Moderne Theoretische Informatik

Masterstudiengang Informatik
Sommersemester 2019

Prof. Dr. Wolfgang Mauerer
wolfgang.mauerer@oth-regensburg.de

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Inhalt

OTH Regensburg

- Professor für theoretische Informatik
- Leiter Labor für Digitalisierung
- Forschungsinteressen
  - Innovation durch Theorie (Zufallserzeugung, stochastische Methoden)
  - Industrie 4.0 (*Cyber Physical* und *Smart Embedded*)
  - Statistik und maschinelles Lernen in der SW-Entwicklung
- Hacken *und* Beweisen

Siemens Corporate Research

- Linux (Kernel, Low-Level)
- Harte Echtzeit (MRT, Simatic, Àndroit, …)
- Statistik und maschinelles Lernen in der SW-Architektur
  `siemens.github.com/codeface`
- Früher: MPL (QIT, QED)

Vorlesungszeiten

- Do, 08:15–10:00, K005
- Do, 11:45–13:15, K016

Übungen

- Übungen (≈ 25%) werden in die Vorlesung integriert
- Bei Gelegenheit: Rechner für Programmierübungen bereithalten

Scheinkriterien

- Klausur (90min) am Ende des Semesters
  - Keine Hilfsmittel
  - Material der gesamten Vorlesung relevant
  - Orientiert sich an Übungsaufgaben

Modus Operandi

- Seminaristischer Unterricht, aktive Teilnahme und Rückfragen erwünscht.

Themen

0.) Wiederholung

- Bachelor-Vorlesung TI: Komplexität von Algorithmen, **P** und **NP**, **NP**-Vollständigkeit und Reduktion

1.) Anwendung

- Anwendung und Analyse randomisierter Algorithmen, Derandomisierung

2.) Fundament

- Mächtigkeit (und Grenzen) des randomisierten Rechnens

3.) Struktur

- Komplexitätsklassen und ihre Beziehungen

4.) Synthese

- Zufall als Bindeglied zwischen Informatik und Naturwissenschaften

Themenüberblick

- Randomisierte Algorithmen
- Probabilistische Analyse
- Spezielle Derandomisierung von Algorithmen
- Randomisierte Komplexitätsklassen
- Allgemeine Derandomisierung (und deren Grenzen)
- Zufallserzeugung und Zufallsextraktion
- Komplexitätshierarchie und Struktur des Rechnens
- Quantencomputing

Ziele

Praktisch

- Analyse zufälliger Algorithmen & randomisierte Analyse
- Fundamentale Rolle des Zufalls; Grenzen und Anwendbarkeit von Berechnungsmodellen
- Kryptographie, numerische Simulation, Produktlinien, Produktionsoptimierung, Verkehrsplanung, Schach, Ego-Shooter, …
- … neue Entwicklungen der letzten 30 Jahre

Konzeptionell

- Heranführen an aktuelle Forschungsprobleme und moderne Ansätze
- Arbeit mit Formalismen und abstrakter, kompakter Darstellungsweise
- *Warum* und *wann* sind Probleme leicht oder schwer?
- Anknüpfung an Informationstheorie und Naturwissenschaften

Literatur

- C. Moore und St. Mertens, *The Nature of Computation*, Oxford University Press, 2011.
- M. Mitzenmacher, E. Upfal, *Probability and Computing*, Cambridge University Press, 2005.
- S. Arora und B. Barak, *Computational Complexity*, Cambridge University Press, 2009.

Target Audience Check

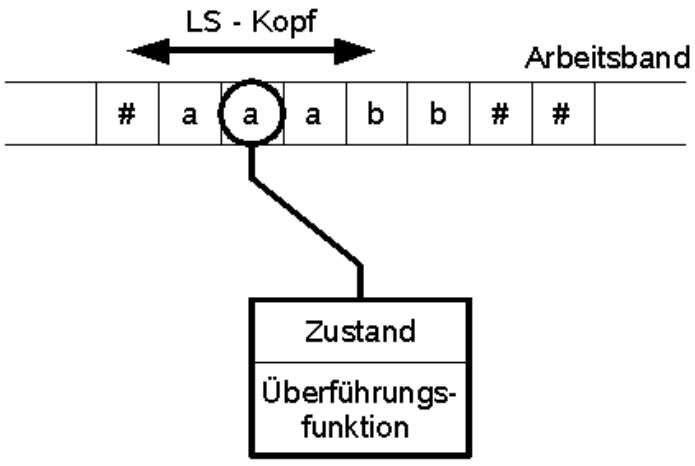
- Chomsky-Hierarchie?
- DEAs, NEAs, Kellerautomaten, Turing-Maschinen?
- Zusammenhang zu formalen Sprachen?
- Komplexitätsklassen **P** und **NP**?
- Halteproblem und Satz von Rice?
- Gödel'scher Unvollständigkeitssatz?
- Zufallsvariablen und ihre Eigenschaften?
- Endliche Körper?

Überblick Theoretische Informatik I

Wiederholung: Themen

- Entscheidungsprobleme und Berechnungskomplexität
- Analyse von Algorithmen und Zeitkomplexität
- Skalierungsverhalten und Komplexitätsklassen
- P und NP: Determinismus und Nichtdeterminismus
- NP-vollständige Probleme

Turing-Maschine



Entscheidungsprobleme III

(Nicht-)Deterministische Turing-Maschine

Eine *Turing-Maschine* ist gegeben durch ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$. Dabei ist

- Q eine endliche Zustandsmenge
- Σ ein endliches Eingabealphabet
- $\Gamma \supset \Sigma$ ein endliches Arbeitsalphabet
- δ die Transitionsfunktion mit Signatur
  - Deterministisch: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$
  - Nicht-Deterministisch: $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$
- $q_0 \in Q$ der Startzustand
- $\square \in \Gamma - \Sigma$ das »Blank«-Symbol
- $F \subseteq Q$ die Menge der Endzustände

Entscheidungsprobleme IV

Gegeben Sprache L, Turing-Maschine M.

Akzeptanz

- $\forall\, w \in L$: M hält in q_{accept}
- $\forall\, w \notin L$: M hält *nicht* in q_{accept}

Entscheidbarkeit

- $\forall\, w \in L$: M hält in q_{accept}
- $\forall\, w \notin L$: M hält in q_{reject}
- TM hält immer!

Grenzen des Rechnens

- Es gibt nicht-entscheidbare Probleme
- Nicht-deterministische TM ändert nichts an Entscheidbarkeit.

Entscheidungsprobleme IV

Gegeben Sprache L, Turing-Maschine M.

Akzeptanz

- $\forall\, w \in L$: M hält in q_{accept}
- $\forall\, w \notin L$: M hält *nicht* in q_{accept}

Entscheidbarkeit

- $\forall\, w \in L$: M hält in q_{accept}
- $\forall\, w \notin L$: M hält in q_{reject}
- TM hält immer!

Grenzen des Rechnens

- Es gibt nicht-entscheidbare Probleme
- Nicht-deterministische TM ändert nichts an Entscheidbarkeit.

Entscheidungsprobleme IV

Äquivalente Berechenbarkeits-Modelle

- Deterministische und Nicht-Deterministische TMs
- Mehrband-TMs
- Goto- und While-Sprache
- Registermaschinen
- μ-rekursive Funktionen

Summa Summarum

Turing-Berechenbar $\Leftrightarrow$ Berechenbar durch Standard-Programmiersprache

Church-Turing-These

Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.

Classical Complexity Theoretic Church-Turing Thesis

A probabilistic Turing machine can efficiently simulate any realistic model of computation.

- Genaueres im weiteren Verlauf der Vorlesung …

Church-Turing-These

Die Klasse der Turing-berechenbaren Funktionen stimmt mit der Klasse der intuitiv berechenbaren Funktionen überein.

Classical Complexity Theoretic Church-Turing Thesis

A probabilistic Turing machine can efficiently simulate any realistic model of computation.

- Genaueres im weiteren Verlauf der Vorlesung …

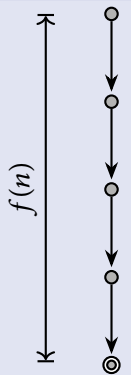
Zeit ist Geld

- Wenn eine Funktion berechenbar ist, *wie aufwendig* ist die Berechnung?
- Laufzeit- und Ressourcenmessung erforderlich (in Abhängigkeit der Eingabe)
  - Wie lange (mit wie vielen Zeitschritten) rechnet die TM?
  - Wie viele Bandzellen werden benötigt?
- Welche entscheidbaren Probleme sind tatsächlich (mit vorhandenen Ressourcen) lösbar?

Definition: Laufzeitmessung für DTM

- Gegeben: DTM M, darauf entscheidbare Sprache L.
  Länge der Eingabe: $n \equiv |w|, w \in L$.
- Laufzeit bzw. Zeitkomplexität: $f : \mathbb{N} \to \mathbb{N}$, f gibt maximale Anzahl Schritte von M an.
- M läuft in in Zeit $f(n)$.
  M ist eine $f(n)$ Turing-Maschine.

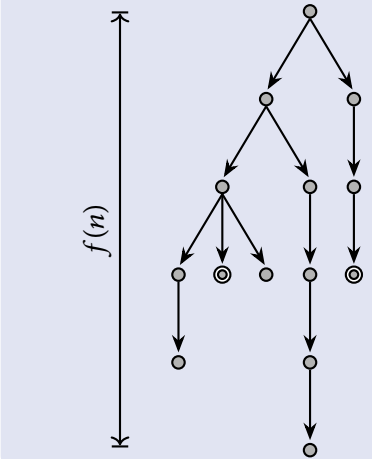
Illustration



Definition: Laufzeitmessung für NTM

- Gegeben: NTM M, darauf entscheidbare Sprache L.
  Länge der Eingabe: $n \equiv |w|, w \in L$.
- Definition Laufzeit bzw. Zeitkomplexität: $f : \mathbb{N} \to \mathbb{N}$, f gibt maximale Anzahl Schritten von M an.
- Aussagen:
  M läuft in in Zeit $f(n)$.
  M ist eine $f(n)$-Turing-Maschine.

Illustration



Probleme

- Exakte Laufzeitbestimmung: Komplexe Ausdrücke.
- »Einmaleffekte«: Initialisierung, Buchhaltung, …
- Konstanten können (je nach Ressourcenmodell) von HW abhängen.
- Statische Beiträge irrelevant für große Eingaben.

Asymptotik

- $f(n) = 6n^3 + 2n^2 + 20n + 45$
- $f(1000) = 6 \times 10^9 + 2 \times 10^6 + 20 \times 1000 + 45$
- Quadratischer Term $\approx$ 0.001 kubischer Term
- Wesentlich: $f(n) = \mathcal{O}(n^3)$
- Vorsicht bei »=«! Keine Gleichheit im üblichen Sinn.

