlichkeitsraum):** A Probability space  $\Omega = \{\omega_1, \dots, \omega_n\}$  (Assumption for infinite probability spaces  $\Omega = \mathbb{N}_0$ )

**Lemma 2.2** For outcomes  $A, B$  the following applies:

1.  $Pr[\emptyset] = 0, Pr[\Omega] = 1$
2.  $0 \leq Pr[A] \leq 1$
3.  $Pr[\hat{A}] = 1 - Pr[A]$
4. if  $A \subseteq B, Pr[A] \leq Pr[B]$
5. (Additionssatz) Wenn die Ereignisse  $A_1, \dots, A_n$  paarweise disjunkt sind (also wenn für alle Paare  $i \neq j$  gilt, dass  $A_i \cap A_j = \emptyset$ ) so folgt:

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

**Union Bound (Boolesche Ungleichung):** Für beliebige Ereignisse  $A_1, \dots, A_n$  gilt:

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

**Siebformel, Prinzip Inklusion/Exklusion:** Für Ereignisse  $A_1, \dots, A_n (n \geq 2)$  gilt:

$$\begin{aligned} Pr\left[\bigcup_{i=1}^n A_i\right] &= \sum_{i=1}^n Pr[A_i] - \sum_{1 \leq i_1 \leq i_2 \leq n} Pr[A_{i_1} \cap A_{i_2}] + \dots \\ &\quad + (-1)^{l+1} \sum_{1 \leq i_1 < \dots < i_l \leq n} Pr[A_{i_1} \cap \dots \cap A_{i_l}] + \dots \\ &\quad + (-1)^{n+1} \cdot Pr[A_1 \cap \dots \cap A_n] \end{aligned}$$

**Laplace Raum:** endlicher Wahrscheinlichkeitsraum, in dem alle Elementarereignisse gleich wahrscheinlich sind. In einem Laplace-Raum gilt für jedes Ereignis  $E$ :

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

# Möglichkeiten  $k$  Elemente aus einer  $n$ -elementigen Menge zu ziehen :

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

(a)

	geordnet	ungeordnet
mit Zurücklegen	(1, 1), (1, 2), (1, 3) (2, 1), (2, 2), (2, 3) (3, 1), (3, 2), (3, 3)	{1, 1}, {1, 2}, {1, 3} {2, 2}, {2, 3}, {3, 3}
ohne Zurücklegen	(1, 2), (2, 1), (2, 1) (2, 3), (3, 1), (3, 2)	{1, 2}, {1, 3}, {2, 3}

(b)

$$(n^k := n(n-1)(n-2)\dots(n-k-1) = \frac{n!}{(n-k)!})$$

**Bedingte Wahrscheinlichkeit:** A und B seien Ereignisse mit  $Pr[B] > 0$ . Die bedingte Wahrscheinlichkeit  $Pr[A|B]$  (Die W'keit, dass Ereignis A eintrifft, wenn wir schon wissen dass Ereignis B eingetreten ist) von A gegeben B ist definiert durch:

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

**Satz 2.12 Satz der totalen W'keit:** Die Ereignisse  $A_1, \dots, A_n$  seien paarweise disjunkt und es gelte  $B \subseteq A_1 \cup \dots \cup A_n$  dann folgt:

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

**Satz 2.10 Multiplikationssatz:** Seien die Ereignisse  $A_1, \dots, A_n$  gegeben. Falls  $\Pr[A_1 \cap \dots \cap A_n] > 0$  ist, gilt:

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

**Satz von Bayes:** Die Ereignisse  $A_1, \dots, A_n$  seien paarweise disjunkt. Ferner  $B \subseteq A_1 \cup \dots \cup A_n$  ein Ereignis mit  $Pr[B] > 0$ . Dann gilt für ein beliebiges  $i = 1, \dots, n$ :

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

**Unabhängigkeit Zwei Ereignisse:** Die Ereignisse A und B heissen unabhängig, wenn gilt:

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

**Unabhängigkeit:** Die Ereignisse  $A_1, \dots, A_n$  heißen unabhängig, wenn für alle Teilmengen  $I \subseteq \{1, \dots, n\}$  mit  $I = \{i_1, \dots, i_k\}$  gilt, dass

$$Pr[A_{11}, \cap \dots \cap A_{ik}] = Pr[A_{i1}] \dots Pr[A_{ik}] \quad (2.2)$$

Eine unendliche Familie von Ereignissen  $A_i$  mit  $i \in \mathbb{N}$  heisst unabhängig, wenn (2.2) für jede endliche Teilmenge  $I \subseteq \mathbb{N}$  erfüllt ist. Dies ist offensichtlich erfüllt, wenn die Ereignisse physikalisch unabhängig sind (e.g wenn jedes  $A_i$  einem unabhängigem Münzwurf entspricht) aber ist nicht unbedingt erforderlich.

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

$$Pr[A_1^{s_1} \cap \dots \cap A_n^{s_n}] = Pr[A_1^{s_1}] \dots Pr[A_n^{s_n}]$$

wobei  $A_i^0 = \overline{A_i}$  und  $A_i^1 = A_i$

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

Eine Funktion welche jede Element unser Wahrscheinlichkeitsraum ein Reelle Zahl zuordnet.

$$X : \Omega \rightarrow \mathbb{R}$$

"X ≤ 5" steht für

---

Wir werfen eine Münze drei Mal:  $X := \text{Anzahl „Kopf“}$

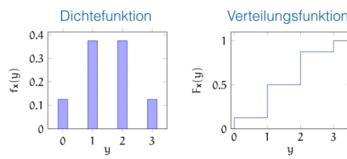
Wahrscheinlichkeitsraum:  $\Omega = \{\text{KKK}, \text{KKZ}, \dots, \text{ZZZ}\}$  Wir werfen einen Würfel drei Mal:

$$\text{Prob}[X=0] = \Pr[\{\text{ZZZ}\}] = 1/8$$

$$\text{Prob}[X=1] = \Pr[\{\text{ZZK}, \text{ZKZ}, \text{KZZ}\}] = 3/8$$

$$\text{Prob}[X=2] = \Pr[\{\text{ZKK}, \text{KKZ}, \text{KZK}\}] = 3/8$$

$$\text{Prob}[X=3] = \Pr[\{\text{KKK}\}] = 1/8$$



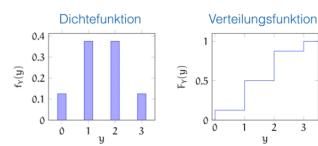
(a)

Wahrscheinlichkeitsraum:  $\Omega = \{1, 2, 3, 4, 5, 6\}^3$

$$\text{Prob}[Y=0] = \Pr[\{\text{erster Wurf gerade}\} \cdot \Pr[\{\text{zweiter Wurf gerade}\} \cdot \Pr[\{\text{dritter Wurf gerade}\}] = 1/8$$

$$\text{Prob}[Y=1] = \Pr[\{\text{erster Wurf gerade}\} \cdot \Pr[\{\text{zweiter Wurf gerade}\} \cdot \Pr[\{\text{dritter Wurf ungerade}\}] + \Pr[\{\text{erster Wurf gerade}\} \cdot \Pr[\{\text{zweiter Wurf ungerade}\} \cdot \Pr[\{\text{dritter Wurf gerade}\}] + \Pr[\{\text{erster Wurf ungerade}\} \cdot \Pr[\{\text{zweiter Wurf gerade}\} \cdot \Pr[\{\text{dritter Wurf gerade}\}] = 3/8$$

$$\text{Prob}[Y=2] = 3/8, \quad \text{Prob}[Y=3] = 1/8$$



(b)

### Bernoulli-Verteilung:

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

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

$$\mathbb{E}[X] = p$$

### Binomial Verteilung:

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

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

$$\mathbb{E}[X] = np \quad \text{Var}[X] = np(1-p) \quad (\text{gleichung für Varianz gilt nur wenn die } X_i \text{'s unabhängig sind})$$

$\text{Bin}(n, \frac{\lambda}{n})$  konvergiert für  $n \rightarrow \infty$  gegen  $\text{Po}(\lambda)$  Beispiel: Werfen einer Münze  $n$  mal,  $X = \text{Anzahl Kopf}$

### Negative Binomialverteilung:

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

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

$$\mathbb{E}[X] = \frac{n}{p}$$

Beispiel: Warten auf den  $n$ -ten Erfolg

### Geometrische Verteilung:

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

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

$$F_X(n) = 1 - (1-p)^n \quad \text{für alle } n=1, 2, \dots$$

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

Beispiel: Wiederholtes werfen einer Münze,  $X = \# \text{ Würfe bis zum ersten Mal Kopf}$

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

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

Beispiel: Wahrscheinlichkeit im ersten Wurf Kopf zu bekommen ist identisch zur Wahrscheinlichkeit nach 1000 Fehlversuchen im 1001ten Wurf Kopf zu bekommen.

### Poisson-Verteilung:

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

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

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

Beispiel: Modellierung seltener Ereignisse e.g  $X := \# \text{ Herzinfarkte in der Schweiz in der nächsten Stunde}$

**Erwartungswert:** Den 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.  
Ist  $X$  eine Zufallsvariable, so gilt:

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

Sei  $X$  eine Zufallsvariable mit  $W_X \subseteq \mathbb{N}_0$ . Dann gilt

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

**Beobachtung:** 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:  $\mathbb{E}[X_A] = Pr[A]$

- Schnitt:  $A \cap B \quad X_{A \cap B} = X_A \cdot X_B$
- Komplement:  $\bar{A} := \Omega \setminus A \quad X_{\bar{A}} = 1 - X_A$
- Vereinigung:  $A \cup B \quad X_{A \cup B}$

Beispiel:

Beispiel: Wir werfen eine Münze 100 Mal X := Anzahl Kopf

Setze für alle  $i = 1, \dots, 100$ :

X<sub>i</sub> := Indikatorvariable für „Kopf“ im **i**ten Wurf

Dann X = X<sub>1</sub> + ... + X<sub>100</sub>

und E[X<sub>i</sub>] = 1/2 und wegen der Linearität des Erwartungswertes

daher E[X] = E[X<sub>1</sub>] + ... + E[X<sub>100</sub>] = 50

Figure 2.3

**Linerarität des Erwartungswerts:** Für Zufallsvariablen  $X_1, \dots, X_n$  und  $X = a_1 X_1 + \dots + a_n X_n + b$  mit  $a_1, \dots, a_n, b \in \mathbb{R}$  gilt:

$$\mathbb{E}[X] = a_1 \mathbb{E}[X_1] + \dots + a_n \mathbb{E}[X_n] + b$$

**Stabiler Menge:** Knoten, die nicht durch Kanten verbunden sind.

**Satz** Für jeden Graphen  $G = (V, E)$  mit  $|V| = n$  und  $|E| = m$  bestimmt der Algorithmus (gehe durch die Knotenmenge und entferne den Knoten und inzidente Kanten mit Wahrscheinlichkeit  $1-p$ , bei den übrig gebliebenen Kanten lösche ein Knoten) eine stabile Menge  $S$  mit

$$\mathbb{E}[S] \geq np - mp^2$$

Beweis:

$X :=$  Anzahl Knoten, die erste Runde „überleben“  
 $\Rightarrow$  jeder einzelne Knoten überlebt mit Wahrscheinlichkeit  $p$ ,  
 wir haben  $n$  Knoten  
 $\Rightarrow$  (Linearität des Erwartungswertes)  $E[X] = np$

$Y :=$  Anzahl Kanten, die erste Runde „überleben“  
 $\Rightarrow$  jede einzelne Kante überlebt mit Wahrscheinlichkeit  $p^2$ ,  
 wir haben  $m$  Kanten  
 $\Rightarrow$  (Linearität des Erwartungswertes)  $E[Y] = mp^2$

$S \geq X - Y$  da wir höchstens einen Knoten pro Kante löschen  
 $\Rightarrow$  (Linearität des Erwartungswertes)  $E[S] \geq E[X] - E[Y]$

Figure 2.4

**Coupon Collector:**

Szenario: Es gibt  $n$  verschiedene Bilder in jeder Runde erhalten wir (gleichwahrscheinlich) eines der Bilder  
 $X :=$  Anzahl Runden bis wir alle  $n$  Bilder besitzen Ziel: Berechne  $E[X]$

Lösungsansatz: betrachte  $n$  Phasen

Phase i: Runden während wir  $i-1$  verschiedene Bilder besitzen  
 $X_i :=$  Anzahl Runden in Phase i,  $X_i \sim \text{Geo}( (n-(i-1))/n )$

Beispiel  $n=4$ :

$\underbrace{2, 2}_{1}, \underbrace{1, 2}_{2}, \underbrace{2, 2}_{3}, \underbrace{3, 1, 3, 2}_{4}, \underbrace{3, 1, 4}_{6}$	<small>erhaltenes Bild</small> <small>Phase</small> <small><math>X_i</math></small>	$\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i] = \sum_{i=1}^n \frac{n}{n-i+1} = n \cdot \sum_{i=1}^n \frac{1}{i} = n \cdot H_n,$ $H_n = \ln n + O(1)$
(a)		(b)

$\Rightarrow$  die Laufzeit der Coupon Collector ist  $\mathcal{O}(n \ln n + n)$

**Varianz:** Für ein Zufallsvariable  $X$  mit  $\mu = E[X]$  definieren wir die Varianz  $Var[X]$  durch:

$$Var[X] := E[(X - \mu)^2] = \sum_{x \in W_X} (x - \mu)^2 \cdot Pr[X = x]$$

$$Var[X] = E[X^2] - E[X]^2$$

$$Var[a \cdot X + b] = a^2 \cdot Var[X] \quad X \text{ beliebig, } a, b \in \mathbb{R}$$

(b verschwindet, da verschieben der werte kein einfluss auf die Abweichung vom Durchschnitt hat)

Standardabweichung:

$$\sigma := \sqrt{Var[X]}$$

**Dichten:**  $X, Y$  Zufallsvariablen:

Gemeinsame Dichte:

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

Randdichte:

$$f_X(x) = \sum_{y \in W_Y} f_{:X,Y}(x, y)$$

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

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

Alternativ:

$$f_{X_1, \dots, X_n}(x_1, \dots, x_n) = f_{X_1}(x_1) \cdot \dots \cdot f_{X_n}(x_n) \text{ für alle } (x_1, \dots, x_n) \in W_{X_1} \times \dots \times W_{X_n}$$