Definition: Landau-Symbol

Gegeben $f, g : \mathbb{N} \to \mathbb{R}^+$. Man sagt $f(n) = \mathcal{O}(g(n))$, wenn $\exists c, n_0 \in \mathbb{N}^+$, so dass $\forall n \geq n_0$

$$f(n) \leq c \cdot g(n).$$

Asymptotische Notation

Bezeichnung	$C = \lim_{n\to\infty} \frac{f(n)}{g(n)}$	Grob gesprochen
$f(n) = \mathcal{O}(g(n))$	$C < \infty$	$f \leq g$
$f(n) = \Omega(g(n))$	$C > 0$	$f \geq g$
$f(n) = \Theta(g(n))$	$0 < C < \infty$	$f = g$
$f(n) = o((g(n))$	$C = 0$	$f < g$
$f(n) = \omega(g(n))$	$C = \infty$	$f > g$

Komplexitätsklassen

- Gruppierung vergleichbar schwieriger Algorithmen.
- Üblicherweise: Zeit- und Platzbedarf, andere Ressourcen als Maß möglich.

Zeitkomplexitätsklasse TIME

Sei $t : \mathbb{N} \to \mathbb{R}^+$. Die Zeitkomplexitätsklasse $\mathrm{TIME}(t(n))$ enthält alle Sprachen, die von einer $\mathcal{O}(t(n))$-DTM entschieden werden.

Probleme

- Gute Funktionen t zur Klassifikation?
- Detailunterschiede in Berechnungsmodellen? (Erinnerung: Für $t(n)$ mit $t(n) \geq n$ gibt es für jede $\mathcal{O}(t(n))$-Mehrband-TM eine gleichwertige $\mathcal{O}(t^2(n))$-Einband-TM).

Komplexitätsklassen

- Gruppierung vergleichbar schwieriger Algorithmen.
- Üblicherweise: Zeit- und Platzbedarf, andere Ressourcen als Maß möglich.

Zeitkomplexitätsklasse TIME

Sei $t : \mathbb{N} \to \mathbb{R}^+$. Die Zeitkomplexitätsklasse $\mathbf{TIME}(t(n))$ enthält alle Sprachen, die von einer $\mathcal{O}(t(n))$-DTM entschieden werden.

Probleme

- Gute Funktionen t zur Klassifikation?
- Detailunterschiede in Berechnungsmodellen? (Erinnerung: Für $t(n)$ mit $t(n) \geq n$ gibt es für jede $\mathcal{O}(t(n))$-Mehrband-TM eine gleichwertige $\mathcal{O}(t^2(n))$-Einband-TM).

Komplexitätsklassen

- Gruppierung vergleichbar schwieriger Algorithmen.
- Üblicherweise: Zeit- und Platzbedarf, andere Ressourcen als Maß möglich.

Zeitkomplexitätsklasse TIME

Sei $t : \mathbb{N} \to \mathbb{R}^+$. Die Zeitkomplexitätsklasse $\mathbf{TIME}(t(n))$ enthält alle Sprachen, die von einer $\mathcal{O}(t(n))$-DTM entschieden werden.

Probleme

- Gute Funktionen t zur Klassifikation?
- Detailunterschiede in Berechnungsmodellen? (Erinnerung: Für $t(n)$ mit $t(n) \geq n$ gibt es für jede $\mathcal{O}(t(n))$-Mehrband-TM eine gleichwertige $\mathcal{O}(t^2(n))$-Einband-TM).

Definition: Komplexitätsklasse P

$$\mathbf{P} \equiv \bigcup_k \mathbf{TIME}(n^k)$$

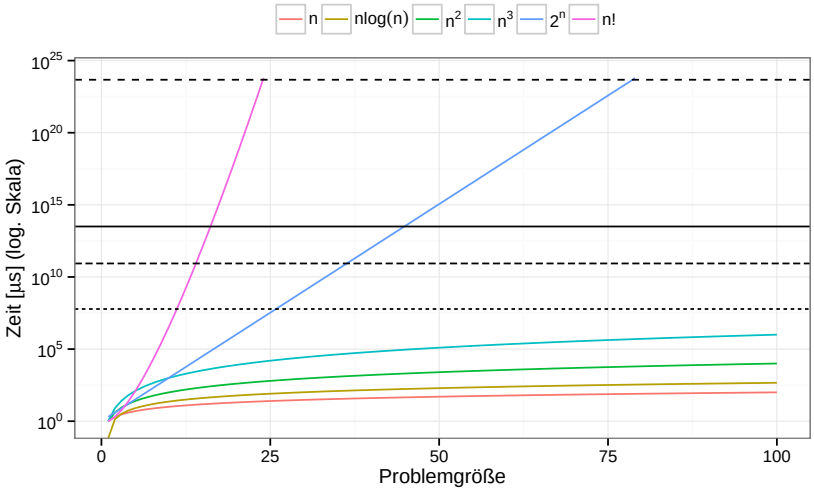
Eigenschaften

- Invariant gegenüber polynomialen Beschleunigungen/Verlangsamungen.
- Klasse der praktisch lösbaren Probleme (keine strikte Trennschärfe!).
- Enthält $\mathbf{TIME}(1)$, $\mathbf{TIME}(\log n)$, $\mathbf{TIME}(n \log n)$, …

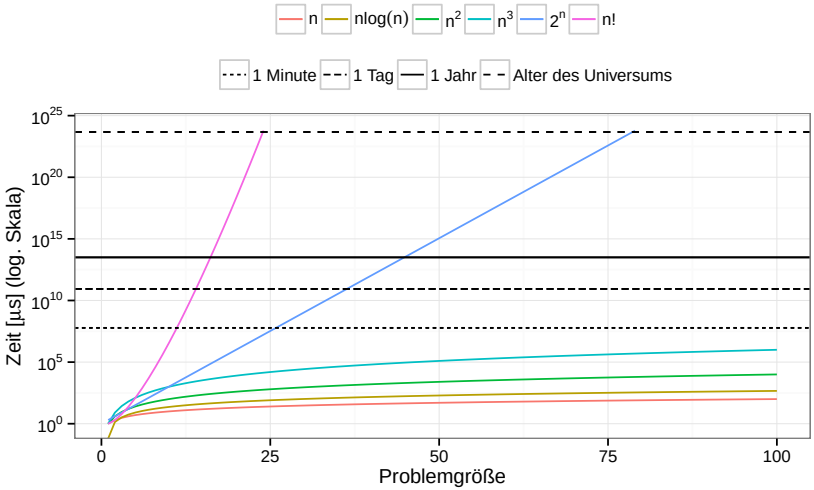
Eigenschaften

- Fast Fourier-Transformation
- Sortieralgorithmen (Merge Sort, Quicksort, Bubblesort, …)
- Binäre Suche
- Primzahl-test
- …

Warum polynomiales Skalierungsverhalten?



Warum polynomiales Skalierungsverhalten?



P und NP

$$\mathbf{P} \equiv \bigcup_k \mathbf{TIME}(n^k) \tag{1}$$

$$\mathbf{NP} \equiv \bigcup_k \mathbf{NTIME}(n^k) \tag{2}$$

Interpretation von NP

- NP bedeutet *nicht* nicht-polynomial!
- Was ist unter nicht-deterministisch polynomial zu verstehen?

P und NP

$$\mathbf{P} \equiv \bigcup_k \mathbf{TIME}(n^k) \tag{1}$$

$$\mathbf{NP} \equiv \bigcup_k \mathbf{NTIME}(n^k) \tag{2}$$

Interpretation von NP

- **NP** bedeutet *nicht* nicht-polynomial!
- Was ist unter nicht-deterministisch polynomial zu verstehen?

Inhalt

Probleme in NP

- k-Färbbarkeit von Graphen (k-Colour)
- Hamilton'scher Pfad (Hampath)
- Erfüllbarkeit logischer Formeln (Sat)
- Graph-Isomorphismus (Graphiso)
- …

Gemeinsamkeiten

- Lösungen schwer zu finden
- Korrektheit bekannter Lösungen leicht überprüfbar (keine selbstverständliche Eigenschaft! Beispiel: $\overline{\text{Hampath}}$)

Achtung!

Enthaltensein in **NP** bedeutet nicht notwendigerweise, dass kein effizienter Algorithmus *existiert*.

Struktur von NP

- Es gibt besonders schwere Probleme in **NP**
- Alle anderen Probleme in **NP** sind auf besonders schwere Varianten effizient zurückführbar

Grundproblem Reduzierbarkeit

- Gegeben: Problem B mit Lösung, Problem A.
- Gesucht: Methode, die mit Lösung von B Lösung von A ermittelt.
- Vorgehensweise:
  - Eingabe von A auf Eingabe von B abbilden.
  - Sicherstellen, dass »Ja« (»Nein«)-Instanzen auf »Ja« (»Nein«)-Instanzen abgebildet werden
- Abbildung muss *effizient* zu berechnen sein!

Definition: Polynomial Zeit-berechenbare Funktion

$f : \Sigma^* \to \Sigma^*$ ist polynomial Zeit-berechenbar, wenn eine polynomial-Zeit DTM M existiert, die für jede Eingabe ω mit $f(\omega)$ als Bandinhalt hält.

Grundproblem Reduzierbarkeit

- Gegeben: Problem B mit Lösung, Problem A.
- Gesucht: Methode, die mit Lösung von B Lösung von A ermittelt.
- Vorgehensweise:
  - Eingabe von A auf Eingabe von B abbilden.
  - Sicherstellen, dass »Ja« (»Nein«)-Instanzen auf »Ja« (»Nein«)-Instanzen abgebildet werden
- Abbildung muss *effizient* zu berechnen sein!

Definition: Polynomial Zeit-berechenbare Funktion

$f : \Sigma^* \to \Sigma^*$ ist polynomial Zeit-berechenbar, wenn eine polynomial-Zeit DTM M existiert, die für jede Eingabe ω mit $f(\omega)$ als Bandinhalt hält.

Definition: Polynomial Zeit-berechenbare Funktion

$f : \Sigma^* \to \Sigma^*$ ist polynomial Zeit-berechenbar, wenn eine polynomial-Zeit DTM M existiert, die für jede Eingabe ω mit $f(\omega)$ als Bandinhalt hält.

Definition: Polynomial-Zeit Abbildungs-Reduzierbarkeit

Sprache A ist *polynomial Zeit-Abbildungs-Reduzierbar (polynomial time mapping reducible, many-to-one-reducible)* auf Sprache B,

$$A \leq_{\mathrm{p}} B,$$

wenn $\exists f : \Sigma^* \to \Sigma^*$, f polynomial Zeit-berechenbar, so dass $\forall w$:

$$w \in A \Leftrightarrow f(w) \in B.$$

Polynomiale Zeit-Reduzierbarkeit II

Satz

Wenn $A \leq_p B$ und $B \in \mathbf{P}$, dann ist $A \in \mathbf{P}$.

Algorithmus

- Gegeben: Eingabe $\omega \in A$, TM M für B.
- Berechne $f(\omega)$
- Führe M auf $f(\omega)$ aus; gibt Resultat zurück.

Anmerkungen

- Komposition zweier Polynome ist weiterhin Polynom
- Funktioniert auch für NTMs

Definition: NP-Vollständigkeit

Eine Sprache B ist NP-vollständig, wenn gilt:

- $B \in \mathbf{NP}$
- $\forall A \in \mathbf{NP} : A \leq_\mathrm{p} B$

NPC ist die Klasse aller **NP**-vollständigen Probleme.

Satz I

$$B \in \mathbf{NPC} \wedge B \in \mathbf{P} \Rightarrow \mathbf{P} = \mathbf{NP}$$

Satz II

Wenn $B \in \mathbf{NPC}$ und $B \leq_\mathrm{p} C$, dann gilt

$$C \in \mathbf{NPC}$$

Satz III (Cook-Levin)

$$\textsc{Sat} \in \mathbf{NPC}$$

NP-Vollständigkeit: Konsequenzen

- Alle **NPC**-Probleme sind gleich schwierig
- **NPC**-Probleme sind *nicht* in **P**, wenn $\mathbf{P} \neq \mathbf{NP}$
- $\exists$ Algorithmus in **P** für **NP**-vollständiges Problem ☞ $\mathbf{P} = \mathbf{NP}$
- Starke aktuelle Vermutung:

$$\mathbf{NP} \neq \mathbf{P}$$

- Preisgeld für Beweis: 1 Million US-Dollar
  - Siehe *Millenium Prize*
  - Theoretische Informatik *ist* kommerziell interessant :)

Liste **NP**-vollständiger bzw. **NP**-harter Probleme

Graphen

Soziale Netzwerke, Internet, Compiler, …

- Graphhomomorphismus (Netzwerke strukturell identisch?)
- Längster Pfad
- Graphfärbbarkeit (Registerallokation)

Optimierung

Produktionsplanung, Scheduling, …

- Bin Packing (Verteilung in Container)
- Travelling Salesman
- Quadratic Assignment (elektronische Komponenten plazieren)

Liste **NP**-vollständiger bzw. **NP**-harter Probleme

Formale Sprachen

Compiler, Big Data, Maschinelles Lernen, …

- Längste gemeinsame Teilsequenz
- String-String-Korrektur

Spiele und Puzzles

Daddeln

- Super Mario Bros (schnellstmöglicher Lauf)
- Mastermind (Lösung finden)
- Lemmings (Level lösen)

Siehe beispielsweise `dimacs.rutgers.edu/~graham/pubs/papers/cormodelemmings.pdf` oder `arxiv.org/pdf/1203.1895v1.pdf`

NP-Vollständigkeit V

Konsequenzen

- Viele praktische Probleme können (sehr wahrscheinlich) nicht effizient gelöst werden
- Approximationsverfahren wichtig
  - Manche Probleme können beweisbar nicht approximiert werden
  - Tritt sehr selten auf
- Details: Siehe Vorlesung »Algorithmen und Datenstrukturen«

Erfüllbarkeit logischer Ausdrücke: SAT

- Boole'sche Variablen: $x_0, x_1, \ldots, x_n$
- Verknüpfungen: *Oder* ($\vee$), *Und* ($\wedge$), *Negation* ($\neg$), Kurzschreibweise: $\bar{x}_i \equiv \neg x_i$
- Klauseln: *Disjunktion* (Oder-Verknüpfung) von Variablen und negierten Variablen. Bespiel:

$$K_1 = (x_3 \vee x_7 \vee \bar{x}_2)$$

- Boolescher Ausdruck: *Konjunktion* (Und-Verknüpfung) von Klauseln. Beispiel:

$$B = K_1 \wedge K_2 \wedge K_3 = (x_3 \vee x_7 \vee \bar{x}_2) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_4 \vee x_6)$$

Konjunktive Normalform

- Konjunktion von Disjunktionen: *Konjunktive Normalform* (KNF, CNF)
- Jede Klausel enthält maximal i Literale ☞ i-CNF

Erfüllbarkeit

- Ausdruck B erfüllt, wenn Belegung der Variablen $x_1, \ldots, x_n$ existiert, die B wahr macht
- Beispiele:
  - $B = (x_0 \vee x_1 \vee x_2)$: ✓
  - $B = (x_0 \vee x_1) \wedge (\bar{x}_0 \vee x_2)$: ✓
  - $B = (x_0 \vee x_1) \wedge \bar{x}_0 \wedge \bar{x}_1$: ✗

Sprache SAT

$$\text{SAT} \equiv \{K \mid K \text{ ist erfüllbarer Boole'scher Ausdruck in KNF}\}$$

Praktische Anwendungen

- Konstringierte Planungsprobleme (Stunden-, Ablaufplanung, …)
- Produktlinien in der Softwarentwicklung
- Konfigurationsoptionen-Konsistenz (bsp. Linux-Kernel)

SAT-Solver: Siehe `www.satcompetition.org`

NP-Vollständigkeit IX: SAT IV

Satz von Cook und Levin

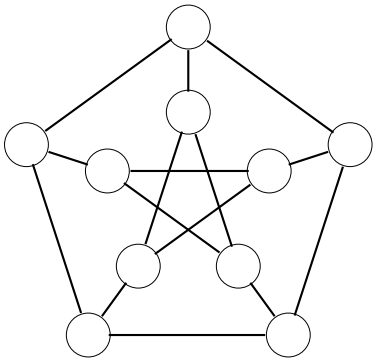
Für jedes Problem $A \in \mathbf{NP}$ gilt

$$A \leq_p \text{SAT}$$

Henne und Ei

- **NP**-Vollständigkeit von SAT *ohne* Reduktion auf andere Probleme in **NP** beweisbar (Illustration: **3-Colour**).
- Startpunkt für **NP**-Vollständigkeits-Beweise

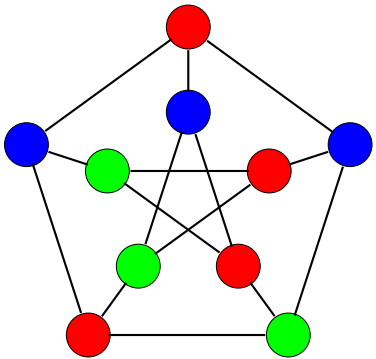
NP-Vollständigkeit X: Drei-Färbbarkeit I



Problemstellung

- Jeder Knoten: Eine Farbe (rot, grün, blau)
- Zwei durch Kante verbundene Knoten: Unterschiedliche Farbe
- Existiert gültige Farbzuweisung?

NP-Vollständigkeit X: Drei-Färbbarkeit I



Problemstellung

- Jeder Knoten: Eine Farbe (rot, grün, blau)
- Zwei durch Kante verbundene Knoten: Unterschiedliche Farbe
- Existiert gültige Farbzuweisung?

Definition Drei-Färbbarkeit

$$\begin{aligned}\text{3-Colour} = \{G = (V, E) \mid G \text{ ist ungerichteter Graph} \\ \wedge\, c : V \to \{r, g, b\}, \\ \forall (u, v) \in E : c(u) \neq c(v)\}\end{aligned}$$

Komplexität von 3-Colour

- Sat ist mindestens so schwierig wie 3-Colour: 3-Colour $\leq_p$ Sat
- 3-Colour ist **NP**-vollständig: Sat $\leq_p$ 3-Colour

Beweis: 3-Colour $\leq_p$ Sat

Knotenfarbe über Variablen festlegen

$$r_i = 1 : \text{ Rot}, g_i = 1 : \text{ Grün}, b_i = 1 : \text{ Blau}.$$

1.) (Mindestens) eine Farbe pro Knoten

$$\alpha_1 = \bigwedge_{i=1}^{n} (r_i \vee g_i \vee b_i)$$

Beweis: 3-Colour $\leq_p$ Sat

Knotenfarbe über Variablen festlegen

$$r_i = 1 : \text{ Rot}, g_i = 1 : \text{ Grün}, b_i = 1 : \text{ Blau}.$$

2.) Kein Knoten hat zwei Farben

$$\begin{aligned}\alpha_2 &= \bigwedge_{i=1}^{n} \left[(\overline{r_i \wedge g_i}) \wedge (\overline{g_i \wedge b_i}) \wedge (\overline{r_i \wedge b_i}) \right] \\ &= \bigwedge_{i=1}^{n} \left[(\bar{r}_i \vee \bar{g}_i) \wedge (\bar{g}_i \vee \bar{b}_i) \wedge (\bar{r}_i \vee \bar{b}_i) \right]\end{aligned}$$

Beweis: 3-Colour $\leq_p$ Sat

Knotenfarbe über Variablen festlegen

$$r_i = 1 : \text{ Rot}, g_i = 1 : \text{ Grün}, b_i = 1 : \text{ Blau}.$$

3.) Knoten an gemeinsamer Kante: Verschiedene Farben

$$\begin{aligned}\alpha_3 &= \bigwedge_{(v_i,v_j)\in E} \left[(\overline{r_i \wedge r_j}) \wedge (\overline{g_i \wedge g_j}) \wedge (\overline{b_i \wedge b_j})\right] \\ &= \bigwedge_{(v_i,v_j)\in E} \left[(\bar{r}_i \vee \bar{r}_j) \wedge (\bar{g}_i \vee \bar{g}_j) \wedge (\bar{b}_i \vee \bar{b}_j)\right]\end{aligned}$$

Beweis: 3-Colour $\leq_p$ Sat

Knotenfarbe über Variablen festlegen

$$r_i = 1 : \text{ Rot}, g_i = 1 : \text{ Grün}, b_i = 1 : \text{ Blau}.$$

Überprüfung durch Sat

- Zu überprüfender Ausdruck: $K = \alpha_1 \wedge \alpha_2 \wedge \alpha_3$
  - Erfüllbar: Färbung existiert
  - Nicht erfüllbar: Färbung existiert nicht
- ☞ 3-Colour ist Spezialfall von Sat!

Definition: Verifizierer V

Gegeben: Wort w, Zertifikate c und Sprache A mit

$$A = \{w \mid V \text{ akzeptiert } \langle w, c\rangle \text{ für ein Zertifikat } c\}.$$

V ist *Verifizierer* für A, dessen Komplexität auf Basis von $|w|$ (*nicht* von c!) gemessen wird.

Definition: NP

NP ist die Klasse aller Sprachen mit Polynomialzeit-Verifizierer.

Zusammenhang

Beide Definitionen von **NP** sind identisch!