Beispiel:

$$\Omega = \{1, 2, 3, 6\} \text{ mit } Pr[\omega] = 1/4 \text{ für alle } \omega \in \Omega$$

$$\begin{aligned} X(\omega) &= \begin{cases} 1 & \text{wenn } \omega \text{ durch 2 teilbar} \\ 0 & \text{sonst} \end{cases} & f_X(i) &= \begin{cases} 1/2 & \text{für } i=0,1 \\ 0 & \text{sonst} \end{cases} \\ Y(\omega) &= \begin{cases} 1 & \text{wenn } \omega \text{ durch 3 teilbar} \\ 0 & \text{sonst} \end{cases} & f_Y(i) &= \begin{cases} 1/2 & \text{für } i=0,1 \\ 0 & \text{sonst} \end{cases} \\ f_{X,Y}(i,j) &= \begin{cases} 1/4 & \text{für alle } (i,j) \in \{0,1\} \times \{0,1\} \\ 0 & \text{sonst} \end{cases} & \Rightarrow & X \text{ und } Y \text{ sind unabhängig} \end{aligned}$$

Figure 2.6

Für zwei Indikatorvariablen X und Y gilt:

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

Lemma: Sind  $X_1, \dots, X_n$  unabhängige Zufallsvariablen und  $S_1, \dots, S_n$  beliebige Mengen mit  $S_i \subseteq W_{X_i}$ , dann gilt:

$$Pr[X_1 \in S_1, \dots, X_n \in S_n] = Pr[X_1 \in S_1] \dots Pr[X_n \in S_n]$$

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

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

Summe von Zufallsvariablen: Für zwei unabhängige Zufallsvariablen X und Y sei  $Z := X + Y$ . Es gilt:

$$f_Z(z) = \sum_{x \in W_X} f_X(x) \cdot f_Y(z - x)$$

Es folgt:

$$Poisson(\lambda_1) + Poisson(\lambda_2) = Poisson(\lambda_1 + \lambda_2) \quad Bon(n_1, p) + Bin(n_2, p) = Bin(n_1 + n_2, p)$$

falls die lambda's und n's unabhängig sind

Rechenregeln für Momente:

$$\begin{aligned} \mathbb{E}[X + Y] &= \mathbb{E}[X] + \mathbb{E}[Y] & \forall X, Y \\ \mathbb{E}[X \cdot Y] &= \mathbb{E}[X] \cdot \mathbb{E}[Y] & \forall X, Y \text{ unabhängig} \\ \text{Var}[X + Y] &= \text{Var}[X] + \text{Var}[Y] & \forall X, Y \text{ unabhängig} \\ \text{Var}[X \cdot Y] &\neq \text{Var}[X] \cdot \text{Var}[Y] & \text{muss ausgerechnet werden} \end{aligned}$$

Multiplikativität des Erwartungswerts: Für unabhängige Zufallsvariablen  $X_1, \dots, X_n$  gilt:

$$\mathbb{E}[X_1 \cdot \dots \cdot X_n] = \mathbb{E}[X_1] \cdot \dots \cdot \mathbb{E}[X_n]$$

Satz: Für unabhängige Zufallsvariablen  $X_1, \dots, X_n$  und  $X := X_1 + \dots + X_n$  gilt:

$$\text{Var}[X] = \text{Var}[X_1] + \dots + \text{Var}[X_n]$$

## 2.3 Abschätzen von Wahrscheinlichkeiten

**Waldsche Identität:** N und X seien zwei unabhängige Zufallsvariable, wobei für den Wertebereich von N gelte:  $W_N \subseteq \mathbb{N}$ . Weiter sei:

$$Z := \sum_{i=1}^N X_i$$

wobei  $X_1, X_2, \dots$  unabhängige Kopien von X seien. Dann gilt:

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

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

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t} \quad \forall X \geq 0, \forall t > 0$$

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

$$\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\text{Var}[X]}{t^2} \quad \forall X, \forall t > 0$$

insbesondere

$$\Pr[X \geq \mathbb{E}[X] + t] \leq \frac{\text{Var}[X]}{t^2}$$

**Ungleichung von Chernoff:** Die Obereschranke von Chernoff liefert ein viel kleineres Fehler als dass von Chebyshev

$$\Pr[X \geq (1 + \delta)\mathbb{E}[X]] \leq e^{-\frac{1}{3}\delta^2\mathbb{E}[X]} \quad \forall X \sim \text{Bin}(n, p), \forall 0 < \delta < 1$$

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

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

## 2.4 Randomisierte Algorithmen:

**Target-Shooting:** Gegeben zwei endliche Mengen  $S \subseteq U$  bestimme  $|S|/|U|$ . Annahmen:

- Wir können ein Element aus U effizient zufällig gleichverteilt wählen
- es gibt eine effizient berechenbare Funktion

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

---

**TARGET-SHOOTING**  
1: Wähle  $u_1, \dots, u_N \in U$  zufällig, gleichverteilt und unabhängig  
2: return  $N^{-1} \cdot \sum_{i=1}^N \mathbb{I}_S(u_i)$

---

Notation:  $Y_i := \mathbb{I}_S(U_i)$  für alle  $i=1, \dots, N$

$Y_1, \dots, Y_N$  unabhängige Bernoulli-Variablen mit  $\Pr[Y_i = 1] = |S|/|U|$

$$Y := \frac{1}{N} \sum_{i=1}^N Y_i = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_S(u_i)$$

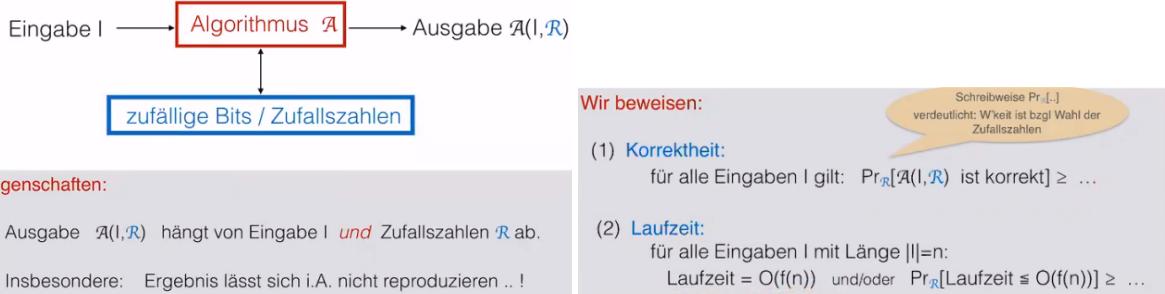
Dann gilt:  $\mathbb{E}[Y] = |S|/|U|$  ... unabhängig von der Wahl von N

$$\text{Var}[Y] = \frac{1}{N} \left( \frac{|S|}{|U|} - \left( \frac{|S|}{|U|} \right)^2 \right)$$

Seien  $\delta, \epsilon > 0$ . Falls  $N \geq 3 \frac{|U|}{|S|} \cdot \epsilon^{-2} \cdot \log(2/\delta)$  so ist die Ausgabe des Algorithmus TARGET-SHOOTING mit Wahrscheinlichkeit mindestens  $1 - \delta$  im Intervall  $[(1 - \epsilon) \frac{|S|}{|U|}, (1 + \epsilon) \frac{|S|}{|U|}]$ . Oder äquivalent,

$$\Pr[|Y - \frac{|S|}{|U|}| > \epsilon \frac{|S|}{|U|}] = \Pr[|Y - \mathbb{E}[Y]| > \epsilon \mathbb{E}[Y]] = \Pr[|NY - \mathbb{E}[NY]| > \epsilon \mathbb{E}[NY]] \leq \delta$$

Da  $NY = Y_1 + \dots + Y_N$  eine Summe von N unabhängigen Bernoulli-Variablen ist, kann dies nun leicht mit Hilfe der Chernoff Schranken hergeleitet werden.



Zufallsalgorithmen werden deterministisch indem wir den Zufallsgenerator einen Startwert geben.

**Las-Vegas Algorithmen:** Geben nie eine falsche Antwort, aber manchmal keine Antwort ( Ausgabe = ???).

$$\text{Ziel: } \Pr[\text{Antwort} = \text{???}] = \text{winzig}$$

**Monte-Carlo Algorithmen:** Geben immer eine Antwort aber manchmal auch eine falsche Antwort.

$$\text{Ziel: } \Pr[\text{AntwortFalsch}] = \text{winzig}$$

**Entscheidungsproblem:** Problem wo man entscheidet zwischen Ja oder Nein.

**Fehlerkorrektur:** Für Las Vegas und Monte Carlo Algorithmen kann man durch wiederholungen den fehlere kleiner machen. Beispiel: Las Vegas

$$\Pr[\text{Antwort} = \text{???}] \leq 1 - \epsilon \\ \Rightarrow N := \epsilon^{-1} \ln(\delta^{-1}) \text{ Wiederholungen reduzieren den Fehler auf } \Pr[\text{Antwortimmer} \text{???}] \leq (1 - \epsilon)^N \leq \delta$$

In der Praxis ruft man Las-Vegas-Algorithmen meist einfach so lange auf, bis sie eine Antwort geben.  
Beispiel Monte Carlo Algorithmen für Entscheidungsprobleme: Einseitiger Fehler:

$$\Pr[\text{Antwortfalsch} | \text{Antwort} = \text{Ja}] = 0 \\ \Pr[\text{Antwortfalsch} | \text{Antwort} = \text{Nein}] \leq 1 - \epsilon \\ \Rightarrow \epsilon^{-1} \ln(\delta^{-1}) \text{ Wiederholungen reduzieren Fehler auf } \Pr[\text{Antwort falsch}] \leq \delta$$

Der Algorithmus liefert Ja, wenn ein Aufruf Ja ausgibt und Nein, wenn alle Wiederholungen Nein ausgeben.  
Zweiseitiger Fehler (Algorithmus kann falsch sein wenn es die Antwort Ja oder Nein ausgibt):

$$\Pr[\text{Antwortfalsch}] \leq \frac{1}{2} - \epsilon \Rightarrow 12(\frac{1}{2} - \epsilon) \ln(\delta^{-1}) \text{ Wiederholungen reduzieren Fehler auf } \Pr[\text{Antwortfalsch}] \leq \delta$$

Der Algorithmus liefert die Mehrheit der gesehenen Antworten (i.e Ja falls mehr ausgaben Ja waren sonst Nein)

**Satz 2.72: (Las Vegas Algorithmus)** Sei  $A$  ein randomisierter Algorithmus, der nie eine falsche Antwort gibt, aber zuweilen '???' ausgibt, wobei

$$\Pr[A(I)\text{korrekt}] \geq \epsilon \quad \forall I$$

Dann gilt für alle  $\delta > 0$ : bezeichnet man mit  $A_\delta$  den Algorithmus der  $A$  solange aufruft bis entweder ein Wert verschieden von '???' ausgegeben wird (und  $A_\delta$  diesen Wert dann ebenfalls ausgibt) oder bis  $N = \epsilon^{-1} \ln(\delta^{-1})$  mal '???' ausgegeben wurde (und  $A_\delta$  dann ebenfalls '???' ausgibt), so gilt für den Algorithmus  $A_\delta$ , dass

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

**Satz 2.74: (Monte-Carlo Algorithmus Einseitiger Fehler)** Sei  $A$  ein randomisierter Algorithmus, der immer eine der beiden Antworten Ja oder Nein ausgibt, wobei

$$\Pr[A(I) = \text{Ja}] = 1 \quad \text{falls } I \text{ eine Ja-Instanz ist} \\ \Pr[A(I) = \text{Nein}] \geq \epsilon \quad \text{falls } I \text{ eine Nein-Instanz ist}$$

Dann gilt für alle  $\delta > 0$ : bezeichnet man mit  $A_\delta$  den Algorithmus der  $A$  solange aufruft bis entweder der Wert Nein ausgegeben wird (und  $A_\delta$  dann ebenfalls Nein ausgibt) oder bis  $N = \epsilon^{-1} \ln(\delta^{-1})$  mal Ja ausgegeben wurde (und  $A_\delta$  dann ebenfalls Ja ausgibt), so gilt für alle Instanzen  $I$

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

**Satz 2.75: (Monte-Carlo Algorithmus Beidseitiger Fehler)** Sei  $\epsilon > 0$  und  $A$  ein randomisierter Algorithmus, der immer eine der beiden Antworten Ja oder Nein ausgibt, wobei

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

Dann gilt für alle  $\delta > 0$ : bezeichnet man mit  $A_\delta$  den Algorithmus, der  $N = 4\epsilon^{-2} \ln \delta^{-1}$  unabhängige Aufrufe von  $A$  macht und dann die Mehrheit der erhaltenen Antworten ausgibt, so gilt für den Algorithmus  $A_\delta$ , dass

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

**Satz 2.76:** Sei  $\epsilon > 0$  und A ein randomisierter Algorithmus für ein Maximierungsproblem, wobei gelte:

$$Pr[A(I) \geq f(I)] \geq \epsilon$$

Dann gilt für all  $\delta > 0$ : bezeichnet man mit  $A_\delta$  den Algorithmus, der  $N = \epsilon^{-1} \ln(\delta^{-1})$  unabhängige Aufrufe von A macht und die beste der erhaltenen Antworten ausgibt, so gilt für den Algorithmus  $A_\delta$ , dass

$$Pr[A_\delta(I) \geq f(I)] \geq 1 - \delta$$

(Für Minimierungsprobleme gilt eine analoge Aussage wenn wir  $\geq f(I)$  durch  $\leq f(I)$  ersetzen)

### Abschätzungen für Binomialverteilung:

- **Markov:**  $Pr[X \geq C \cdot \mathbb{E}[X]] \leq \frac{1}{C}$
- **Chebyshev:**  $Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\text{Var}[X]}{t^2}$   
z.B  $t = c \cdot \mathbb{E}[X]$   
 $\Rightarrow Pr[X \leq (1 - c)\mathbb{E}[X] \text{ oder } X \geq (1 + c)\mathbb{E}[X]] \leq \frac{n \cdot p \cdot (1-p)}{(c \cdot np)^2} = \frac{1}{cn} \cdot \frac{1-p}{p}$   
 Dieses term = 1 für  $p = \frac{1}{2}$  aber gross für  $p \approx 0$  bzw  $p \approx 1 \rightarrow$  Man kann Chebyshev anwenden wenn der Binomialverteilung ein vernünftig grosses n hat und wenn p von 0 und 1 weg beschränkt ist.
- **Chernoff:**  $Pr[|X - \mathbb{E}[X]| \geq \delta \mathbb{E}[X]] \leq e^{-\frac{1}{2}\delta^2 \mathbb{E}[X]} + e^{-\frac{1}{3}\delta^2 \mathbb{E}[X]} \leq 2 \cdot e^{-\frac{1}{2}\delta^2 \mathbb{E}[X]}$   
 wir wählen  $\delta = c \Rightarrow Pr[X \leq (1 - c)\mathbb{E}[X] \text{ oder } X \geq (1 + c)\mathbb{E}[X]] \leq 2 \cdot e^{-\frac{1}{2}c^2 np}$

## 2.5 Sortieren

---

```
QUICKSORT(A, ℓ, r)
1: if ℓ < r then
2:   p ← Uniform({ℓ, ℓ + 1, …, r})           ▷ wähle Pivotelement zufällig
3:   t ← PARTITION(A, ℓ, r, p)
4:   QUICKSORT(A, ℓ, t − 1)
5:   QUICKSORT(A, t + 1, r)
```

---

### Satz:

- Quicksort bestimmt *immer* das richtige Ergebnis
- $E[\text{Laufzeit}] = O(n \ln n)$

**Quicksort:** Da Quicksort immer das richtige Ergebnis gibt ist es ein Las-Vegas Algorithmus. Wir wollen zeigen dass  $E[\text{Laufzeit}] = O(n \cdot \ln(n))$

$t_n := E[\text{zeit beim Sortieren von } n \text{ Elementen}]$

$T_n := \# \text{Vergleiche beim sortieren von } n \text{ Elementen}$

$$T_n = n - 1 (\text{vergleiche mit Pivot element}) + \sum_{i=1}^n Pr[\text{pivot ist der } i \text{ kleinste element}] \cdot (T_{i-1} + T_{n-i})$$

$$\Rightarrow T_n = n - 1 + \frac{1}{n} \sum_{i=1}^n (t_{i-1} + t_{n-i}) \text{ Durch linearität der Erwartungswert kann T durch t ersetzt werden}$$

$$\Rightarrow t_n = n - 1 + \frac{1}{n} \sum_{i=1}^n (t_{i-1} + t_{n-i}) \rightarrow \text{Löse Rekursion} \Rightarrow O(n \cdot \ln(n))$$

**Selektieren:** Aufgabe: finde das k-te kleinste Element aus einem unsortierten Array

**Selektieren mit QuickSelect:** . Wenn dass Array A sortiert ist, gibt k den platz der k kleinste elements an. Wir

---

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

---

### Satz:

- QuickSelect bestimmt *immer* das richtige Ergebnis
- $E[\text{Laufzeit}] = O(n)$

wählen ein Pivot p und wir partitionieren dass array so dass alle elemente kleiner als p links von p sind und alle elemente die grösser sind rechts von p. Danach vergleichen wir den index von p und k. Falls k kleiner als p ist, wissen wir dass unser gesuchte element sich im linke teilarray befindet sonst im rechten. (falls index von p = index von k ist p unser gesuchtes element).

*Beweis*  $\mathbb{E}[\text{Laufzeit}] = \mathcal{O}(n)$

Die Laufzeit ist proportional zur anzahl vergleiche. In Jeder schritt wird der funktion auf der linke seite oder rechte seite.

$$\Rightarrow T := \#\text{vergleiche} = \sum_{i=0}^N (r_i - l_i)$$

Wir definieren eine neue Zufallsvariable  $N_j := \#\{i \mid (\frac{3}{4})^j r_i - l_i \leq (\frac{3}{4})^{j-1} n\}$

$$\Rightarrow T \leq \sum N_j \cdot (\frac{3}{4})^{j-1} \cdot n$$

$$\Rightarrow \mathbb{E}[T] \leq \sum \mathbb{E}[N_j] \cdot (\frac{3}{4})^{j-1} \cdot n$$

Beobachtung: Wir werden mit Wahrscheinlichkeit  $\geq \frac{1}{2}$  mind  $\frac{1}{4}$  der Elemente wegerfen.  $(r_i - l_i \rightarrow r_{i+1} - l_{i+1}$ , sodass  $r_{i+1} - l_{i+1} \leq \frac{3}{4}(r_i - l_i)$

$$\Rightarrow \mathbb{E}[N_j] \leq \mathbb{E}[\text{Geo}(\frac{1}{2})] = 2$$

$$\Rightarrow \mathbb{E}[T] \leq 2n \sum (\frac{3}{4})^{j-1} \leq 2n \frac{1}{1-\frac{3}{4}} = 8n \Rightarrow \text{Laufzeit} = \mathcal{O}(n)$$

## 2.6 Primzahltest

Primzahlfunktion: Gibt an wie viele primzahlen bis zu einer zahl x es gibt.

$$\pi(x) := |\{n \in \mathbb{N} \mid n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$$

Grösster gemeinsamer Teiler: Für  $m, n \in \mathbb{Z}$  sei  $\text{ggT}(m, n)$  der grösste gemeinsame Teiler von m und n. Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden ( $\mathcal{O}((\log nm)^3)$ ). Natürlich gilt:

$$\text{ggT}(a, n) > 1 \text{ für } a \in [n-1] \Rightarrow n \text{ nicht prim}$$

---

### Euklid-Primzahltest( $n$ )

---

- 1: Wähle  $a \in [n-1]$ , zufällig gleichverteilt
  - 2: if  $\text{ggT}(a, n) > 1$  then return 'keine Primzahl'
  - 3: else return 'Primzahl'
- 

Bemerkungen:

- Ausgabe "keine Primzahl" ist immer richtig
- Falls n nicht prim: Falsche Ausgabe "Primzahl" mit W'keit  $\frac{|\mathbb{Z}_n^*|}{n-1}$

### Multiplikative Gruppe mod n:

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \text{ggT}(a, n) = 1\}$$

### Eulersche Phi-Funktion:

$$\phi(n) := |\mathbb{Z}_n^*|$$

Falls n prim:  $\mathbb{Z}_n^* = [n-1]$  und  $\phi(n) = n-1$

Falls  $n = p^2$  p prim:  $\phi(n) = p(p-1) = n - \sqrt{n}$  ( $\Rightarrow \frac{|\mathbb{Z}_n^*|}{n-1} \approx 1 - \frac{1}{\sqrt{n}}$  D.h die wahrscheinlichkeit dass wir in Irre geführt werden ist gross) Bemerkungen zu Fermats Primzahltest:

## Erinnerung Gruppentheorie

---

- ▶ Satz von Lagrange  $\Rightarrow$  Ist  $H \leq G$  ( $H$  Untergruppe von  $G$ ), dann ist  $|H|$  ein Teiler von  $|G|$ .
- ▶  $\text{ord}(a) := \min\{i \in \mathbb{N} \mid a^i = 1\}$ , die Ordnung der von  $a$  erzeugten Untergruppe  $\langle a \rangle := \{a^0 = 1, a^1, a^2, \dots\}$ .
- ▶  $\text{ord}(a)$  ist Teiler von  $|G|$ . Folglich gilt

$$a^{|G|} = a^{\text{ord}(a) \frac{|G|}{\text{ord}(a)}} = 1^{\frac{|G|}{\text{ord}(a)}} = 1.$$

- ▶ In  $\mathbb{Z}_n^*$ :  $a^{\varphi(n)} = 1$  für alle  $a \in \mathbb{Z}_n^*$  (weil  $\varphi(n) = |\mathbb{Z}_n^*|$ )
- ▶ In  $\mathbb{Z}_n^*$ , n prim:  $a^{n-1} = 1$  für alle  $a \in \mathbb{Z}_n^*$  (weil  $\varphi(n) = n-1$ ).

### Satz (Kleiner fermatscher Satz)

Ist  $n \in \mathbb{N}$  prim, so gilt für alle Zahlen  $a \in [n-1]$

$$a^{n-1} \equiv 1 \pmod{n} \quad (\text{bzw. } a^{n-1} = 1 \text{ in } \mathbb{Z}_n^*).$$

---

**Fermat-Primzahltest( $n$ )**

---

```

1: Wähle  $a \in [n - 1]$ , zufällig gleichverteilt
2: if  $\text{ggT}(a, n) > 1$  oder  $a^{n-1} \not\equiv 1 \pmod{n}$  then
3:   return 'keine Primzahl'
4: else
5:   return 'Primzahl'

```

---

- $a^{n-1} \neq 1 \pmod{n}$  kann schnell mit binärer Exponentiation berechnet werden
- Die Ausgabe keine Primzahl ist immer richtig
- Die Ausgabe Primzahl ist falsch mit W'keit  $\frac{|PB_n|}{n-1} < \frac{1}{2}$  es sei denn  $n$  ist eine Carmichael-Zahl
- Wir machen einen Fehler, falls  $n$  nicht prim,  $\text{ggT}(a,n) = 1$  und  $a^{n-1} \equiv 1 \pmod{n}$ . Ein solches  $a$  nennen wir **Pseudoprime Zahlbasis** von  $n$ .

#### Pseudoprime Zahlbasen und Carmichael-Zahlen:

$$PB_n := \{a \in [n-1] | \text{ggT}(a, n) = 1 \text{ und } a^{n-1} \equiv_n 1\} = \{a \in \mathbb{Z}_n^* | a^{n-1} \equiv_n 1\}$$

$n$  heisst Carmichael-Zahl, falls  $n$  nicht prim ist und  $PB_n = \mathbb{Z}_n^*$   
 $PB_n$  ist eine Untergruppe von  $\mathbb{Z}_n^*$ , weil

$$a^{n-1} \equiv_n 1 \wedge b^{n-1} \equiv_n 1 \Rightarrow (ab \bmod n)^{n-1} \equiv_n 1$$

Folglich, wenn  $PB_n \neq \mathbb{Z}_n^*$  (d.h  $n$  ist nicht Carmichael), dann ist  $|PB_n|$  ein echter Teiler von  $\mathbb{Z}_n^*$  und  $|PB_n| \leq \frac{\phi(n)}{2} < \frac{n-1}{2}$

#### Zertifikate für nicht prim:

- triviales Zertifikat, falls  $a > 2$  und  $a|n$ : Anteil  $> \frac{1}{n}$
- Euklid Zertifikat, falls  $\text{ggT}(a,n) > 1$ : Anteil  $> \frac{1}{\sqrt{n}}$
- Fermat Zertifikat, falls  $a^{n-1} \bmod n \neq 1$ : Anteil  $> \frac{1}{2}$ , ausser  $n$  ist Carmichael
- Miller-Rabin Zertifikat, falls

$$(a^d, a^{2d}, \dots, a^{2^k d}) \neq \begin{cases} (1, 1, 1, \dots, 1) & (\iff a^d \bmod n = 1) \\ (\dots, \dots, n-1, 1, \dots, 1) & (\iff \exists i : a^{2^i d} \bmod n = n-1) \end{cases}$$

Anteil  $> \frac{3}{4}$  ( $n$  ungerade)

**Miller-Rabin Test:** See below

#### Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \dots, n-1\}$$

Für  $n$  prim, ist  $(\mathbb{Z}_n, +, \cdot)$ , ' $\cdot$ ' und ' $+$ ' mod  $n$ , ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

In einem Körper hat die Gleichung  $x^2 = 1$ , zwei Lösungen:

$$x = 1 \text{ und } x = -1 = n-1$$

Ist  $n > 2$  prim und  $a \in [n-1]$ , dann gilt

- ▶  $a^{n-1} = 1$ , (kleiner fermatscher Satz)
- ▶  $a^{\frac{n-1}{2}} \in \{1, n-1\}$ , (Körpereigenschaft oben)
- ▶ falls  $a^{\frac{n-1}{2}} = 1$  und  $\frac{n-1}{2}$  gerade, dann  $a^{\frac{n-1}{4}} \in \{1, n-1\}$ ,
- ▶ ... bis  $a^{\frac{n-1}{2^i}} = n-1$  oder  $\frac{n-1}{2^i}$  ungerade.

---

**Miller-Rabin-Primzahltest( $n$ )** (n > 1, ungerade!)

---

```

1:  $d, k \in \mathbb{N}$ , mit  $n-1 = 2^k d$ ,  $d$  ungerade
2: Wähle  $a \in [n-1]$ , zufällig gleichverteilt
3: if  $a^d \bmod n \neq 1$  und  $\nexists i < k : a^{2^i d} \bmod n = n-1$  then
4:   return 'keine Primzahl'
5: else
6:   return 'Primzahl'

```

---

- ▶ Die Ausgabe 'keine Primzahl' ist immer richtig.
- ▶ Die Ausgabe 'Primzahl' ist falsch mit W'keit  $\leq \frac{1}{4}$ .

## 2.7 Duplikate Finden in grossen Datensätzen

**Problemstellung:**  $S = (s_1, s_2, \dots, s_n)$ , Folge von  $n$  Elementen (Datensatz).  $(i,j)$ ,  $1 \leq i < j \leq n$  heisst Duplikat in  $S$ , falls  $s_i = s_j$ . Beispiel: Gegeben ein Datensatz  $S$ , finde alle Duplikate.

$$S = (A \ C \ B \ Z \ C \ B \ C)$$

Duplikate  $\text{Dupl}(S) = \{(2, 5), (6, 7), (5, 7), (3, 6)\}$  (4 Duplikate)

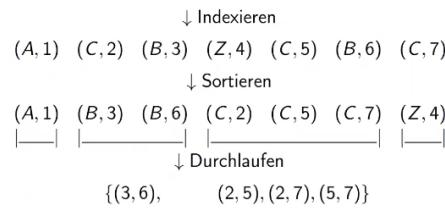
weitere Annahme:

Die Elemente in  $S$  sind gross d.h Speicherzugriffe und Vergleiche sind teuer.

## Lösung:

- Erste Lösung:

Sortiere die Folge  $((s_1, 1), (s_2, 2), \dots, (s_n, n))$  nach erster Koordinate. Dann sind die Duplikate leicht durch einen Durchlauf der sortierten Folge zu finden.