Satz: NP-Definitionen sind äquivalent

Es gilt $L \in \mathbf{NP}$ genau dann, wenn es eine DTM M mit polynomialer Laufzeit gibt und

$$L = \{w \mid \text{ es gibt ein } c_w \in \Sigma^*, \text{ so dass } M \text{ das Wort } \langle w, c_w \rangle \text{ akzeptiert}\}.$$

Beweis: Siehe Tafel.

Satz von Cook und Levin

Für jedes Problem A $\in$ **NP** gilt

$$A \leq_p \textsc{Sat}$$

Beweis (Struktur)

- Für jede Sprache A $\in$ **NP**: Reduktion für Maschine $M(A)$ konstruieren
- Für jedes Wort $\omega \in$ A:
  - Boole'sche Formel ϕ konstruieren, die $M(A)$ repräsentiert.
  - ϕ simuliert M mit Eingabe ω
  - Maschine akzeptiert ☞ Formel erfüllbar ☞ $M(A)$ akzeptiert ☞ $\omega \in A$
  - Maschine akzeptiert nicht ☞ $\omega \notin A$
- Beweis: Siehe Tafel.

Entscheidungs- und Optimierungsprobleme I

Problem der/des Handlungsreisenden (TSP)

- Gegeben: n Städte und die Entfernungen dazwischen
- Gesucht: Rundreise durch alle Städte mit minimaler Gesamtlänge
- Optimierungsproblem!

Formulierung als Entscheidungsproblem

- Gegeben: n Städte, Entfernungen dazwischen und ein Wert k
- Gesucht: Gibt es eine Rundreise durch alle Städte mit Länge $\leq k$

Zusammenhang

- Entscheidungsproblem nicht in **P** ☞ Optimierungsproblem nicht in **P**
- Optimierungsproblem nicht leicher als Entscheidungsproblem

Entscheidungs- und Optimierungsprobleme II

Unterschied: Entscheidung vs. Optimierung

- Entscheidung: Ja/Nein-Antwort
  - Mehrere Lösungen: Alle gleich gut
- Optimierung: Minimierung/Maximierung
  - Mehrere Lösungen: Ranking
  - Numerisches Maß mit jeder Lösung verknüpft
  - Suche nach Lösung mit maximalem/minimalem Maß

Entscheidungs- und Optimierungsprobleme II

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem besteht aus

- einer Menge von Eingabeinstanzen (effizient entscheidbar).
- einer Funktion, die Eingabeinstanzen auf Lösungswörter (mit beschränkter Länge) abbildet.
- einer Zielfunktion (*objective function*), die das Maß einer Lösung berechnet
- einem Optimierungsziel (Minimierung/Maximierung)

Entscheidungs- und Optimierungsprobleme II

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- einer Menge von Eingabeinstanzen (effizient entscheidbar).
- einer Funktion, die Eingabeinstanzen auf Lösungswörter (mit beschränkter Länge) abbildet.
- einer Zielfunktion (*objective function*), die das Maß einer Lösung berechnet
- einem Optimierungsziel (Minimierung/Maximierung)

Entscheidungs- und Optimierungsprobleme II

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- $I_F \subseteq \Sigma^*$: In **P** entscheidbare Menge von Eingabeinstanzen.
- einer Funktion, die Eingabeinstanzen auf Lösungswörter (mit beschränkter Länge) abbildet.
- einer Zielfunktion (*objective function*), die das Maß einer Lösung berechnet
- einem Optimierungsziel (Minimierung/Maximierung)

Entscheidungs- und Optimierungsprobleme II

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- $I_F \subseteq \Sigma^*$: In **P** entscheidbare Menge von Eingabeinstanzen.
- $S_F(x) \subseteq \Sigma^*$: Funktion, die alle Lösungen für eine Eingabe $x \in I_F$ berechnet.
- einer Zielfunktion (*objective function*), die das Maß einer Lösung berechnet
- einem Optimierungsziel (Minimierung/Maximierung)

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- $I_F \subseteq \Sigma^*$: In **P** entscheidbare Menge von Eingabeinstanzen.
- $S_F(x) \subseteq \Sigma^*$: Funktion, die alle Lösungen für eine Eingabe $x \in I_F$ berechnet. Anforderungen:
  - $\exists$ Polynom p_F (nur von F abhängig)
  - $\exists\, \pi : I_F \times \Sigma^* \to \{T, F\} \in \mathbf{P}$, das gültige Lösungen identifiziert (nur von F abhängig).

$$S_F(x) \equiv \{y : |y| \leq p_F(|x|) \wedge \pi_F(x, y))\}$$

  S_F ist in **P** entscheidbar.
- einer Zielfunktion (*objective function*), die das Maß einer Lösung berechnet
- einem Optimierungsziel (Minimierung/Maximierung)

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- $I_F \subseteq \Sigma^*$: In **P** entscheidbare Menge von Eingabeinstanzen.
- $S_F(x) \subseteq \Sigma^*$: Funktion, die alle Lösungen für eine Eingabe $x \in I_F$ berechnet. Anforderungen:
  - $\exists$ Polynom p_F (nur von F abhängig)
  - $\exists\, \pi : I_F \times \Sigma^* \to \{T, F\} \in \mathbf{P}$, das gültige Lösungen identifiziert (nur von F abhängig).

$$S_F(x) \equiv \{y : |y| \leq p_F(|x|) \wedge \pi_F(x, y))\}$$

  S_F ist in **P** entscheidbar.
- $m_F : I_F \times \Sigma^* \to \mathbb{N}$ ist die Zielfunktion (*objective function*).
  - $m_F \in \mathbf{FP}$
  - $m_F(x, y)$ nur für $y \in S_F(x)$ definiert.
- einem Optimierungsziel (Minimierung/Maximierung)

Definition: NPO-Optimierungproblem

Ein **NPO**-Optimierungsproblem ist ein 4-Tupel $F = (I_F, S_F, m_F, \text{opt})$ über einem Alphabet Σ:

- $I_F \subseteq \Sigma^*$: In **P** entscheidbare Menge von Eingabeinstanzen.
- $S_F(x) \subseteq \Sigma^*$: Funktion, die alle Lösungen für eine Eingabe $x \in I_F$ berechnet. Anforderungen:
  - $\exists$ Polynom p_F (nur von F abhängig)
  - $\exists\, \pi : I_F \times \Sigma^* \to \{T, F\} \in \mathbf{P}$, das gültige Lösungen identifiziert (nur von F abhängig).

$$S_F(x) \equiv \{y : |y| \leq p_F(|x|) \wedge \pi_F(x, y))\}$$

  S_F ist in **P** entscheidbar.
- $m_F : I_F \times \Sigma^* \to \mathbb{N}$ ist die Zielfunktion (*objective function*).
  - $m_F \in \mathbf{FP}$
  - $m_F(x, y)$ nur für $y \in S_F(x)$ definiert.
- $\text{opt} \in \{\min, \max\}$

Entscheidungs- und Optimierungsprobleme III

Lösung Optimierungsproblem

- Gegeben: Optimierungsproblem $F = (I_F, S_F, m_F, \text{opt})$ über Alphabet Σ
- Optimale Lösung y für Eingabe x:

$$m_{\text{opt}}(x) = \text{opt}\{m_F(x,z) : z \in S_F(x) \wedge \pi_F(x,z) = \text{T}\}$$

Nichtdeterminismus

NTM zur Berechnung von S zulässig!

Definition: PO

- **PO** ist die Unterklasse von **NPO**-Problemen, die auf einer DTM in polynomialer Zeit lösbar sind.
- Triviale Inklusion: **PO** $\subseteq$ **NPO**.

Entscheidungs- und Optimierungsprobleme IV

Beispiel: Minimum Vertex Cover (Min-Vc)

- Entscheidungsproblem k-Vc: Zu ungerichtetem Graph $G = (V, E)$ ist Teilmenge $C \subseteq V$ gesucht, so dass $|C| \leq k$ und $\forall (i, j) \in E : i \in C \vee j \in C$
- Optimierungsproblem: Kleinstes Cover für G gesucht

Beispiele für Vertex Covers: Siehe Tafel

Entscheidungs- und Optimierungsprobleme V

Min-Vc ∈ NPO

Enthaltensein wird in zwei Stufen gezeigt:

1. Min-Vc erfüllt strukturelle Kriterien eines Optimierungsproblems.
2. Alle Ressourcenbeschränkungen sind erfüllt.

1.) Strukturelle Kriterien

- $I_F = \{(V, E)\}$, d.h. Menge der Graphen G.
- $S_F : (V, E) \to \mathcal{P}(V)$, d.h. alle Teilmengen der Knoten sind eine potentielle Abdeckung
- $\forall (V, E) \in I \; \forall C \in \mathcal{P}(V) : \pi((V, E), C) = T$, wenn $\forall (i, j) \in E : i \in C \vee j \in C$. Das Prädikat π ist wahr, wenn eine gegebene Teilmenge der Knoten eine Abdeckung darstellt.
- $m((V, E), C) = |C|$. Als Maß wird die Größe der Abdeckung verwendet.

Entscheidungs- und Optimierungsprobleme V

Min-Vc ∈ NPO

Enthaltensein wird in zwei Stufen gezeigt:

1. Min-Vc erfüllt strukturelle Kriterien eines Optimierungsproblems.
2. Alle Ressourcenbeschränkungen sind erfüllt.

2.) Ressourcenbeschränkungen

- I ist in **P** entscheidbar (simple Überprüfung, ob Element ungerichteten Graph spezifiziert)
- $\exists$ Polynom $p : \forall x \in I \, \forall y \in S(x) : |y| \leq p(x)$. Dies trifft zu, da $\forall C \in \mathcal{P}(V) : |C| \leq |V| \leq |(V, E)|$.
- $y \in S(x)$ ist in deterministischer poly-Zeit entscheidbar, da π in poly-Zeit entscheidbar (über Kanten iterieren und prüfen, ob jeweils mindestens ein Knoten in C enthalten)
- $m \in \mathbf{FP} : \forall C \in \mathcal{P}(V) : |C|$ in poly-Zeit berechenbar (Elemente zählen).