- (i) Sortieren:  $O(n \log n)$  Vergleiche und Speicherzugriffe
  - (ii) Durchlaufen:  $O(n + |\text{Dupl}(\mathcal{S})|)$

- Zweite Lösung: Hash Funktion

Seien die Elemente in  $S$  aus einem Universum  $U$ . Wir verwenden eine Hashfunktion  $h: U \rightarrow [m]$  und setzen voraus:

- $h$  ist effizient berechenbar
  - $h$  verhält sich wie eine Zufallsfunktion d.h.

$$\forall u \in U \ \forall i \in [m] : Pr[h(u) = i] = \frac{1}{m}$$

Bei Hashfunktionen wird  $m$  viel kleiner als  $|U|$  gewählt d.h es gibt eine Komprimierung. Jedes  $h(s_i)$  ist zufällig gleichverteilt in  $[m]$ , aber

$$s_i = s_j \Rightarrow h(s_i) = h(s_j)$$

Es kann sein, dass zwei verschiedene Werte auf den gleichen Zahl abgebildet werden, das nennt man **Kollision**.

$A$ $\underbrace{(h(A), 1)}_{31}$	$C$ $\underbrace{(h(C), 2)}_{27}$	$B$ $\underbrace{(h(B), 3)}_{12}$	$Z$ $\underbrace{(h(Z), 4)}_{12}$	$C$ $\underbrace{(h(C), 5)}_{27}$	$B$ $\underbrace{(h(B), 6)}_{12}$	$C$ $\underbrace{(h(C), 7)}_{27}$	<p><b>Kollisionen</b> sind die <i>neuen</i> (unerwünschten) Duplikate im Hashmap, d.h. die Paare <math>(i, j)</math>, <math>1 \leq i &lt; j \leq n</math>, mit <math>s_i \neq s_j</math> und <math>h(s_i) = h(s_j)</math>. Sei <math>K_{i,j}</math> die Bernoulli Variable mit  <math>K_{i,j} = 1 \Leftrightarrow (i, j)</math> ist eine Kollision .</p>
			$\downarrow$ Sortieren				$K_{i,j} = 1 \Leftrightarrow (i, j)$ ist eine Kollision .
$(12, 3)$ $\rule{0.5cm}{0.4pt}$	$(12, 4)$ $\rule{0.5cm}{0.4pt}$	$(12, 6)$ $\rule{0.5cm}{0.4pt}$	$(27, 2)$ $\rule{0.5cm}{0.4pt}$	$(27, 5)$ $\rule{0.5cm}{0.4pt}$	$(27, 7)$ $\rule{0.5cm}{0.4pt}$	$(31, 1)$ $\rule{0.5cm}{0.4pt}$	$\Pr[K_{i,j} = 1] = \begin{cases} 1/m & \text{falls } s_i \neq s_j, \\ 0 & \text{sonst,} \end{cases}$ und daher $\mathbb{E}[K_{i,j}] \leq \frac{1}{m}$

Beim Durchlaufen müssen eventuelle Kollisionen (hier z.B. wegen  $h(B) = h(Z)$ ) im Hashmap noch geprüft werden.

$$\mathbb{E}[\#\text{Kollisionen}] \leq \binom{n}{2} \frac{1}{m} < 1 \quad \text{für } m = n^2$$

Mit  $m \equiv n^2$  ist der Mehraufwand durch Kollisionen konstant.

$$\text{Zeit: } \underbrace{O(n)}_{\text{Hashmap}} + \underbrace{O(n \log n)}_{\text{Sortieren von } \lceil 2 \log n \rceil \text{-Bit Zahlen}} + \underbrace{O(n + |\text{Dupl}(\mathcal{S})|)}_{\text{Durchlaufen}}$$

zusätzlicher Speicher:  $O(\underbrace{n \log n}_{\text{Indices}} + \underbrace{n \log m}_{\text{Hashwerte}}) \stackrel{m=n^2}{=} O(n \log n)$

- Dritter Lösung: Bloom-Filter Wir wählen  $m, k \in \mathbb{N}$  und  $k$  Hashfunktionen (zufällig)

$$h_i : U \rightarrow [m] \text{ i = 1,...,k}$$

d.h jede  $s_i$  in S wird ein Hash-Vektor zugeordnet. Wir benutzen auch ein boolsches Feld M[1..m] welche am anfangs all Einträge auf 0 gesetzt werden. Wir gehen jetzt die elemente von S durch und setzen M[i] auf 1 falls der indize in der Hashvektor enthalten ist und M[i] vorher 0 war. Falls alle M[i] = 1 für diese hashvektor waren, dann nehmen wir an dass wir dieses element schon einmal gesehen haben und fügen den indizes in  $\mathcal{L}$  eine Liste der Wiederholungen ein. Wir interessieren uns für die Wahrscheinlichkeit einer falsche eintrag in  $\mathcal{L}$

Betrachten wir  $s_i$  mit Hash-Vektor  $(x_1, \dots, x_k)$ .

$$X_i = 1 \Leftrightarrow \begin{cases} s_i \text{ tritt in } (s_1, \dots, s_{i-1}) \text{ nicht auf und} \\ M[x_1] = \dots = M[x_k] = 1 \text{ vor dem Abarbeiten von } s_i. \end{cases}$$

d.h.  $i$  wird ein falscher  $\mathcal{L}$ -Eintrag.

Nach den ersten  $i-1$  Daten,  $\ell \leq i-1$  verschieden, gilt für  $x \in [m]$ :

$$\Pr[M[x] = 0] = \left(1 - \frac{1}{m}\right)^{k\ell} \geq \left(1 - \frac{1}{m}\right)^{k(i-1)}$$

$$\Pr[X_i = 1] \leq \Pr[\forall j : M[x_j] = 1] \stackrel{(*)}{\leq} \left(1 - \left(1 - \frac{1}{m}\right)^{k(i-1)}\right)^k$$

(\*) nutzt "negative Korrelation", hier nicht behandelt.

$$\mathbb{E}[\#\text{falsche Einträge in } \mathcal{L}] = \sum_{i=1}^n \mathbb{E}[X_i] \leq n \cdot \left(1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}\right)^k$$

Dieser Wert ist  $O(1)$  für  $k = \ln n$  und  $m = n \ln n$ .

$k$  und  $m$  gross  $\Rightarrow$  #Falsche Einträge klein.

Laufzeit:  $kn$  Hashfunktionsberechnungen.  $k$  gross  $\Rightarrow$  langsamer.

Zusätzlicher Speicher:  $m$ .  $m$  gross  $\Rightarrow$  mehr Speicher.

## 2.8 Lange Pfade

**LONG-PATH Problem:** Gegeben  $(G, B)$ ,  $G$  eine Graph und  $B \in \mathbb{N}_0$  stelle fest ob es einen Pfad der Länge  $B$  in  $G$  gibt. Ein Pfad der Länge  $l$  in einem Graph  $G = (V, E)$  ist eine Folge von paarweise verschiedenen Knoten

$$\langle v_0, v_1, \dots, v_l \rangle \text{ mit } \{v_{i-l}, v_i\} \in E \text{ für } i = 1, \dots, l$$

d.h. es liegen  $l+1$  Knoten auf einem Pfad der Länge  $l$ .

Wenn es einen Polynomialzeit-Algorithmus für LONG-PATH gibt, dann gibt es einen Polynomialzeit-Algorithmus für das Hamiltokreis Problem was NP-Vollständig ist d.h. wir vermuten es gibt keinen Polynomialzeit algorithmus für den LONG-PATH. In der Regel sucht man kurze lange pfade i.e  $B = \mathcal{O}(\log n)$ . Wir werden folgende hilfsmittel brauchen:

- ▶  $[n] := \{1, 2, \dots, n\}$ ;  $[n]^k$  ist die Menge der Folgen über  $[n]$  der Länge  $k$ , es gilt  $|[n]^k| = n^k$ ;  $\binom{[n]}{k}$  ist die Menge der  $k$ -elementigen Teilmengen von  $[n]$  und es gilt  $\left|\binom{[n]}{k}\right| = \binom{n}{k}$ .
- ▶ Für jeden Graph  $G = (V, E)$  gilt  $\sum_{v \in V} \deg(v) = 2|E|$ .
- ▶  $k$  Knoten man mit  $[k]$  auf genau  $k^k$  Arten färben,  $k!$  dieser Färbungen nutzen jede Farbe genau einmal.
- ▶ Für  $c, n \in \mathbb{R}^+$ , gilt  $c^{\log n} = n^{\log c}$ . Also, z.B.  $2^{\log n} = n^{\log 2} = n$  und  $2^{O(\log n)} = n^{O(1)}$  ist polynomiell in  $n$ .
- ▶ Für  $n \in \mathbb{N}_0$  gilt  $\sum_{i=0}^n \binom{n}{i} = 2^n$ , eine Anwendung des Binomialatzes:  $\sum_{i=0}^n \binom{n}{i} x^i y^{n-i} = (x+y)^n$ .
- ▶ Für  $n \in \mathbb{N}_0$  gilt  $\frac{n!}{e^n} \geq e^{-n}$ . Potenzreihenentwicklung der Exponentialfunktion:
$$e^n = \sum_{i=0}^{\infty} \frac{n^i}{i!} \geq \frac{n^n}{n!} \quad (\text{der Term in der Summe für } i = n)$$
- ▶ Wiederholt man ein Experiment mit Erfolgswahrscheinlichkeit  $p$  solange bis man Erfolg hat, dann ist der Erwartungswert der Anzahl der Versuche  $\frac{1}{p}$  (geometrische Verteilung  $\text{Geo}(p)$ ).
- ▶ Dynamische Programmierung, ...

**Bunte Pfade:** Ein Pfad heisst bunt, falls alle seine Knoten verschiedene Farben haben.

**COLORFUL-PATH Problem:** Gegeben  $(G, \gamma)$ ,  $G = (V, E)$  ein Graph und  $\gamma : V \rightarrow [k]$ , stelle fest ob es einen bunten Pfad der Länge  $k-1$  (d.h mit  $k$  knoten) in  $G$  gibt. ( $G$  muss nicht notwendigerweise eine gültige Färbung haben d.h dass beliebig viele knoten dürfen die gleiche Farbe besitzen) Ein bunter Pfad der Länge  $i$  zu  $v$  besteht aus einem  $\gamma(v)$ -freien

Wir fixieren einen Knoten  $v$  und  $i \in \mathbb{N}_0$ . Wir betrachten nur bunte Pfade der Länge  $i$  (mit  $i+1$  Knoten), die in  $v$  enden, und sammeln alle Farbmengen, die auf solchen Pfdaden auftreten können.

$$P_i(v) := \{S \in \binom{[k]}{i+1} \mid \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$

- ▶  $\forall S \in P_i(v) : \gamma(v) \in S$ .
- ▶  $P_0(v) = \{\{\gamma(v)\}\}$ .
- ▶  $P_1(v) = \{\{\gamma(x), \gamma(v)\} \mid x \in N(v), \gamma(x) \neq \gamma(v)\}$ .  
 $N(x) :=$  Menge der Nachbarn von  $x$ .
- ▶  $\exists$  bunter Pfad mit  $k$  Knoten  $\Leftrightarrow \bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$ .

bunten Pfad der Länge  $i-1$  zu einem Nachbarn  $x$  von  $v$  plus dem Schritt zu  $v$ .

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

Wir berechnen alle  $P_i(v), v \in V$  mit den  $P_{i-1}(v), v \in V$  gegeben:

Regenbogen( $G, \gamma$ )	$G$ Graph, $\gamma$ $k$ -Färbung
1: <b>for all</b> $v \in V$ <b>do</b>	
2: <b>for all</b> $x \in N(v)$ <b>do</b>	
3: <b>for all</b> $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ <b>do</b>	$ P_{i-1}(x)  \cdot i$
4: $P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$	

Mit  $m := |E|$  und wegen  $|P_{i-1}(x)| \leq \binom{k}{i}$  ist der Zeitaufwand

$$O\left(\sum_{v \in V}^{\deg(v)} \binom{k}{i} \cdot i\right) = O\left(\binom{k}{i} \cdot i \cdot m\right).$$

$$O\left(|V| + \sum_{i=1}^{k-1} \left( \binom{k}{i} \cdot i \cdot m \right) + |V|\right) = O(2^k km).$$

wegen  $\sum_{i=1}^{k-1} \binom{k}{i} \leq \sum_{i=0}^k \binom{k}{i} = 2^k$

D.h. für  $k = \log n$ : Laufzeit  $O(mn \log n)$ ,  
 $k = O(\log n)$ : Laufzeit  $O(\text{poly}(n))$ .

Damit kann man den LONG-PATH Problem lösen:

Setze  $k := B+1$ , färbe  $G$  zufällig mit  $k$  Farben, und suche einen bunten Pfad mit  $k$  Knoten. Angenommen  $G$  hat einen bunten Pfad  $P$  mit  $k$  Knoten. Dann ist die W'keit  $p_{\text{perf}}$ , dass es in  $(G, \gamma)$  einen bunten Pfad gibt:

$$p_{\text{perf}} := \Pr[\exists \text{ bunter Pfad mit } k \text{ Knoten}] \geq \Pr[P \text{ ist bunt}] = \frac{k!}{k^k} \geq e^{-k}$$

Daraus folgt die erwartete Anzahl Versuche, bis man einen bunten Pfad findet  $\leq e^k (Geo(e^{-k}))$

Der Laufzeit für einen Versuch ist in:  $O(2^k km) \quad p_{\text{perf}} \geq e^{-k}$

Macht man  $\lceil \lambda e^k \rceil$  Versuche:

- Laufzeit:  $\mathcal{O}(\lambda(2e)^k km)$

- W'keit dass der Algorithmus den Pfad nicht findet ist höchstens

$$\leq (1 - e^{-k})^{\lceil \lambda e^k \rceil} \leq (e^{-e^{-k}})^{\lceil \lambda e^k \rceil} \leq e^{-\lambda} \text{ (wobei wir beim ersten schritt } 1+x \leq e^x \text{ angewendet haben)}$$

# Chapter 3

## Flüsse in Netzwerken

### 3.1 Einführung & Modellierung

**MaxFlow Problem:** Gegeben ein Netzwerk, finde einen Fluss grössten Werts.

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

- $(V, A)$  ist ein gerichteter Graph (ohne Schleifen)
- $s \in V$  die Quelle
- $t \in V \setminus \{s\}$ , die Senke
- $c : A \rightarrow \mathbb{R}_0^+$  die Kapazitätsfunktion

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

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

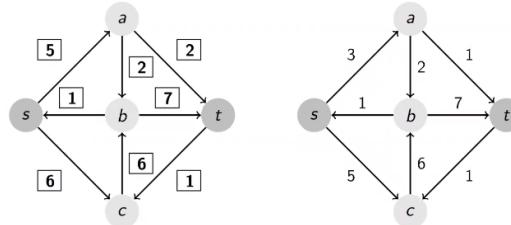
$$\sum_{u \in V : (u,v) \in A} f(u,v) = \sum_{u \in V : (v,u) \in A} f(v,u)$$

i.e die Summe der Flüsse in einen Knoten ist gleich die Flüsse aus dem Knoten.