Entscheidungs- und Optimierungsprobleme VI

Zugrundeliegende Sprache eines NPO-Problems

- $L_{\text{opt}} = (I, S, m, \text{opt}) \in \mathbf{NPO}$: Optimierungsproblem. L_{dec}: Zugehöriges Entscheidungsproblem.
- Wenn Instanz x Lösung mit passendem Maß für L_{opt} besitzt ☞ $(x, k) \in L_{\text{dec}}$.

$$L_{\text{dec}} \equiv \{(x, k) \mid \exists x \in I, k > 0 : m(x, y) \,\text{op}\, k\}$$

$$\text{op} = \begin{cases} \geq & \text{für opt} = \max \\ \leq & \text{für opt} = \min \end{cases}$$

Satz: Zusammenhang zwischen NP und NPO

$$L_{\text{opt}} \in \mathbf{NPO} \Rightarrow L_{\text{dec}} \in \mathbf{NP}$$

Beweis: Siehe Literatur

Entscheidungs- und Optimierungsprobleme VII

Folgerungen

- Optimierungsproblem aus **PO** ☞ Entscheidungsproblem in **P**.
- Wenn $\mathbf{P} \neq \mathbf{NP}$ ☞ Jedes Problem in **NPO** mit zugrundeliegendem **NPC**-Entscheidungsproblem ist nicht in **PO**.
- $\mathbf{P} \neq \mathbf{NP} \Rightarrow \mathbf{PO} \neq \mathbf{NPO}$

Beweis: Gesunder Menschenverstand bzw. Literatur.

Entscheidungs- und Optimierungsprobleme VIII

Optimierung in der Praxis

Probleme in **NPO** mit zugrundeliegendem **NP**-vollständigen Entscheidungsproblem können nicht mit praxistauglichem Aufwand gelöst werden. Mögliche Auswege:

- Auf **P** = **NP** hoffen.
- Lösung *approximieren* (siehe SAL).
- Optimum mit Wahrscheinlichkeit $p < 1$ berechnen.

Entscheidungs- und Optimierungsprobleme IX

Nicht-optimale Lösungen: Approximation

- $m_{\text{opt}}(x)$: Optimale Lösung
- $y \in S(x)$: Gültige, möglicherweise nicht optimale Lösung mit Wert
  - $m_F(x, y) < m_{\text{opt}}(x)$ bei opt = max
  - $m_F(x, y) > m_{\text{opt}}(x)$ bei opt = min
- Güte $r(x)$ der Approximation:

$$r(x) \equiv \max\left\{\frac{m(x,y)}{m_{\text{opt}}(x)}, \frac{m_{\text{opt}}(x)}{m(x,y)}\right\}$$

- Globales Kriterium: Schlechteste Approximation aller Eingabewerte x.

Entscheidungs- und Optimierungsprobleme X

Relevante Komplexitätsklassen

- **APX** (*approximable*): Mit konstanter Güte approximierbare Probleme
- **PTAS** (*polynomial time approximation scheme*): Für jede beliebige Approximationsgüte existiert ein Poly-Zeit-Algorithmus (polynomial in Eingabelänge)
- **FPTAS** (*fully polynomial time approximation scheme*): Wie **PTAS**, zusätzlich polynomial in Approximationsgrad

$$\mathbf{PO} \subseteq \mathbf{FPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$$

A compendium of NP optimisation problems

Siehe `http://www.nada.kth.se/~viggo/wwwcompendium/`

Satz von Ladner: Struktur von NP

- Wenn $\mathbf{P} \neq \mathbf{NP}$, dann gibt es (mindestens) ein Problem $X \in \mathbf{NP}$ mit $X \notin \mathbf{P}$ und $X \notin \mathbf{NPC}$
- $\mathbf{P} \neq \mathbf{NP} \Rightarrow \mathbf{NP} - \mathbf{P} \neq \mathbf{NPC}$

Satz von Ladner II

Erfüllbarkeit

Sei x eine aussagenlogische Formel in geeigneter Kodierung. Definiere

$$\textsc{Sat}(x) = \begin{cases} \mathrm{T} & \text{für } x \in \textsc{Sat} \\ \mathrm{F} & \text{für } x \notin \textsc{Sat}. \end{cases}$$

Konstruierte Sprache

Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Definiere

$$A(x) \equiv \begin{cases} \textsc{Sat}(x) & \text{wenn } f(|x|) \text{ gerade} \\ \mathrm{F} & \text{wenn } f(|x|) \text{ ungerade.} \end{cases}$$

Satz von Ladner III

Gezieltes ins-Knie-Schießen

f soll folgende Anforderungen erfüllen:

1. $f(n)$ ist in Zeit poly(n) berechenbar ☞ A $\in$ **NP**
2. Kein Poly-Zeit-Programm Π entscheidet A ☞ $A \notin$ **P**
3. Kein Poly-Zeit-Programm Π reduziert SAT auf A ☞ A $\notin$ **NPC**

Gödelisierung von Programmen

- Programm $\vec{x}$: Zeichenkette $\vec{x} \in \Sigma^*$ mit Σ = ASCII-Code.
- ASCII-Code: In Binär darstellbar ☞ $\exists n \in \mathbb{N} : \vec{x} \in \mathbb{F}_2^n$.
- Programm $\vec{x}$ entspricht Zahl $i = \sum_{k=0}^{n-1} x_k 2^k$ ☞ Π_i ist (deterministisches) Programm mit »Quellcode« i.
- Nicht jede Zahl ist sinnvolles Programm ☞ $\Pi_j \equiv$ `return 'Unfug'` in diesem Fall.

Satz von Ladner III

Gezieltes ins-Knie-Schießen

f soll folgende Anforderungen erfüllen:

1. $f(n)$ ist in Zeit poly(n) berechenbar ☞ A $\in$ **NP**
2. Kein Poly-Zeit-Programm Π entscheidet A ☞ $A \notin$ **P**
3. Kein Poly-Zeit-Programm Π reduziert SAT auf A ☞ A $\notin$ **NPC**

Gödelisierung von Programmen

- Programm $\vec{x}$: Zeichenkette $\vec{x} \in \Sigma^*$ mit Σ = ASCII-Code.
- ASCII-Code: In Binär darstellbar ☞ $\exists n \in \mathbb{N} : \vec{x} \in \mathbb{F}_2^n$.
- Programm $\vec{x}$ entspricht Zahl $i = \sum_{k=0}^{n-1} x_k 2^k$ ☞ Π_i ist (deterministisches) Programm mit »Quellcode« i.
- Nicht jede Zahl ist sinnvolles Programm ☞ $\Pi_j \equiv$ `return 'Unfug'` in diesem Fall.

Satz von Ladner III

Gezieltes ins-Knie-Schießen

f soll folgende Anforderungen erfüllen:

1. $f(n)$ ist in Zeit poly(n) berechenbar ☞ A $\in$ **NP**
2. $\forall i \in \mathbb{N}_0 \exists \vec{y} : \Pi_i(\vec{y}) \neq A(\vec{y})$
3. Kein Poly-Zeit-Programm Π reduziert SAT auf A ☞ A $\notin$ **NPC**

Gödelisierung von Programmen

- Programm $\vec{x}$: Zeichenkette $\vec{x} \in \Sigma^*$ mit Σ = ASCII-Code.
- ASCII-Code: In Binär darstellbar ☞ $\exists n \in \mathbb{N} : \vec{x} \in \mathbb{F}_2^n$.
- Programm $\vec{x}$ entspricht Zahl $i = \sum_{k=0}^{n-1} x_k 2^k$ ☞ Π_i ist (deterministisches) Programm mit »Quellcode« i.
- Nicht jede Zahl ist sinnvolles Programm ☞ $\Pi_j \equiv$ `return 'Unfug'` in diesem Fall.

Satz von Ladner III

Gezieltes ins-Knie-Schießen

f soll folgende Anforderungen erfüllen:

1. $f(n)$ ist in Zeit poly(n) berechenbar ☞ $A \in \mathbf{NP}$
2. $\forall i \in \mathbb{N}_0 \exists \vec{y} : \Pi_i(\vec{y}) \neq A(\vec{y})$
3. $\forall i \in \mathbb{N}_0 \exists \vec{y} : \textsc{Sat}(\vec{y}) \neq A(\Pi_i(\vec{y}))$

Gödelisierung von Programmen

- Programm $\vec{x}$: Zeichenkette $\vec{x} \in \Sigma^*$ mit Σ = ASCII-Code.
- ASCII-Code: In Binär darstellbar ☞ $\exists n \in \mathbb{N} : \vec{x} \in \mathbb{F}_2^n$.
- Programm $\vec{x}$ entspricht Zahl $i = \sum_{k=0}^{n-1} x_k 2^k$ ☞ Π_i ist (deterministisches) Programm mit »Quellcode« i.
- Nicht jede Zahl ist sinnvolles Programm ☞ $\Pi_j \equiv$ `return 'Unfug'` in diesem Fall.

Satz von Ladner IV

```
1: procedure F(n)
2:     if n < 0 then return 0
3:     end if
4:     if F(n − 1) = 2i then                                          ▷ Gerade Zahl
5:         while Weniger als n Zeitschritte verbraucht do
6:             for y⃗ ∈ Σ* do                                          ▷ Iteration in lexikographischer Ordnung
7:                 if Π_i(y⃗) ≠ A(y⃗) then
8:                     return F(n − 1) + 1
9:                 end if
10:            end for
11:        end while
12:        return F(n − 1)                                            ▷ Uhr abgelaufen
13:    else                                                           ▷ Ungerade Zahl: F(n − 1) = 2i + 1
14:        while Weniger als n Zeitschritte verbraucht do
15:            for y⃗ ∈ Σ* do                                          ▷ Iteration in lexikographischer Ordnung
16:                if SAT(y⃗) ≠ A(Π_i(y⃗)) then
17:                    return F(n − 1) + 1
18:                end if
19:            end for
20:        end while
21:        return F(n − 1)                                            ▷ Uhr abgelaufen
22:    end if
23: end procedure
```

Satz von Ladner V

Funktionsweise von $f(n)$

- Wenn $f(n-1)$ *gerade*: Zeitbeschränkte Suche nach $\vec{y}$ mit

$$\Pi_i(\vec{y}) \neq A(\vec{y})$$

  - Nicht erfolgreich: Wert $f(n)$ bleibt gleich $f(n-1)$
  - Erfolgreich: Unterschiedliche Instanz gefunden ☞
    - Bedingung 2 für i erfüllt
    - Übergang zur nächsten Zahl
- Wenn $f(n-1)$ *ungerade*: Zeitbeschränkte Suche nach $\vec{y}$ mit

$$\textsc{Sat}(\vec{y}) \neq A(\Pi_i(\vec{y}))$$

  - Nicht erfolgreich: Wert $f(n)$ bleibt gleich $f(n-1)$
  - Erfolgreich: Unterschiedliche Instanz gefunden ☞
    - Bedingung 3 für i erfüllt
    - Übergang zur nächsten Zahl

Satz von Ladner VI

Beobachtungen

- Wachsendes n ☞ Mehr Aufwand bei $\vec{y}$-Suche
- Wenn $f(n)$ alle Werte produziert ☞ Bedingungen 2 und 3 global korrekt
- Verzögerte Diagonalisierung (*lazy diagonalisation*)

Noch zu zeigen

1. Ist $f(n)$ in Poly-Zeit berechenbar?
2. Produziert f wirklich alle Werte?

Satz von Ladner VI

Noch zu zeigen

1. Ist $f(n)$ in Poly-Zeit berechenbar?
2. Produziert f wirklich alle Werte?

$f \in$ FP

- Für jedes $\vec{y}$: poly($|\vec{y}|$) Schritte zur Berechnung von $\Pi_i(\vec{y})$.
- Berechnung SAT, A: Superpolynomial, aber Zeitbegrenzung!
- Berechnung $f(n)$ hängt nur von $f(|\vec{y}|)$ ab mit $|\vec{y}| = \mathcal{O}(\log(n))$ (vgl. Anzahl Strings, die in n Schritten prüfbar) ☞ Keine zirkuläre Definition, Berechnung in **FP**

Satz von Ladner VI

Noch zu zeigen

1. Ist $f(n)$ in Poly-Zeit berechenbar?
2. Produziert f wirklich alle Werte?

$f(n)$ produziert alle Werte

- Annahme: $f(n)$ konstant ab *geradem* Wert

$$\exists n_0 \in \mathbb{N} : f(n) = 2i \; \forall n \geq n_0$$

- Zeitlimit tritt immer ein
- $\Pi_i(\vec{y}) = A(\vec{y})$ für alle $\vec{y}$ ☞ Π_i Poly-Zeit-Algorithmus für A
- Definition von A ☞ A und SAT nur in endlichen vielen Instanzen unterschiedlich ☞ Effiziente Lösung für SAT
  - Lookup-Tabelle für $n < n_0$ verwenden
  - Ansonsten Π_i in Poly-Zeit berechnen, um A und damit SAT zu lösen
  - Da SAT $\in$ **NPC** ☞ Widerspruch!

Satz von Ladner VI

Noch zu zeigen

1. Ist $f(n)$ in Poly-Zeit berechenbar?
2. Produziert f wirklich alle Werte?

$f(n)$ produziert alle Werte

- Annahme: $f(n)$ konstant ab *ungeradem* Wert

$$\exists n_0 \in \mathbb{N} : f(n) = 2i + 1 \; \forall n \geq n_0$$

- Zeitlimit tritt immer ein
- $\text{SAT}(\vec{y}) = A(\Pi_i(\vec{y}))$ für alle $\vec{y}$
- Poly-Zeit Reduktion $\text{SAT} \leq_p \text{A}$
- $\text{A}(\vec{x}) = \text{F}$ für alle $|\vec{x}| \geq n_0$, also endliche Anzahl Ja-Instanzen
  - Lookup-Tabelle für $n < n_0$ verwenden
  - $\text{A} \in \mathbf{P}$ ☞ $\text{SAT} \in \mathbf{P}$ ☞ Widerspruch!

Satz von Ladner VI

Konsequenzen

- Struktur des Problems nicht unbedingt praxisrelevant
- Beweis für $X \in \mathbf{NPI}$ bislang für nicht-pathologische X gescheitert
- Kandidatenliste dünn, Tendenz: Abnehmend
  - Isomorphism (Graphen, Gruppen), Discrete Log, Factoring
  - Frühere Kandidaten: Primes (in **P**, Max 2-Sat (in **NPC**), Linear Programming (in **P**)

Beispiel: Siehe Rechner

Minesweeper ist NP-vollständig II

Instanz

2	?	2	2	?	2	?	?	2
?	3	4	?	?	2	3	?	2
?	?	?	?	3	1	2	?	1
?	2	2	2	?	?	2	2	2
2	2		1	?	?	3	?	?
?	3	2	2	2	?	2	3	?
2	?	?	1	1	?	1	1	1
1	2	2	2	1	1	1	?	1
?	?	?	?	?	1	1	?	1

Zuweisung

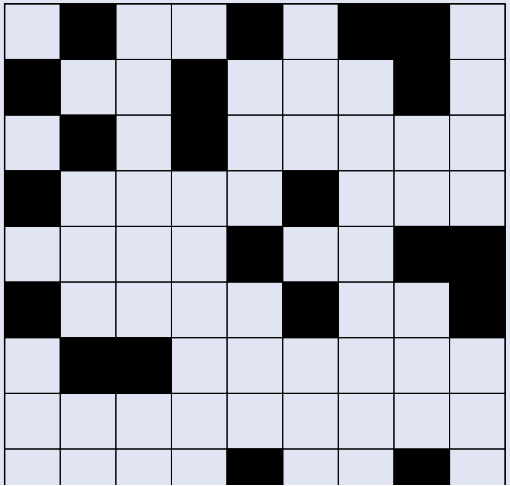
Existiert Zuweisung von Minen auf das Spielfeld, die kompatibel mit Beschriftung ist?

Minesweeper ist NP-vollständig II

Instanz

2	?	2	2	?	2	?	?	2
?	3	4	?	?	2	3	?	2
?	?	?	?	3	1	2	?	1
?	2	2	2	?	?	2	2	2
2	2		1	?	?	3	?	?
?	3	2	2	2	?	2	3	?
2	?	?	1	1	?	1	1	1
1	2	2	2	1	1	1	?	1
?	?	?	?	?	1	1	?	1

Zuweisung



Offline-Minesweeper

Gegeben ein Minesweeper-Spielfeld: Existiert eine Minenkonfiguration, die kompatibel zur bislang bekannten Belegung ist?

Formale Definition

- Minesweeper-Instanz gegeben durch eine $n \times m$-Matrix über dem Alphabet $\{0, 1, \ldots, 8, ?\}$
- *Zuweisung* $\alpha : \mathbb{N} \times \mathbb{N} \to \Gamma$ über $\Gamma = \{0, \mathrm{M}\}$ mit

$$\alpha(x_{i,i}) \equiv \alpha(i, j) = y$$

mit $y \in \Gamma$ und für alle $x_{i,j}$ einer Instanz I.

Lösung

Eine Zuweisung erfüllt/löst eine Instanz I genau dann, wenn folgende Bedingungen gelten:

1. Für alle $x \in I$: Wenn $\alpha(x) = \mathrm{M}$, dann gilt $x = ?$
2. Sei $\delta : \Gamma \to [0,1]$ mit $\delta(0) = 0$ und $\delta(\mathrm{M}) = 1$. Für alle $x \in I$ mit $x \in [0,8]$ gilt

$$\sum_{y \in \mathcal{N}_8(\alpha(x))} \delta(y) = k.$$

Instanz I lösbar $\Leftrightarrow$ Zuweisung α existiert, die I löst.

Minesweeper ∈ NPC

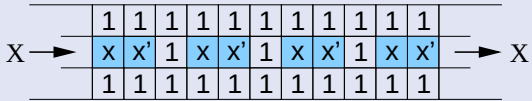
1. Minesweeper ∈ **NP** (siehe Übung)
2. Circuit Sat $\leq_p$ Minesweeper. Siehe Tafel.
   - Elementare Gatter als Minesweeper-Elemente
   - Strukturelle Elemente als Minesweeper-Elemente

Boole'sche Schaltkreise

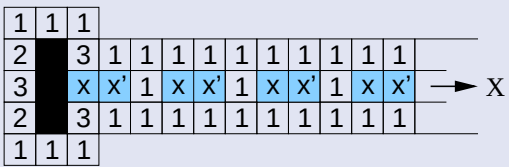
- Elektronische Schaltkreise: Wenigen Elementargatter als Basis (beispielsweise $\{\wedge, \neg\}$ oder $\{\vee, \neg\}$).
- Logische Ausdrück: Schaltkreise (Beispiel: Siehe Tafel)
- Circuit Sat: Ist Bool'scher Schaltkreis $G = (V, E)$ erfüllbar, d.h. gibt es eine Eingabekonfiguration, so dass der Schaltkreis ein wahres Ergebnis ausgibt?
- Circuit Sat ∈ **NPC** (formaler Beweis: Siehe Literatur)

Minesweeper ist NP-vollständig VI

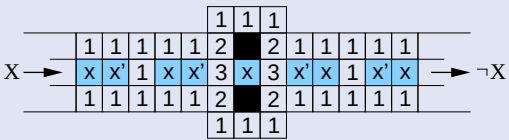
Draht



Endstück

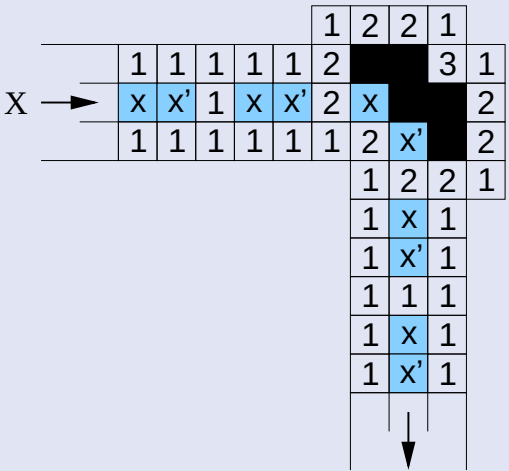


NOT-Gatter

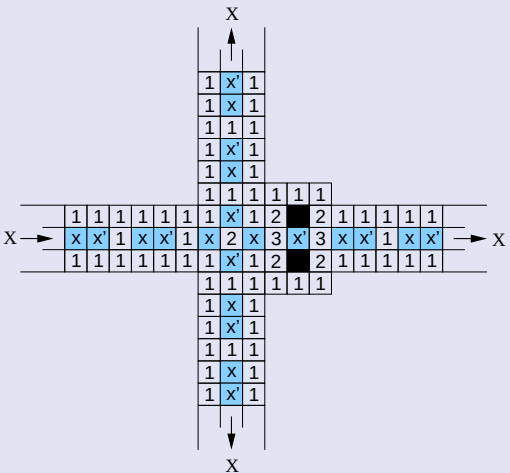


Minesweeper ist NP-vollständig VII

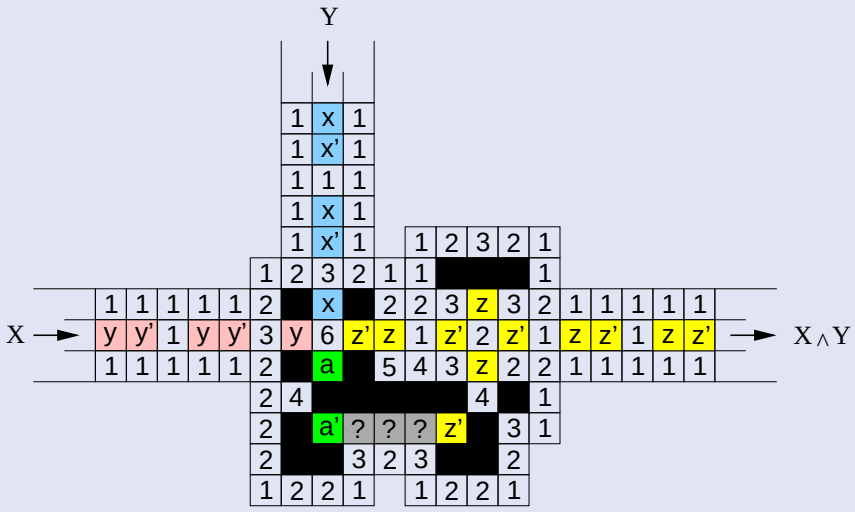
Kurve (aka MS Reib'm)



Splitter



AND-Gatter



Minesweeper ist NP-vollständig IX

Status Quo

- Alle Elementargatter und Leitungen repräsentierbar.
- Problem: Minesweeper inhärent zweidimensional!
  - Keine überkreuzenden Leitungen ☞ Planare Graphen (Digraph G ist planar, wenn er in einer Ebene ohne überkreuzende Kanten dargestellt werden kann)
  - Lösung: Siehe Tafel

Resultat

- Jeder Boole'sche Schaltkreis C durch Minesweeper-Konfiguration repräsentierbar
- Transformation erfordert polynomialen Aufwand
  - Jedes Schaltelement mit k^2 Elementen darstellbar
  - Anzahl Komponenten skaliert polynomial mit Knoten und Kanten von C.