Die Flüssen sind gerichtet. Der Wert eines Flusses  $f$  ist definiert als:

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

Beispiel: links ist der Netzwerk mit der Kapazitäten, rechts die Flüsse. Bei alle Knoten ausser der Quelle und Senke wollen



Fluss mit Wert  $3 - 1 + 5 = 7$

wir dass gleich viel hineinfliest wie herausfliesst. Der **Nettozufluss** der Senke  $t$  gleicht dem Wert des Flusses, d.h

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

**Schnitt:** Ein  $s$ - $t$ -Schnitt für ein Netzwerk ist eine Partition  $(S, T)$  von  $V$  mit  $s \in S$  und  $t \in T$ . Die **Kapazität** eines  $s$ - $t$ -Schnitts  $(S, T)$  ist durch

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

definert. Die Kapazität eines Schnitts  $(S, T)$  ignoriert die Kanten von  $T$  nach  $S$ !  
 $\Rightarrow$  Ist  $f$  ein Fluss und  $(S, T)$  ein  $s$ - $t$ -Schnitt in einem Netzwerk  $(V, A, c, s, t)$  so gilt:

$$val(f) \leq cap(S, T)$$

d.h ein Fluss kann nie grösser sein als die Kapazität eines s-t-Schnitts. Finden wir zu einem Fluss  $f$  einen s-t-Schnitt  $(S, T)$  mit  $\text{cap}(S, T) = \text{val}(f)$ , so ist  $f$  ein maximaler Fluss.

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

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

Da es nur endlich viele s-t-Schnitte gibt:

s-t-MinCut ist ein endliches algorithmisches Problem  
ein minimaler Schnitt existiert immer

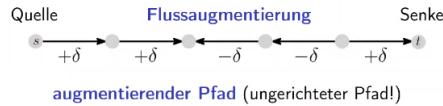
### 3.2 Algorithmen zur berechnung von Flüsse in Netzwerke

#### Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



Unter Beachtung  $\begin{cases} \text{der Kapazität} & \text{bei } "+\delta", \text{ und} \\ \text{des aktuellen Flusses} & \text{bei } "-\delta". \end{cases}$



Für  $e = (u, v)$ , sei  $e^{\text{opp}} := (v, u)$  (entgegen gerichtet Kante).

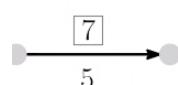
Sei  $N = (V, A, c, s, t)$  ein Netzwerk ohne entgegen gerichtete Kanten<sup>1</sup> und sei  $f$  ein Fluss in  $N$ . Das Restnetzwerk

$N_f := (V, A_f, r_f, s, t)$  ist wie folgt definiert:

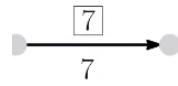
1. Ist  $e \in A$  mit  $f(e) < c(e)$ , dann ist  $e$  eine Kante in  $A_f$ , mit  $r_f(e) := c(e) - f(e)$ .

Netzwerk

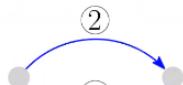
Restnetzwerk  
Restkapazität



2. Ist  $e \in A$  mit  $f(e) > 0$ , dann ist  $e^{\text{opp}}$  in  $A_f$ , mit  $r_f(e^{\text{opp}}) = f(e)$ .



3.  $A_f$  enthält nur Kanten wie in (1) und (2).



$r_f(e)$ ,  $e \in A_f$ , nennen wir die Restkapazität der Kante  $e$ .



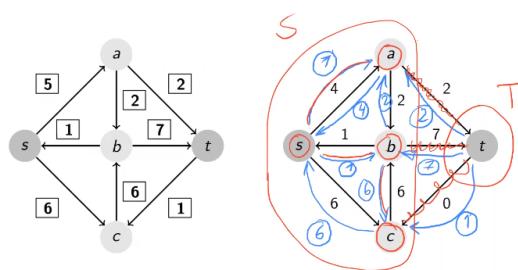
<sup>1</sup>Vereinfachende Annahme, ist aber nicht essentiell.

**Restnetzwerk:** Satz: Sei  $N$  ein Netzwerk (ohne entgegen gerichtete Kanten), Ein Fluss  $f$  ist maximaler Fluss  $\iff$  es im Restnetzwerk  $N_f$  keinen gerichteten s-t-Pfad gibt (d.h es gibt nur kanten in der vorgegebenen Richtung).

Für jeden maximalen Fluss  $f$  gibt es einen s-t-Schnitt  $(S, T)$  mit  $\text{val}(f) = \text{cap}(S, T)$

Es gibt im Restnetzwerk  $N_f$  einen gerichteten s-t-Pfad  $\Rightarrow f$  kann augmentiert werden ( $\Rightarrow f$  ist nicht maximal) Es gibt im Restnetzwerk  $N_f$  keinen gerichteten s-t-Pfad  $\Rightarrow \exists s - t$ -Schnitt  $(S, T)$  mit  $\text{cap}(S, T) = \text{val}(f)$  ( $\Rightarrow f$  ist maximal)

#### Beweis – Beispiel „Finde den Schnitt“



Fluss mit Wert  $4 - 1 + 6 = 9$

## Ford-Fulkerson Algorithmus

---

**Ford-Fulkerson( $V, A, c, s, t$ )**

```

1:  $f \leftarrow 0$                                 ▷ Fluss konstant 0
2: while  $\exists s-t$ -Pfad  $P$  in  $N_f$  do      ▷ augmentierender Pfad
3:   Augmentiere den Fluss entlang  $P$ 
4: return  $f$                                 ▷ maximaler Fluss

```

---

- ▶ Wir können nicht garantieren, dass der Algorithmus terminiert.
- ▶ Der Algorithmus kann bei Kapazitäten aus  $\mathbb{R}$  unendlich laufen.
- ▶ Bei Kapazitäten aus  $\mathbb{N}_0$  bleiben im Algorithmus die Flüsse und Restkapazitäten ganzzahlig, und wir verbessern den Fluss um mindestens 1 in jedem Augmentierungsschritt.

### Analyse

Sei  $n := |V|$  und  $m := |A|$  für Netzwerk  $N = (V, A, c, s, t)$ .

- ▶ Angenommen  $c: A \rightarrow \mathbb{N}_0$  und  $U := \max_{e \in A} c(e)$ . Dann gilt  $\text{val}(f) \leq \text{cap}(\{s\}, V \setminus \{s\}) \leq (n-1)U$  und es gibt höchstens  $(n-1)U$  Augmentierungsschritte.
- ▶ Ein Augmentierungsschritt  
Suche  $s-t$ -Pfad in  $N_f$ , Augmentieren, Aktualisierung von  $N_f$  benötigt  $O(m)$  Zeit.

Satz (Ford-Fulkerson mit ganzzahligen Kapazitäten)

Sei  $N = (V, A, c, s, t)$  ein Netzwerk mit  $c: A \rightarrow \mathbb{N}_0^{\leq U}$ ,  $U \in \mathbb{N}$ , ohne entgegen gerichtete Kanten.<sup>2</sup> Dann gibt es einen ganzzahligen maximalen Fluss. Er kann in Zeit  $O(mnU)$  berechnet werden.

<sup>2</sup>Vereinfachende Annahme, ist aber nicht essentiell.

### Ford-Fulkerson Algorithmus:

**Capacity-Scaling:** Sind in einem Netzwerk alle Kapazitäten ganzzahlig und höchstens  $U$ , so kann eine ganzzahliger maximaler Fluss in Zeit  $\mathcal{O}(mn(1 + \log U))$  berechnet werden.

**Dynamic Trees:** Der maximale Fluss eines Netzwerks kann in Zeit  $\mathcal{O}(mn \log n)$  berechnet werden

### 3.3 Flüsse im Netzwerken: Anwendung

**Maximum Bipartite Matching Problem:** Wir bilden jeden bipartiten Graphen (mit vorgegebener Knotenpartition  $U \uplus W$ ) Es gilt:

$$\overbrace{G = (U \uplus W, E)}^{\text{bipartiter Graph}} \mapsto \overbrace{N_G = (\underbrace{U \uplus W \uplus \{s, t\}}_{\text{Knotenmenge}}, A, c, s, t)}^{\text{Netzwerk}}$$

- ▶  $s \neq t$  zusätzliche Knoten.
- ▶  $A := \{s\} \times U \cup \{(u, w) \in U \times W \mid \{u, w\} \in E\} \cup W \times \{t\}$ .
- ▶  $c \equiv 1$ .

- Matching  $M$  in  $G \mapsto$  Fluss  $f_M$  in  $N_G$  mit  $\text{val}(f_M) = |M|$
- Ganzz. Fluss  $f$  in  $N_G \mapsto$  Matching  $M$  in  $G$  mit  $|M| = \text{val}(f)$

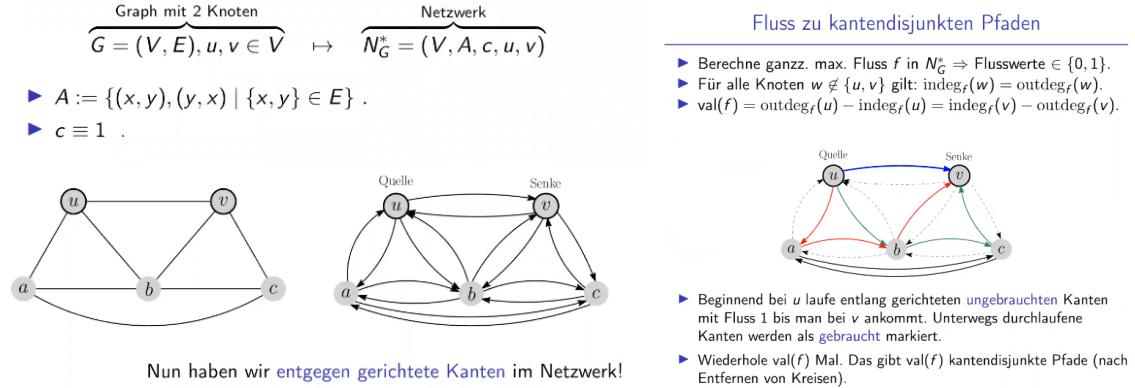
Maximum Matching in  $G$  „ $\simeq$ “ ganzz. Maxflow in  $N_G$ .

$$\max_{M \text{ Matching in } G} |M| = \max_{f \text{ Fluss in } N_G} \text{val}(f)$$

$\Rightarrow$  Bipartites maximum Matching kann in  $\mathcal{O}(mn)$  berechnet werden (geht auch in  $\mathcal{O}(m+n)\sqrt{n}$

**Kantendisjunkte Pfade Problem:** Gegeben ein Graph  $G$  mit zwei ausgezeichneten Knoten  $u$  und  $v$ ,  $v \neq u$  bestimme eine möglichst grosse Menge Kantendisjunkter  $u-v$ -Pfade.

## Graph zu Netzwerk (für kantendisjunkte Wege)



**Bildsegmentierung:** Gegeben ein Bild (aus Pixeln mit Farbwerten), trenne Vordergrund von Hintergrund. (Ein Bild ist eine Menge  $P$  von Pixeln mit Farben, mit einer Nachbarschaftsrelation, die sagt welche Pixel nebeneinander liegen.) Wir definieren ein Bild als Graph  $(P, E)$  mit Farbinformation  $x : P \rightarrow \text{Farben}$  und erhalten pro Bild drei Informationen und bilden eine Qualitätsfunktion für Vorder-/Hintergrundspartition  $(A, B)$  von  $P$ .

$$\begin{aligned}\alpha : P &\rightarrow \mathbb{R}_0^+ & \alpha_p \text{ grösser} &\Rightarrow \text{eher im Vordergrund} \\ \beta : P &\rightarrow \mathbb{R}_0^+ & \beta_p \text{ grösser} &\Rightarrow \text{eher im Hintergrund} \\ \gamma : E &\rightarrow \mathbb{R}_0^+ & \gamma_e \text{ grösser} &\Rightarrow \text{eher im gleichen Teil}\end{aligned}$$

Qualitätsfunktion für Vorder-/Hintergrundspartition  $(A, B)$  von  $P$ :

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E, |e \cap A|=1} \gamma_e .$$

### Problemstellung:

Bildsegmentierung. Gegeben ein Bild  $(P, E)$  mit

$$\alpha : P \rightarrow \mathbb{R}_0^+, \beta : P \rightarrow \mathbb{R}_0^+, \gamma : E \rightarrow \mathbb{R}_0^+$$

finde eine Partition  $(A, B)$  von  $P$  die

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E, |e \cap A|=1} \gamma_e$$

maximieren.

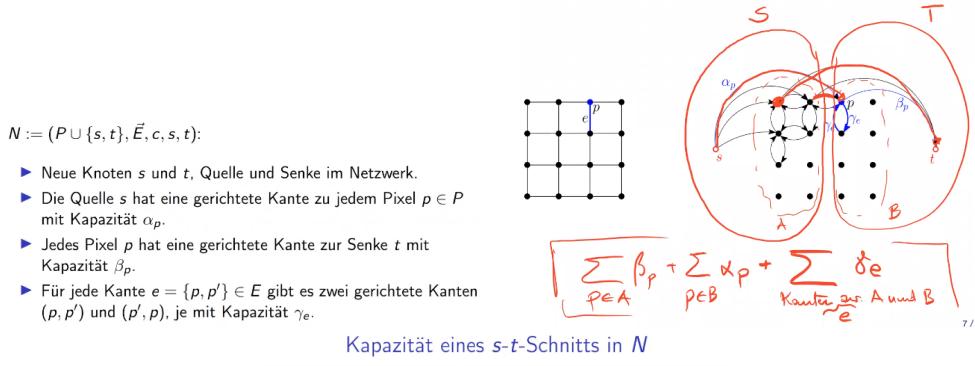
Mit  $Q := \sum_{p \in P} (\alpha_p + \beta_p)$  gilt:

$$q(A, B) := Q - \sum_{p \in A} \alpha_p - \sum_{p \in B} \beta_p - \sum_{e \in E, |e \cap A|=1} \gamma_e$$

$\Rightarrow q(A, B)$  zu maximieren ist äquivalent zur Minimierung von

$$q'(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p + \sum_{e \in E, |e \cap A|=1} \gamma_e$$

Ein Bild definieren wir als Netzwerk folgendermassen:



Sei  $(S, T)$  ein  $s-t$ -Schnitt, und  $A := S \setminus \{s\}$  und  $B := T \setminus \{t\}$ . Welche Kanten mit welchem Beitrag sind in diesem  $s-t$ -Schnitt?