Kurze Durchsage von Donald Knuth

Welche Adjektive x klingen gut in

- »SAT is x«
- »It is x to decide whether a given graph has a Hamiltonian cycle.«
- »It is unknown whether FROB is an x problem.«

Vorschläge: $x \in$ {Hard, Tough, Herculean, Formidable, Arduous}.

Community-Vorschläge

- *Hard-Boiled* (Ken Steiglitz)
- *Hard-Ass* (Al Meyer)
- *Exparent* (Mike Paterson)
- *Supersat* (Al Meyer)
- *PET* (Shen Lin)

Kurze Durchsage von Donald Knuth

Welche Adjektive x klingen gut in

- »SAT is x«
- »It is x to decide whether a given graph has a Hamiltonian cycle.«
- »It is unknown whether FROB is an x problem.«

Vorschläge: $x \in$ {Hard, Tough, Herculean, Formidable, Arduous}.

Community-Vorschläge

- *Hard-Boiled* (Ken Steiglitz) ☞ Tribut an Cook
- *Hard-Ass* (Al Meyer) ☞Hard as Satisfiability
- *Exparent* (Mike Paterson) ☞exponential + apparent
- *Supersat* (Al Meyer) ☞greater than or equal to satisfiability
- *PET* (Shen Lin) ☞Probably exponential time *oder* provably exponential time/previously exponential time

Offene Forschungsfragen

- $\mathbf{P} \subseteq \mathbf{NP}$, aber $\mathbf{NP} \subseteq \mathbf{P}$?
- Existiert ein Problem in **NP**, das nicht in **P** liegt?
- Können andere physikalische Rechnermodelle (z.B. Quantencomputer) helfen?

Interessante Probleme

- Welche Komplexitätsklassen spiegeln die Realität bestmöglich wieder?
- Wann sind Approximationslösungen ausreichend?
- Was passiert, wenn $\mathbf{P} = \mathbf{NP}$ sein sollte?

Inhalt

Anwendung: Randomisierte Algorithmen

Möglichkeiten

- Einfach zu implementieren
- Effizient zu berechnen
- Oft geeignet für Parallelisierung
- Ermöglichen Symmetriebrechung in ansonsten deterministischen Systemen (vor allem Kommunikation)
- Realistische(re) Einschätzung von Komplexität: NP verliert (oft) seinen Schrecken

Schwierigkeiten

- Reproduzierbarkeit (Fehlersuche und Testen)
- Keine Effizienzgarantie
- Mehr Fehlerfälle
- Unübliches Denkmuster

Anwendung: Randomisierte Algorithmen

Möglichkeiten

- Einfach zu implementieren
- Effizient zu berechnen
- Oft geeignet für Parallelisierung
- Ermöglichen Symmetriebrechung in ansonsten deterministischen Systemen (vor allem Kommunikation)
- Realistische(re) Einschätzung von Komplexität: NP verliert (oft) seinen Schrecken

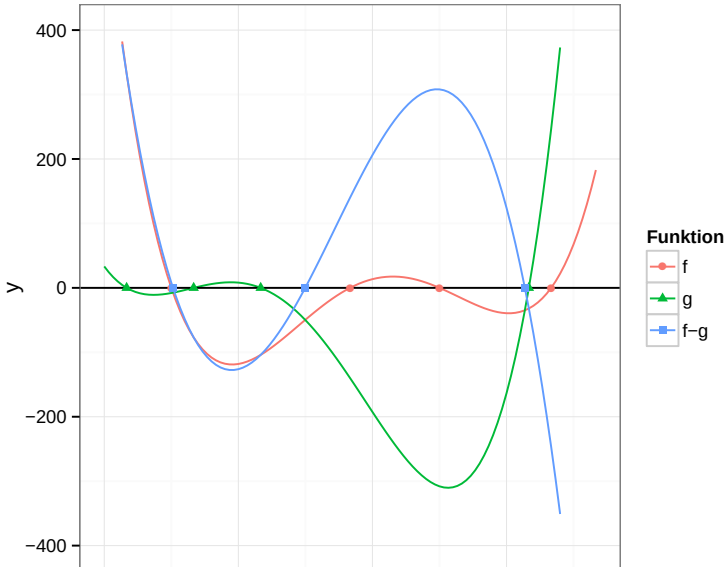
Schwierigkeiten

- Reproduzierbarkeit (Fehlersuche und Testen)
- Keine Effizienzgarantie
- Mehr Fehlerfälle
- Unübliches Denkmuster

Themenüberblick

- Polynom-Gleichheit, Verifikation der Matrixmultiplikation (Fingerprinting), Min-Cut
- Spezielle Derandomisierung von Algorithmen: Large Cuts, Max-SAT

Polynom-Gleichheit



Polynom-Gleichheit I

Zwei Polynome

$$f(x) = (x+1)(x-2)(x+3)(x-4)(x+5)(x-6)$$
$$g(x) = x^6 - 7x^3 + 25$$

Fragestellung

- Gilt $f(x) = g(x)\ \forall x$?
- Was passiert, wenn f oder g nur als Black Box vorhanden sind?

Orthodoxe Ansätze

- $f = g \Leftrightarrow f - g = 0\ \forall x$
- Algebra: Polynom von Grad n hat maximal n Nullstellen.
- $f = \prod_{i=1}^{d}(x - a_i)$ in kanonische Form bringen: $\Theta(d^2)$ Multiplikationen
- Mit FFT: $\mathcal{O}(n \log n)$

Polynom-Gleichheit I

Zwei Polynome

$$f(x) = (x+1)(x-2)(x+3)(x-4)(x+5)(x-6)$$
$$g(x) = x^6 - 7x^3 + 25$$

Fragestellung

- Gilt $f(x) = g(x)\ \forall x$?
- Was passiert, wenn f oder g nur als Black Box vorhanden sind?

Orthodoxe Ansätze

- $f = g \Leftrightarrow f - g = 0\ \forall x$
- Algebra: Polynom von Grad n hat maximal n Nullstellen.
- $f = \prod_{i=1}^{d}(x - a_i)$ in kanonische Form bringen: $\Theta(d^2)$ Multiplikationen
- Mit FFT: $\mathcal{O}(n \log n)$

Polynom-Gleichheit I

Zwei Polynome

$$f(x) = (x+1)(x-2)(x+3)(x-4)(x+5)(x-6)$$
$$g(x) = x^6 - 7x^3 + 25$$

Fragestellung

- Gilt $f(x) = g(x)\ \forall x$?
- Was passiert, wenn f oder g nur als Black Box vorhanden sind?

Randomisierter Ansatz

- Bebobachtung: f und g können in $\mathcal{O}(d)$ an Position r ausgewertet werden (Horner-Schema)
- Wähle Zahl $r \in [1, 100d]$, berechne $f(r)$ und $g(r)$

Rechnung

- Fall 1, $f = g$: $f(r) - g(r) = 0 \; \forall r$
- Fall 2, $f \neq g$: $f - g$ ist Polynom mit maximal d Nullstellen

Nullstellen-Wahrscheinlichkeit

$$p(f(r) = g(r) | f(x) \neq g(x)) = \frac{d}{100d} = \frac{1}{100}.$$

Analyse

	$f = g$	$f \neq g$
$f(r) = g(r)$	✓	✗
$f(r) \neq g(r)$	/	✓

Konsequenz

- Fall 2: Immer korrekt
- Fall 1: Fehler möglich

Polynom-Gleichheit II

Rechnung

- Fall 1, $f = g$: $f(r) - g(r) = 0 \; \forall r$
- Fall 2, $f \neq g$: $f - g$ ist Polynom mit maximal d Nullstellen

Nullstellen-Wahrscheinlichkeit

$$p(f(r) = g(r) | f(x) \neq g(x)) = \frac{d}{100d} = \frac{1}{100}.$$

Analyse

	$f = g$	$f \neq g$
$f(r) = g(r)$	✓	✗
$f(r) \neq g(r)$	/	✓

Konsequenz

- Fall 2: Immer korrekt
- Fall 1: Fehler möglich

Polynom-Gleichheit II

Rechnung

- Fall 1, $f = g$: $f(r) - g(r) = 0 \; \forall r$
- Fall 2, $f \neq g$: $f - g$ ist Polynom mit maximal d Nullstellen

Nullstellen-Wahrscheinlichkeit

$$p(f(r) = g(r) | f(x) \neq g(x)) = \frac{d}{100d} = \frac{1}{100}.$$

Analyse

	$f = g$	$f \neq g$
$f(r) = g(r)$	✓	✗
$f(r) \neq g(r)$	/	✓

Konsequenz

- Fall 2: Immer korrekt
- Fall 1: Fehler möglich

Polynom-Gleichheit III

Klassisch

- Naiv $\Theta(d^2)$
- Raffinierter $\mathcal{O}(d \log d)$

Randomisiert

- 1 randomisierte Auswahl $r \in [1, 100d]$
- 2 Funktionsauswertungen à $\mathcal{O}(d)$
- Algorithmus kann inkorrektes Resultat berechnen!