- Kanten  $(s, p)$  mit  $p \in B$ ; Beitrag zu  $\text{cap}(S, T)$  ist  $\sum_{p \in B} \alpha_p$ .
- Kanten  $(p, t)$  mit  $p \in A$ ; Beitrag zu  $\text{cap}(S, T)$  ist  $\sum_{p \in A} \beta_p$ .
- Kanten  $(p, p')$  des Netzwerks in  $A \times B$  mit Beitrag

$$\sum_{(p, p') \in A \times B, \{p, p'\} \in E} \gamma_{(p, p')}.$$

Es folgt

$$\begin{aligned} q'(A, B) &:= \sum_{p \in A} \beta_p + \sum_{p \in B} \alpha_p + \sum_{e \in E, |e \cap A| = 1} \gamma_e \\ &= \text{cap}(S, T) \end{aligned}$$

Die optimale Partition  $(A, B)$  kann also mit Hilfe von MaxFlow für den minimalen  $s-t$ -Schnitt berechnet werden.

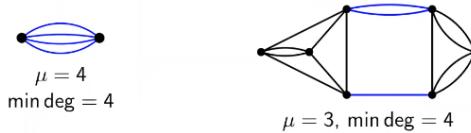
### 3.4 Minimale Schnitte in Graphen

**MIN-CUT Problem:** Gegeben ein Multigraph  $G$ , bestimme die Kardinalität eines minimalen Kantenschnitts.

- Multigraph: Ungerichteter ungewichteter Graph  $G$  ohne Schleifen, aber möglicherweise mit mehreren Kanten zwischen demselben Knotenpaar
- Kantenschnitt: In einem Multigraph  $G$  ist die Menge von Kanten  $C$ , so dass  $(V, E \setminus C)$  unzusammenhängend ist.
- $\mu(G)$  bezeichnet die Kardinalität eines kleinstmöglichen Kantenschnitts in  $G$  d.h.  $\mu(G) := \min_{C \subseteq E, (V, E \setminus C) \text{ unzusammenhängend}} |C|$

Für  $G$  nicht zusammenhängend gilt  $\mu(G) = 0$

In einem Multigraph  $G$  ist der Grad  $\deg(v) = \deg_G(v)$  eines Knoten  $v$  die #inzidenter Kanten (nicht die Anzahl Nachbarn).



So gilt:

$$|E| = \frac{1}{2} \sum_{v \in V} \deg(v) \text{ und } \mu(G) \leq \min_{v \in V} \deg(v)$$

Mögliche Lösungen:

- Erster Versuch:



Wir können bereits den kleinsten  $s-t$ -Schnitt berechnen, d.h. die kleinste #Kanten, die man entfernen muss, um  $s$  von  $t$  zu trennen.  
Zeit:  $O(mnU)$ , bzw.  $O(mn(1 + \log U))$  oder  $O(mn \log n)$ .

Für unser Problem fixieren wir ein  $s$  und betrachten alle  $t \in V \setminus \{s\}$ . Jeder Schnitt „ist“ ein  $s-t$ -Schnitt, für ein  $t \in V \setminus \{s\}$ .

( $n - 1$ ) Mal  $O(mn \log n)$ , gibt  $O(mn^2 \log n) = O(n^4 \log n)$ .

**Kantenkontraktion:** Gegeben  $G = (V, E)$ ,  $e = \{u, v\} \in E$ . Die Kontraktion von  $e$  verschmilzt  $u$  und  $v$  zu einem neuen Knoten  $x_{u,v}$  der nun zu allen Kanten inzident ist, zu denen  $u$  oder  $v$  inzident war. Die Kanten zwischen  $u$  und  $v$  verschwinden. Den entstehenden Graphen wird mit  $G/e$  bezeichnet. Es gilt:

$$\begin{aligned} \deg_{G/e} x_{u,v} &= \deg_G(u) + \deg_G(v) - 2k \\ |E(G/e)| &= |E(G)| - k \end{aligned}$$

wobei  $k = \#$  Kanten zwischen  $u$  und  $v$

Sei  $e = \{u, v\}$ . Es gibt eine natürliche Bijektion

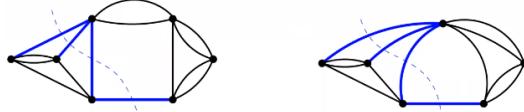
$$E(G) \text{ ohne Kanten zw. } u \text{ und } v \longrightarrow E(G/e).$$

Für  $w, w' \in V(G) \setminus \{u, v\}$ :

$$\{w, w'\} \mapsto \{w, w'\}, \{w, u\} \mapsto \{w, x_{u,v}\}, \{w, v\} \mapsto \{w, x_{u,v}\}$$

Dies induziert eine Bijektion

$$\text{Schnitte in } G \text{ ohne } e \longrightarrow \text{alle Schnitte in } G/e$$



Für jede Schnitt in  $G/e$  gibt es in  $G$  eine äquivalente Schnitt der gleiche Größe.

Lemma: Sei  $G = (V, E)$  ein Multigraph,  $e \in E$ . Dann gilt:

$$\mu(G/e) \geq \mu(G)$$

Falls  $G$  einen minimalen Schnitt  $C$  mit  $e \notin C$  hat, dann gilt:

$$\mu(G/e) = \mu(G)$$

Anders gesagt:  $\mu$  kann durch Kontraktion einer Kante  $e$  nie fallen und bleibt gleich, falls es einen minimalen Schnitt ohne  $e$  gibt. Unser Ziel ist es eine Kante  $e$  zu finden die  $\mu$  erhält. Wir nutzen den folgenden Algorithmus CUT, welche eine zufällige Kante wählt und Kontrahiert bis es nur noch 2 Knoten gibt und gibt den Wert der Schnitt zurück. Wir wissen:

- Der Ergebnis des Algorithmus ist nie kleiner als  $\mu(G)$
- Falls es einen minimalen Schnitt  $C$  gibt, aus dem nie eine Kante kontrahiert wird, so gibt der Algorithmus  $\mu(G)$  aus

Cut( $G$ )	$G$ zusammenhängender Multigraph
1: $G' \leftarrow G$	
2: <b>while</b> $ V(G')  > 2$ <b>do</b>	
3: $e \leftarrow$ gleichverteilt zufällige Kante in $G'$	
4: $G' \leftarrow G'/e$	
5: <b>return</b> Grösse des eindeutigen Schnitts in $G'$	

Sei  $n := |V(G)|$ . Wir setzen voraus:

- Kantenkontraktion in  $O(n)$  Zeit.
- Gleichverteilt zufällige Kante in  $G$  in  $O(n)$  Zeit.  
(Erfordert Darstellung der Mehrfachkanten durch Kantengewichte.)

Damit kann man Cut( $G$ ) mit Laufzeit  $O(n^2)$  implementieren.

Lemma: Sei  $G = (V, E)$ ,  $n := |V|$ . Für  $e$  gleichverteilt zufällig in  $E$  gilt:

$$\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$$

$\hat{p}(G) :=$  Wahrscheinlichkeit, dass Cut( $G$ ) den Wert  $\mu(G)$  ausgibt

$$\hat{p}(n) := \inf_{G=(V,E), |V|=n} \hat{p}(G)$$

Daraus folgt:

**Lemma**

Für alle  $n \geq 3$  gilt  $\hat{p}(n) \geq (1 - \frac{2}{n}) \cdot \hat{p}(n-1)$ .

**Beweis.** Sei  $G = (V, E)$ ,  $n := |V|$ . Damit  $\text{Cut}(G)$  tatsächlich  $\mu(G)$  ausgibt, müssen die beiden folgenden Ereignisse eintreten:

- $E_1 :=$  Ereignis  $\mu(G) = \mu(G/e)$ .
- $E_2 :=$  Ereignis, dass  $\text{Cut}(G/e)$  den Wert  $\mu(G/e)$  ausgibt.

Es gilt nun

$$\hat{p}(G) = \Pr[E_1 \wedge E_2] = \Pr[E_1] \cdot \Pr[E_2 | E_1] \geq (1 - 2/n) \cdot \hat{p}(n-1).$$

Da dies für jeden Multigraph  $G$  mit  $n$  Knoten gilt, folgt auch

$$\hat{p}(n) \geq (1 - \frac{2}{n}) \cdot \hat{p}(n-1).$$

Es gilt also  $\hat{p}(n) \geq \frac{n-2}{n} \cdot \hat{p}(n-1)$ . Wir erhalten so

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \underbrace{\frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3}}_{=1} \cdot \hat{p}(2) = \frac{2}{n(n-1)}$$

**Lemma**

Für alle  $n \geq 2$  gilt

$$\hat{p}(n) \geq \frac{2}{n(n-1)} = 1/\binom{n}{2}$$

□ Das heisst, der Erwartungswert der #Wiederholungen, bis wir das erste Mal  $\mu(G)$  ausgeben ist höchstens  $\binom{n}{2}$ .

Wir wiederholen den Algorithmus  $\text{Cut}(G)$   $\lambda \binom{n}{2}$  mal, für ein  $\lambda > 0$ , und geben dann kleinstejen erhaltene Wert aus.

Zur Erinnerung: Ein Aufruf von  $\text{Cut}(G)$  benötigt Zeit  $O(n^2)$ .

**Satz**

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

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

Mit  $\lambda := \ln n$ , haben wir Zeit  $O(n^4 \log n)$  mit Fehlerw'keit  $\leq 1/n$ .

Die Laufzeit hatten wir aber schon deterministisch (ohne Fehler)!

**Bootstrapping:** Idee ist dass man den Algorithmus Laufzeit verbessert indem man der Algorithmus verwendet. Wenn man der vorherige algorithmus betrachtet, wird die wahrscheinlichkeit sehr gross dass man eine zufällige Kante e wählt welches zur ein falsches ergebnis führt. Der deterministisches Algorithmus wäre sinnvoller bei kleinere anzahl Knoten. Daraus folgt wir müssen eine gleichgewicht finden zwischen Randomisiert und deterministisch. Wir bezeichnen die Anzahl Knoten wo dieses gleichgewicht ist mit t.

Wir brechen bei  $G'$  mit  $t$  Knoten ab und verwenden den randomisierten  $O(t^4)$  Algorithmus mit Erfolgsw'keit  $\geq 1 - e^{-1}$ .

$$\begin{aligned} \hat{p}_t(G) &:= \text{Wahrscheinlichkeit, dass Wert } \mu(G) \text{ ausgegeben wird} \\ \hat{p}_t(n) &:= \inf_{G=(V,E), |V|=n} \hat{p}_t(G), \quad \hat{p}_t(t) \geq 1 - e^{-1} = \frac{e-1}{e} \\ \hat{p}_t(n) &\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{t+1}{t+3} \cdot \frac{t}{t+2} \cdot \frac{t-1}{t+1} \cdot \hat{p}_t(t) \geq \underbrace{\frac{t(t-1)}{n(n-1)}}_{\hat{p}_t(n)} \cdot \frac{e-1}{e} \end{aligned}$$

$\lambda \frac{1}{\hat{p}_t(n)}$ -maliges Wiederholen gibt Fehlerw'keit  $\leq e^{-\lambda}$  und Laufzeit

$$\overbrace{\lambda \frac{n(n-1)}{t(t-1)} \frac{e}{e-1}}^{\text{#Wiederholungen}} \cdot O(\underbrace{n(n-t)}_{\text{Reduktion auf } t \text{ Knoten}} + \underbrace{t^4}_{\text{Alg. auf } t \text{ Knoten}}) = O\left(\lambda \left(\frac{n^4}{t^2} + n^2 t^2\right)\right)$$

Erfolgsw'keit  $\geq 1 - e^{-\lambda}$  und Laufzeit

$$O\left(\lambda \underbrace{\left(\frac{n^4}{t^2} + n^2 t^2\right)}_{\rightarrow \min}\right) \stackrel{t=\sqrt{n}}{=} O(\lambda n^3)$$

**Bootstrapping:** Es bietet sich an, die gleiche Methode nun mit dem neuen  $O(n^3)$  Algorithmus statt dem  $O(n^4)$  Algorithmus zu versuchen, und tatsächlich bekommen wir einen noch besseren, etc.

Im „Limit“ entwickelt sich so ein  $O(n^2 \text{polylog}(n))$ -Algorithmus.  
[Karger&Stein'96]

## Chapter 4

### Kleinster umschliessender Kreis

**Smallest Enclosing Disk-Problem:** Gegeben eine endliche Punktmenge  $P \subseteq \mathbb{R}^2$ , bestimme den Kreis kleinsten Radius, der  $P$  umschliesst. Für einen Kreis  $C$ , bezeichnen wir mit  $C^\bullet$  die geschlossene von  $C$  berandete Kreisscheibe.  $C$  **umschliesst**  $P$ , falls  $P \subseteq C^\bullet$ . (Punkte in  $P$  dürfen auf  $C$  liegen)

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

*Beweis (nur für Eindeutigkeit).* Angenommen,  $P$  hat zwei verschiedene kleinste umschliessende Kreise  $C_1$  und  $C_2$ , beide mit Radius  $r$  und Mittelpunkten  $z_1$  und  $z_2$ ,  $z_1 \neq z_2$ . haben. Es gilt

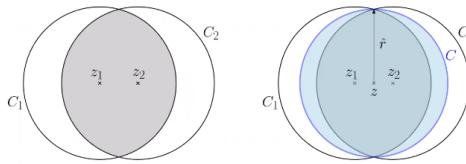
$$P \subseteq C_1^\bullet \cap C_2^\bullet.$$

Sei  $C$  der Kreis mit Mittelpunkt  $z = \frac{1}{2}(z_1 + z_2)$  (Mittelpunkt des Liniensegments zwischen  $z_1$  und  $z_2$ ). Sein Radius  $\hat{r}$  sei der Abstand von  $z$  zu den beiden Schnittpunkten von  $C_1$  und  $C_2$ . Dann gilt

$$P \subseteq C_1^\bullet \cap C_2^\bullet \subseteq C^\bullet \text{ und } \hat{r} = \sqrt{r^2 - \left(\frac{|z_1 z_2|}{2}\right)^2}$$

( $|z_1 z_2|$  Abstand von  $z_1$  zu  $z_2$ ).

Da  $\hat{r} < r$ , sind  $C_1$  und  $C_2$  nicht kleinste umschl. Kreise.  $\square$



Lemma 3.26: Für jede Punktmenge  $P \subseteq \mathbb{R}^2$ ,  $|P| \geq 3$ , gibt es eine Teilmenge  $Q \subseteq P$  so dass  $|Q|=3$  und  $C(Q) = C(P)$ . (Es kann sein dass es mit weniger punkte geht aber ist nicht garantiert).

Erster Ansatz; wir ziehen Kreise durch alle mögliche 3 punktige Teilmengen und schauen ob  $P$  eine Teilmenge davon ist falls Ja geben wir dieses Kreis zurück.

Zweiter Ansatz; Wir laufen wieder alle 3 punktige Teilmengen durch und nehmen den Kreis mit den grössten Radius, da nach der vorherige Lemma ist  $C(P)$  eindeutig und durch 3 punkte gegeben so sparen wir ein faktor n bei der Laufzeit.

---

CompleteEnumeration( $P$ )

```

1: for all  $Q \in \binom{P}{3}$  do
2:   bestimme  $C(Q)$ 
3:   if  $P \subseteq C^\bullet(Q)$  then
4:     return  $C(Q)$ 
```

---

Wir durchlaufen  $\binom{P}{3}$  Mengen  $Q$ , berechnen  $C(Q)$  in  $O(1)$  Zeit, und prüfen  $P \subseteq C^\bullet(Q)$  in  $O(n)$  Zeit. Dies ergibt eine Laufzeit  $O(n^4)$ .

---

CompleteEnumerationSmart( $P$ )

```

1:  $r \leftarrow 0$ 
2: for all  $Q \in \binom{P}{3}$  do bestimme  $C(Q)$  Laufzeit  $O(n^3)$ .
3:   if  $\text{radius}(C(Q)) > r$  then
4:      $C^* \leftarrow C(Q); r \leftarrow \text{radius}(C(Q))$ 
5: return  $C^*$ 
```

---

Dritter Ansatz; Wir wählen immer zufällige 3 elementige teilmengen und bestimmen den kleinstmögliche umschliessende Kreis und schauen ob  $P$  eine Teilmenge davon ist.

---

Randomised\_PrimitiveVersion( $P$ )

```

1: repeat forever
2:   wähle  $Q \subseteq P$  mit  $|Q| = 3$  zufällig und gleichverteilt
3:   bestimme  $C(Q)$ 
4:   if  $P \subseteq C^\bullet(Q)$  then
5:     return  $C(Q)$ 
```

---

Wir treffen auf das richtige  $Q$  mit Wahrscheinlichkeit  $\geq 1/\binom{n}{3}$ . Erwartete Anzahl der Versuche bis zum Erfolg ist  $\leq \binom{n}{3}$ , gibt erwartete Laufzeit  $O(n^4)$ .

Vierter Ansatz: Wir verdoppeln in jeder Iteration die Punkte ausserhalb von  $C(Q)$

Die 11 punkte werden zufällig und gleichverteilt mit folgenden Lemma gewählt: Die Teilmenge  $Q$  wird in zeit  $\mathcal{O}(n)$  bestimmt.

---

```
RANDOMISED_CLEVERVERSION(P)
1: repeat forever
2:   wähle  $Q \subseteq P$  mit  $|Q| = 11$  zufällig und gleichverteilt
3:   bestimme  $C(Q)$ 
4:   if  $P \subseteq C^*(Q)$  then
5:     return  $C(Q)$ 
6:   verdopple alle Punkte von  $P$  ausserhalb von  $C(Q)$ 
```

---

**Lemma 3.27.** Seien  $n_1, \dots, n_t$  natürliche Zahlen und  $N := \sum_{i=1}^t n_i$ . Wir erzeugen  $X \in \{1, \dots, t\}$  zufällig wie folgt:

```

k ← UNIFORMINT(1, N)
x ← 1
while  $\sum_{i=1}^x n_i < k$  do
  x ← x + 1
return x

```

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

**Lemma 3.28** Sei  $P$  eine Menge von  $n$  (nicht unbedingt verschiedenen) 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  $3 \frac{n-r}{r+1} \leq 3 \frac{n}{r+1} \Rightarrow \text{Satz 3.29}$ : Der Algorithmus Randomized CleverVersion berechnet den kleinsten umschliessenden Kreis von  $P$  in erwarteter Zeit  $\mathcal{O}(n \log n)$

**Das Sampling Lemma:** Gegeben sei eine endliche Menge  $S$ ,  $n := |S|$  und  $\phi$  eine beliebige Funktion auf  $2^S$  in einen beliebigen Wertebereich. Wir definieren

$$\begin{aligned} V(R) &= V_\phi(R) := \{s \in S \mid \phi(R \cup \{s\}) \neq \phi(R)\} \\ X(R) &= X_\phi(R) := \{s \in R \mid \phi(R \setminus \{s\}) \neq \phi(R)\} \end{aligned}$$

Elemente in  $V(R)$  nennen wir **Verletzer** von  $R$ , Elemente in  $X(R)$  nennen wir **extrem in  $R$**

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  $\frac{n-1}{r+1} + 1 = \frac{n+1}{r+1}$ . 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  $3 \frac{n-r}{r+1}$

## Chapter 5

### Konvexe Hülle: Jarvis Wrap

**ConvexHull-Problem:** Gegeben eine endliche Punktemenge  $P \subseteq \mathbb{R}^2$ , bestimme die konvexe Hülle von P.

**Liniensegment:** Die kleinste Strecke zwischen zwei Punkten (nicht der Gerade zwischen den Punkten).  
Sei  $d \in \mathbb{N}$ . Für  $v_0, v_1 \in \mathbb{R}^d$  sei

$$\overline{v_0 v_1} := \{(1 - \lambda)v_0 + \lambda v_1 | \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\}$$

**Konvexe Menge:** Eine Menge  $C \subset \mathbb{R}^d$  heißt konvex, falls für alle Punkte in C ist die Liniensegmente eine Teilmenge von C:

$$\forall v_0, v_1 \in C : \overline{v_0 v_1} \subseteq C$$

**Konvexe Hülle:** (denoted  $\text{conv}(S)$ ) einer Menge  $S \subseteq \mathbb{R}^d$  ist der Schnitt aller konvexen Mengen, die S enthalten d.h

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

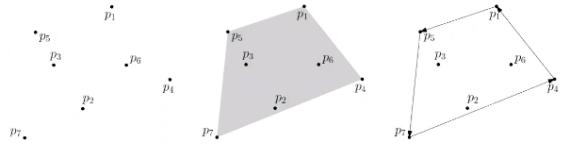
(also der kleinste Menge welche S enthält.) Für eine endliche Punktemenge P in der Ebene wird die konvexe Hülle durch ein Polygon, der Rand von  $\text{conv}(P)$ , bestimmt, dessen Ecken Punkte aus P sind. Bei der Berechnung von  $\text{conv}(P)$  wird die bestimmung der Folge

$$(q_0, q_1, \dots, q_{h-1}), h \leq n$$

der Ecken dieses Polygons, beginnend bei einer beliebigen Ecke  $q_0$  und dann entgegen dem Uhrzeigersinn entlang dieses Polygons gemeint.

$Q := \{q_0, q_1, \dots, q_{h-1}\} \subseteq P$  ist die kleinste Teilmenge von P mit  $\text{conv}(Q) = \text{conv}(P)$

#### Problemstellung



Punktemenge  $P$ , konvexe Hülle  $\text{conv}(P)$ , Polygon  $(p_4, p_1, p_5, p_7)$ .

**ConvexHull-Problem.** Gegeben eine endliche Punktemenge  $P \subseteq \mathbb{R}^2$ , bestimme die Ecken des  $\text{conv}(P)$  umrandenden Polygons, in der Reihenfolge gegen den Uhrzeigersinn.

Vereinfachende Annahme: **Allgemeine Lage**, d.h. keine 3 Punkte auf einer gemeinsamen Geraden, keine 2 Pkt gleiche x-Koordinate.

**Randkante:** Ein Paar  $qr \in P^2, q \neq r$  von P, falls alle Punkte in  $P \setminus \{q, r\}$  links von qr liegen d.h. auf der linken Seite der gerichteten Geraden durch q und r, gerichtet von q nach r liegen. **Lemma:**  $(q_0, q_1, \dots, q_{h-1})$  ist die Eckenfolge des  $\text{conv}(P)$  umschliessenden Polygons gegen den Uhrzeigersinn genau dann wenn alle Paare  $(q_{i-1}, q_i), i = 1, 2, \dots, h$  Randkanten von P sind (Indizes mod h)

**Orientierungstest:** Wie wird entschieden dass ein Punkt p links von einer Randkante liegt?  
(Gegenuhrzeigersinn durchlaufen heißt der Determinante ist positiv, Uhrzeigersinn wird negativ)

## Orientierungstest

Wie entscheiden wir „ $p$  liegt links von  $qr$ “?

### Lemma

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

$$\det(p, q, r) := \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{vmatrix} q_x - p_x & q_y - p_y & \\ r_x - p_x & r_y - p_y & \end{vmatrix} > 0$$

$\Leftrightarrow (q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)$

$\frac{1}{2} |\det(p, q, r)|$  = die Fläche des Dreiecks  $pqr$

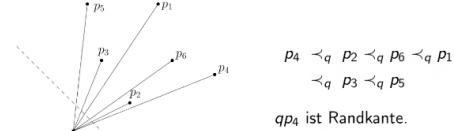
Das Vorzeichen von  $\det(p, q, r)$  bestimmt, wie die Punkte  $p, q, r$  den Rand dieses Dreiecks durchlaufen.

**Konvexe Hülle berechnen:** Erster Ansatz: Gehe durch jedes der  $n(n-1)$  geordneten Paare  $qr$ , und prüfe, ob dies eine Randkante ist, (indem man für alle  $n-2$  Punkte  $p$  in  $P \setminus \{q, r\}$  feststellt, ob  $p$  links von  $qr$  liegt). So haben wir die Randkanten in  $\mathcal{O}(n^3)$  gefunden, die wir nur mehr richtig aneinanderreihen müssen (die Kanten ordnen kann in  $\mathcal{O}(n^2)$  gemacht werden) Zweiter Ansatz: Nehme  $q_0 :=$  Punkt mit kleinster x-Koordinate in  $P$  (annahme es gibt nur ein Punkt mit kleinster x-Koordinate).  $q_0$  ist sicher eine Ecke der konvexen Hülle, also Teil der gesuchten Folge und wir können insbesondere die Folge auch mit  $q_0$  beginnen. Wir bestimmen den nächste Punkt  $q_1$ , der die Randkante  $q_0q_1$  bildet folgendermassen: (falls

## Ordnung um einen Punkt

Gegeben  $q \in P$ , sei  $\prec_q$  eine Relation auf  $P \setminus \{q\}$  mittels

$$p_1 \prec_q p_2 \Leftrightarrow p_1 \text{ rechts von } qp_2$$



$qp_4$  ist Randkante.

### Lemma

Ist  $q$  eine Ecke der konvexen Hülle von  $P$ , 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.

$q$  keine Ecke wäre, dann ist die Relation keine totale Ordnung) Es folgt die Randkante kann in linearer zeit bestimmt werden.  $\Rightarrow$  Jarvis Wrap: Erlauben wir Kollinearitäten müssen wir folgende Sachen adaptieren:

### JarvisWrap( $P$ )

```

1:  $h \leftarrow 0$ 
2:  $p_{\text{now}} \leftarrow$  Punkt in  $P$  mit kleinster x-Koordinate
3: repeat
4:    $q_h \leftarrow p_{\text{now}}$ 
5:    $p_{\text{now}} \leftarrow \text{FindNext}(q_h)$ 
6:    $h \leftarrow h + 1$ 
7: until  $p_{\text{now}} = q_0$ 
8: return ( $q_0, q_1, \dots, q_{h-1}$ )

```

### Satz

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

► Da  $h \leq n$ , läuft JarvisWrap in  $\mathcal{O}(n^2)$  (statt  $\mathcal{O}(n^3)$ ).

► Ist  $h = O(1)$ , z.B.  $\text{conv}(P)$  ist ein Dreieck, läuft der Algorithmus in  $\mathcal{O}(n)$  Zeit.

Für Punkte zufällig in einem Quadrat: Erwartete #Ecken  $\mathcal{O}(\log n)$ .

Für Punkte zufällig in einem Kreis: Erwartete #Ecken  $\mathcal{O}(\sqrt[3]{n})$ .

### Kollinearitäten (3 Punkte auf Gerade), gleiche x-Koord., ...

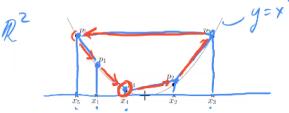
- Anfangspunkt  $q_0$  als den Punkt mit lexikographisch kleinster Koordinate (unter allen mit kleinster x-Koordinate, den mit kleinster y-Koordinate). (Adaption der x-Ordnung der Punkte)
- Der Test „ $p$  rechts von  $qq_{\text{next}}$ “ muss ersetzt werden durch ( $p$  rechts von  $qq_{\text{next}}$ ) oder ( $p$  auf der Geraden durch  $qq_{\text{next}}$  und  $|qp| > |qq_{\text{next}}|$ ). (Adaption der Ordnung  $\prec_q$ )
- In der Regel können wir nicht einmal annehmen, dass die Punkte verschieden sind (z.B. gegeben in einem Feld)!

Software ohne Berücksichtigung dieser Fälle hat geringen Nutzen!

**Untere Schranke für ConvexHull:** Der folgende Bild besagt, dass wir die konvexe Hülle nicht schneller finden können als  $n$  Zahlen zu sortieren d.h. wir haben einen  $\mathcal{O}(n \log n)$  untere Schranke.

### Untere Schranke für ConvexHull

Betrachte eine Folge  $(x_1, x_2, \dots, x_n)$  von Zahlen in  $\mathbb{R}$ . Wir setzen  
 $\rightarrow p_i = (x_i, x_i^2), i = 1, 2, \dots, n$  (vertikale Projektion von der  $x$ -Achse  
im  $\mathbb{R}^2$  auf die Einheitsparabel  $y = x^2$ ).



Aus der Folgen der Ecken der konvexen Hülle von  
 $P := \{p_1, p_2, \dots, p_n\}$  ergibt sich (in linearer Zeit) auch die  
aufsteigend sortierte Reihenfolge der  $x_i$ 's.

Wir haben eine sogenannte **Reduktion** gezeigt: Kann man  
ConvexHull in  $t(n)$  lösen, so kann man in  $t(n) + O(n)$  Zeit sortieren.

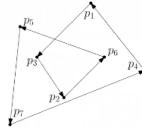
## 5.1 Lokales Verbessern

**lokal konvex:** Gegeben ein Polygon wir prüfen ob der nächste Punkt der Polygon links von den Randkante gegeben durch den zwei vorherigen Punkten ist. Wir machen den folgenden Lokalen Prüfung in Polygon  $(q_0, q_1, \dots, q_{h-1})$ :

$$\forall i, 1 \leq i \leq h : q_{i+1} \text{ links von } q_{i-1} q_i$$

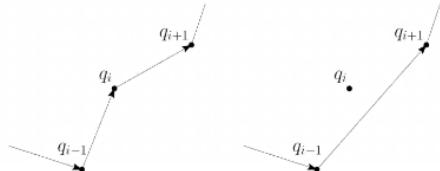
### Lokale Bedingung – Defizite

- $\{q_0, q_1, \dots, q_{h-1}\}$  muss nicht Teilmenge von  $P$  sein.
- Das Polygon muss nicht alle anderen Punkte im Inneren haben.
- Das Polygon kann sich selbst kreuzen.

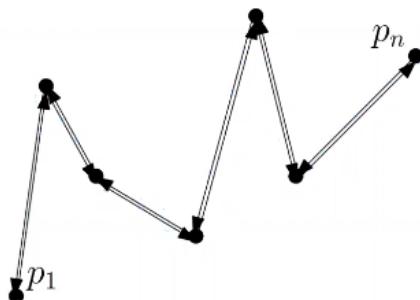


Idee: Wir beginnen mit einem nicht notwendigerweise konvexen  
Polygon, dass die Bedingungen oben nicht verletzt, und  
„konvexitifizieren“ das sukzessive („**lokal Verbessern**“).

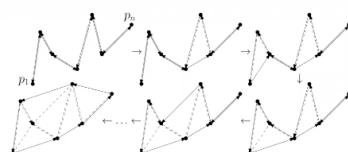
**lokal verbessern:** Gegeben  $(q_0, q_1, \dots, q_{h-1})$ , falls  $q_i$  links von  $q_{i-1} q_{i+1}$  liegt dann entferne  $q_i$  aus der Folge. Eine gültige



startpolygon finden wir, indem wir  $P$  aufsteigend nach  $x$ -Koordinate sortieren. Wir kriegen die Folge  $(p_1, p_2, \dots, p_n)$  und betrachten den polygon  $(p_1, p_2, \dots, p_{n-1}, p_n, p_{n-1}, \dots, p_2)$



- Der Teilstückspolygonzug  $(p_1, \dots, p_n)$  ist  $x$ -monoton (von links nach rechts) und hat keinen Punkt in  $P$  unter sich.
- Der Teilstückspolygonzug  $(p_n, \dots, p_1)$  ist  $x$ -monoton (von rechts nach links) und hat keinen Punkt in  $P$  unter sich.
- Der Teilstückspolygonzug  $(p_1, \dots, p_n)$  liegt nirgends über dem Teilstückspolygonzug  $(p_n, \dots, p_1)$ .



⇒ Das Polygon kann sich nie selber kreuzen und umschliesst alle Punkte. Es gilt:

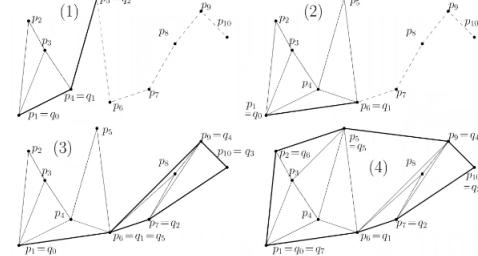
Lokal konvex  $\Rightarrow$  Lösungspolygon

Der Algorithmus ist nicht-deterministisch d.h wir können die lokalen Verbesserungsschritte in beliebiger Reihenfolge machen. Der Reihenfolge wie der Algorithmus die Punkte bearbeitet ist von links nach rechts und dann rechts nach

```

LocalRepair( $p_1, p_2, \dots, p_n$ )           ( $p_1, p_2, \dots, p_n$ ) sortiert
1:  $q_0 \leftarrow p_1; h \leftarrow 0$ 
2: for  $i \leftarrow 2$  to  $n$  do          ▷ unterer Rand, links nach rechts
3:   while  $h > 0$  und  $q_h$  links von  $q_{h-1}p_i$  do
4:      $h \leftarrow h - 1$ 
5:    $h \leftarrow h + 1; q_h \leftarrow p_i$ 
6:  ▷  $(q_0, \dots, q_h)$  untere konvexe Hülle von  $\{p_1, \dots, p_i\}$ 
7:  $h' \leftarrow h$ 
8: for  $i \leftarrow n - 1$  downto 1 do    ▷ oberer Rand, rechts nach links
9:   while  $h' > h$  und  $q_h$  links von  $q_{h-1}p_i$  do
10:     $h' \leftarrow h' - 1$ 
11:    $h \leftarrow h + 1; q_h \leftarrow p_i$ 
12: return  $(q_0, q_1, \dots, q_{h-1})$ 

```



links.

Analyse: Wir beginnen mit einem Polygon mit  $2(n-1)$  Ecken und haben am Ende  $h$  Ecken. Wir verbessern also genau  $2(n-1)-h = \mathcal{O}(n)$  mal.

$\Rightarrow$  Es gibt also  $\mathcal{O}(n)$  erfolgreiche Tests ( $q_h$  links von  $q_{h-1}p_i$ ) und für jeden Punkt  $p_i$  zwei erfolglose

$\Rightarrow$  Der Algorithmus hat also nach dem anfänglichen Sortieren in  $\mathcal{O}(n \log n)$  eine Laufzeit von  $\mathcal{O}(n)$

- ▶ Einbezug von Degeneriertheiten ist einfach (Sortieren lexicographisch, Duplikate nach Sortieren entfernen, „ $q_h$  links von  $q_{h-1}p_i$ “ adaptieren).
- ▶ Der Algorithmus ist numerisch robuster als JarvisWrap, kann nie eine unendliche Schleife laufen.
- ▶ Algorithmus liefert auch eine Triangulierung der Punkte.
- ▶ Lokale Verbesserung kann auch zur Berechnung „guter“ Triangulierungen (Delaunay Triangulierungen) verwendet werden.
- ▶ LocalRepair ist optimal, aber: (i) Es gibt auch einen  $\mathcal{O}(n \log h)$  Algorithmus und (ii) für Punkte zufällig aus Quadrat oder Kreisfläche gibt es einen erwartet  $\mathcal{O}(n)$  Zeit-Algorithmus.

#### Satz

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

## 5.2 Konvexe Mengen

Für  $M$  und  $N$  Mengen, ist  $M^N$  die Menge aller Funktionen  $N \rightarrow M$  (Wenn  $M$  und  $N$  endliche Mengen sind, dann gilt  $|M^N| = |M|^{|N|}$ ) Insbesondere, ist  $M^{[n]}$  die Menge der Folgen der Länge  $n$  mit Elementen aus  $M$ . Wir schreiben oft einfach  $M^n$

$$\mathbb{R}^n \approx \mathbb{R}^{[n]} \approx \mathbb{R}^N \text{ für } |N| = n$$

#### Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel  $N = (V, A, c, s, t)$ , mit  $V$  endliche Menge,  $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$ ,  $s \in V$ ,  $t \in V \setminus \{s\}$  und

$$c \in \mathbb{R}_0^+ A. \quad c \in \mathbb{R}_0^+ m, \quad m := |A| \\ A = \{e_1, e_2, \dots, e_m\}$$

$f \in \mathbb{R}^A$  heißt **Fluss** von  $N$ , falls

$$\forall e \in A: \quad f(e) \geq 0 \quad \text{und} \quad f(e) \leq c(e) \\ \forall v \in V \setminus \{s, t\}: \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

$$\text{wobei } \sigma_{v,e} := \begin{cases} -1 & e \text{ in } v \text{ eingehende Kante,} \\ 1 & e \text{ aus } v \text{ ausgehende Kante, und} \\ 0 & \text{sonst.} \end{cases}$$

Flüsse sind Punkte (bzw. Vektoren) im  $\mathbb{R}^A$  „ $\simeq$ “  $\mathbb{R}^m$  mit gewissen Bedingungen.

**Gerade im  $\mathbb{R}^2$**   $(a, b) \neq (0, 0)$   
Für  $(a, b, c) \in \mathbb{R}^3$  gegeben:  $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

**Ebene im  $\mathbb{R}^3$**   $(a, b, c) \neq (0, 0, 0)$   
Für  $(a, b, c, d) \in \mathbb{R}^4$  geg.:  $\{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz = d\}$

**Hyperebene im  $\mathbb{R}^n$**   $(a_1, \dots, a_n) \neq (0, \dots, 0)$   
Für  $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$  geg.:  $\{(x_1, \dots, x_n) \in \mathbb{R}^n \mid \sum_{i=1}^n a_i x_i = a_0\}$

Jede Ebene ist konvex. Hyperebenen sind konvexe Mengen. Schnitte von Hyperebenen sind konvexe Mengen. 2m Halbräumen für die Zulässigkeitsbedingung und  $n-2$  Hyperebenen für die Flusserhaltung Ford-Fulkerson läuft am Rand der konvexen Menge aller Flüsse zum maximalen Fluss Konvexe Mengen zeichnen sich dadurch aus, dass man beim Optimieren nicht in einem lokalen nichtglobalen Optimum stecken bleiben kann.

## Beispiel - Flusserhaltung

$f \in \mathbb{R}^A$  heisst **Fluss** von Netzwerk  $N = (V, A, c, s, t)$ , falls

$$\begin{aligned} \forall e \in A: \quad & f(e) \geq 0 \text{ und } f(e) \leq c(e) \\ \forall v \in V \setminus \{s, t\}: \quad & \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0 \end{aligned}$$

Flusserhaltung

Die Flusserhaltungsbedingungen für  $v \in V \setminus \{s, t\}$  sind Hyperebenen. Die Flüsse, die alle diese Flusserhaltungen erfüllen bilden also einen Schnitt von Hyperebenen und daher eine konvexe Menge.

**Halbebene im  $\mathbb{R}^2$**   
 $(a, b) \neq (0, 0)$   
 Für  $(a, b, c) \in \mathbb{R}^3$  gegeben:  $\{(x, y) \in \mathbb{R}^2 \mid ax + by \geq c\}$

**Halbraum im  $\mathbb{R}^n$**   
 $(a_1, \dots, a_n) \neq (0, \dots, 0)$   
 Für  $(a_0, \dots, a_n) \in \mathbb{R}^{n+1}$  geg.:  $\{(x_1, \dots, x_n) \in \mathbb{R}^n \mid \sum_{i=1}^n a_i x_i \geq a_0\}$

**Beobachtung**  
 Halbräume sind konvexe Mengen.  
 Schnitte von Halbräumen sind konvexe Mengen.

## Beispiel - Zulässigkeit

$f \in \mathbb{R}^A$  heisst **Fluss** von Netzwerk  $N = (V, A, c, s, t)$ , falls

$$\begin{aligned} \forall e \in A: \quad & f(e) \geq 0 \text{ und } f(e) \leq c(e) \\ \forall v \in V \setminus \{s, t\}: \quad & \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0 \end{aligned}$$

$\swarrow$  Zulässigkeit

Die Zulässigkeitsbedingungen für  $e \in A$  sind Halbräume. Flüsse bilden also einen Schnitt von  $2m$  Halbräumen und  $n - 2$  Hyperebenen im  $\mathbb{R}^A$  ( $n := |V|$ ,  $m := |A|$ ).

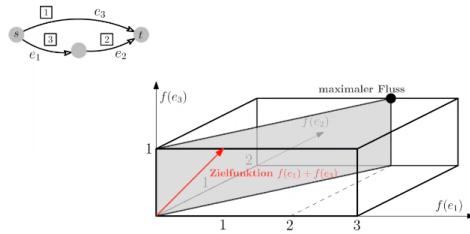
**Die Menge der Flüsse ist eine konvexe Menge im  $\mathbb{R}^A$ .**

Ziel ist es in dieser konvexen Menge einen Punkt zu finden, der

$$\underline{\text{netoutflow}(s) := \sum_{e \in A} \sigma_{s,e} \cdot f(e)}$$

(eine lineare Funktion) maximiert.

## Beispiel – Menge der Flüsse



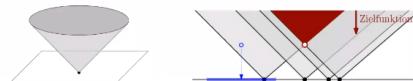
## Beispiel – Kleinster umschliessender Kreis

Ein Kreis ist bestimmt durch ein **Tupel**  $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$ .

Der Kreis (des Kreis-Tupels  $K$ ) umschliesst Punkt  $p := (x_1, x_2)$  falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Die Kreis-Tupel  $(c_1, c_2, r)$ , deren Kreise  $p = (x_1, x_2)$  umschließen, bilden einen vertikalen Kegel (konvex) im  $\mathbb{R}^3$  mit Spitze  $(x_1, x_2, 0)$ .



Unser Ziel ist es im Schnitt aller Kegel, eine konvexe Menge, einen Punkt (d.h. Kreis-Tupel) zu finden, das  $r$  minimiert. (Der tiefste Punkt im Schnitt ist immer durch höchstens drei Kegel bestimmt.)