Polynom-Gleichheit IV: Fehlerw'keit verkleinern

Nullstellen-Wahrscheinlichkeit

$$p(f(r) = g(r) | f(x) \neq g(x)) = \frac{d}{100d} = \frac{1}{100}.$$

Verbesserung der Fehlerwahrscheinlichkeit

1. Wähle r aus größerem Intervall: $\frac{d}{5000d} = \frac{1}{5000}$
2. k-fache Ausführung:

$$p_{\text{korrekt}}(k) = 1 - \left(\frac{1}{100}\right)^k = 1 - 100^{-k}$$

Erfolgsw'keit konvergiert exponentiell gegen 1

Theorem: Schwartz-Zippel

Für ein multivariates Polynom $P(x_1, x_2, \ldots, x_n)$ gilt, wenn $r_1, r_2, \ldots, r_n$ uniform aus $\mathbb{F}_p$ ausgewählt werden, dass:

$$\mathbb{P}(P(r_1, r_2, \ldots, r_n) = 0) \leq \frac{\deg(P)}{|\mathbb{F}_p|}.$$

Stand der Forschung

- Effiziente Lösung für Problem *Polynomial Identity Testing* (PIT) ist deterministisch unbekannt!

Anwendungen

- Perfect Matching
- Min-Cut

Umwandlung in deterministischen Algorithmus

- Maximal d Nullstellen $\Rightarrow d+1$ Versuche ausführen!
- Nach $d+1$ Versuchen ohne Zurücklegen: $p(f(r) = g(r) | f(x) \neq g(x)) = 1$ (d.h. r wird sicher gefunden)
- Allerdings: $d+1$-fache Ausführung eines Algorithmus mit Laufzeit $\mathcal{O}(d) \Rightarrow \mathcal{O}(d^2)$
- Nicht besser als klassischer Ansatz!
- Dennoch: Probabilistische Analyse $\Rightarrow$ Neuer Blickwinkel auf Problem
- Derandomisierung immerhin *effizient* (nicht selbstverständlich!)

Umwandlung in deterministischen Algorithmus

- Maximal d Nullstellen $\Rightarrow d + 1$ Versuche ausführen!
- Nach $d + 1$ Versuchen ohne Zurücklegen: $p(f(r) = g(r) | f(x) \neq g(x)) = 1$ (d.h. r wird sicher gefunden)
- Allerdings: $d + 1$-fache Ausführung eines Algorithmus mit Laufzeit $\mathcal{O}(d) \Rightarrow \mathcal{O}(d^2)$
- Nicht besser als klassischer Ansatz!
- Dennoch: Probabilistische Analyse $\Rightarrow$ Neuer Blickwinkel auf Problem
- Derandomisierung immerhin *effizient* (nicht selbstverständlich!)

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Alle möglichen Elementar-Ergebnisse eines Zufallsprozesses
2. Erlaubte Ereignisse, zusammengesetzt aus Ergebnissen
3. Eine Zuordnung von W'keiten zu Ereignissen

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. Jedes Ereignis hat eine W'keit zwischen 0 und 100%
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Erlaubte Ereignisse, zusammengesetzt aus Ergebnissen
3. Eine Zuordnung von W'keiten zu Ereignissen

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. Jedes Ereignis hat eine W'keit zwischen 0 und 100%
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Eine Zuordnung von W'keiten zu Ereignissen

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. Jedes Ereignis hat eine W'keit zwischen 0 und 100%
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. Jedes Ereignis hat eine W'keit zwischen 0 und 100%
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. Jedes Ereignis hat eine W'keit zwischen 0 und 100%
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. $\forall E \in \Omega : 0 \leq \mathbb{P}(E) \leq 1$
2. W'keit aller Ergebnisse summiert sich zu 100%
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. $\forall E \in \Omega : 0 \leq \mathbb{P}(E) \leq 1$
2. $\mathbb{P}(\Omega) = 1$
3. W'keiten disjunkter Ereignisse addieren sich

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Definition: Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

1. $\forall E \in \Omega : 0 \leq \mathbb{P}(E) \leq 1$
2. $\mathbb{P}(\Omega) = 1$
3. Für paarweise disjunkte $E_1, E_2, \ldots$ gilt

$$\mathbb{P}\left(\bigcup_i E_i\right) = \sum_i \mathbb{P}(E_i)$$

Definition: Wahrscheinlichkeitsraum $\Omega, \mathcal{F}, \mathbb{P}$

1. Nichtleerer Ergebnisraum Ω
2. Eine Menge $\mathcal{F}$ der erlaubten Ereignisse, wobei $\mathcal{F} \ni E_i \subseteq \Omega \; \forall i$
3. Ein Wahrscheinlichkeitsmaß $\mathbb{P} : \mathcal{F} \to \mathbb{R}$

Beschränkung: Diskreter Wahrscheinlichkeitsraum

Endliches und abzählbar unendliches Ω für unsere Zwecke ausreichend

Anwendung auf Polynom-Gleichheit

- Elementarergebnis: Alle Zahlen $r \in \mathbb{K}$. Achtung:
  - $\mathbb{K} = \mathbb{R}$ problematisch
  - $\mathbb{K} = \mathbb{N}$ abgedeckt, $\mathbb{K} = \mathbb{F}_p$ dito
- Auswertung über $\Omega = \mathbb{F}_p$: $\mathbb{P}(\Omega) = 1$, $\mathbb{P}(E_i) = 1/100d$

Schnitt- und Vereinigungsmenge

- $E_1 \cap E_2$: Beide Ereignisse gleichzeitig
- $E_1 \cup E_2$: Mindestens eines der Ereignisse
- $E_1 - E_2$: Ereignis aus E_1, das nicht in E_2 vorkommt
- $\bar{E} = \Omega - E$: Gegenereignis

$$\mathbb{P}(E_1 \cup E_2) = \mathbb{P}(E_1) + \mathbb{P}(E_2) - \mathbb{P}(E_1 \cap E_2)$$

Theorem: Bonferroni-Ungleichung (Union Bound)

Für jede endliche oder abzählbar unendliche Reihe von Ereignissen $E_1, E_2, \ldots$ gilt

$$\mathbb{P}\left(\bigcup_i E_i\right) \leq \sum_i \mathbb{P}(E_i)$$

Definition: Unabhängige Ereignisse

Ereignisse $E_i, i \in I$ sind *unabhängig* genau dann, wenn für jede Teilmenge $I \subseteq [1,k]$ gilt

$$\mathbb{P}\left(\bigcap_{i\in I} E_i\right) = \prod_{i\in I} \mathbb{P}(E_i).$$

Definition: Bedingte Wahrscheinlichkeit

Die Wahrscheinlichkeit, dass Ereignis E eintritt, wenn Ereignis F bereits eingetreten ist, ist

$$\mathbb{P}(E|F) = \frac{\mathbb{P}(E \cap F)}{\mathbb{P}(F)}.$$

Sampling bei Polynom-Gleichheit

Sei E_i das Ereignis, dass beim i-ten Lauf eine Nullstelle an Position r_i (d.h. $f(r_i) - g(r_i) = 0$) angetroffen wird. Fehlerwahrscheinlichkeit nach k Wiederholungen

- mit Zurücklegen:

$$\begin{aligned}\mathbb{P}(\text{»Fehler«}) &= \mathbb{P}(E_1 \cap E_2 \cap \ldots \cap E_k) \\ &= \prod_{i=1}^{k} \mathbb{P}(E_i) \leq \prod_{i=1}^{k} \frac{d}{100d} = \left(\frac{1}{100}\right)^k\end{aligned}$$

- ohne Zurücklegen:

$$\mathbb{P}(\text{»Fehler«}) \leq \prod_{j=1}^{k} \frac{d-(j-1)}{100d-(j-1)} < \left(\frac{1}{100}\right)^k$$

Beweis: Siehe Tafel

Verfahren

$\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$, $\mathbf{A} = \mathbf{B} \cdot \mathbf{C}$:

$$A_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

i, k = (Zeile, Spalte)

Grenzen

- Naiv $\Theta(n^3)$
- Untere Schranke $\Omega(n^2)$ (verfeinert: $\Omega(n^2 \log n)$)

Algorithmen

- Strassen (1969): $\mathcal{O}(n^{\log_2 7}) \approx \mathcal{O}(n^{2.807})$
- Coopersmith-Winograd (1990): $\mathcal{O}(n^{2.3754})$
- Stothers/Williams (2011/12): $\mathcal{O}(n^{2.3736})$, $\mathcal{O}(n^{2.3727})$

Implementierungskomplexität

- Naiv: Drei verschachtelte Schleifen
- Moderne Verfahren: > 50 Seiten zur Beschreibung
- Verifikation sinnvoll
- Safety: Unabhängige Algorithmen uU zwingend erforderlich, Zeitverlust dennoch unerwünscht

Matrizen versus Vektoren

- Matrix-Matrix: $\Theta(n^\omega), \omega > 2$
- Matrix-Vektor: $\Theta(n^2)$
- Grundidee: $(\mathbf{A} \cdot \mathbf{B})\vec{r} = \mathbf{C}\vec{r}\ \forall r \in \mathbb{R}^n$, wenn $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$
- Bezeichnung: Freivalds-Methode

Theorem

Wenn $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$ und $\vec{r}$ uniform zufällig aus $\mathbb{F}_2^n$ gewählt wird, gilt

$$\mathbb{P}(\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}) \leq \frac{1}{2}$$

Beweis: Siehe Tafel

Lemma

Uniforme, zufällige Wahl von $\vec{r} = (r_1, r_2, \ldots, r_n)$ ist äquivalent dazu, r_i unabhängig, uniform zufällig zu wählen

Matrizen versus Vektoren

- Matrix-Matrix: $\Theta(n^\omega), \omega > 2$
- Matrix-Vektor: $\Theta(n^2)$
- Grundidee: $\mathbf{A} \cdot (\mathbf{B}\vec{r}) = \mathbf{C}\vec{r}\ \forall r \in \mathbb{R}^n$, wenn $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$
- Bezeichnung: Freivalds-Methode

Theorem

Wenn $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$ und $\vec{r}$ uniform zufällig aus $\mathbb{F}_2^n$ gewählt wird, gilt

$$\mathbb{P}(\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}) \leq \frac{1}{2}$$

Beweis: Siehe Tafel

Lemma

Uniforme, zufällige Wahl von $\vec{r} = (r_1, r_2, \ldots, r_n)$ ist äquivalent dazu, r_i unabhängig, uniform zufällig zu wählen

Matrizen versus Vektoren

- Matrix-Matrix: $\Theta(n^\omega), \omega > 2$
- Matrix-Vektor: $\Theta(n^2)$
- Grundidee: $\mathbf{A} \cdot (\mathbf{B}\vec{r}) = \mathbf{C}\vec{r}\ \forall r \in \mathbb{F}_2^n$, wenn $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$
- Bezeichnung: Freivalds-Methode

Theorem

Wenn $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$ und $\vec{r}$ uniform zufällig aus $\mathbb{F}_2^n$ gewählt wird, gilt

$$\mathbb{P}(\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}) \leq \frac{1}{2}$$

Beweis: Siehe Tafel

Lemma

Uniforme, zufällige Wahl von $\vec{r} = (r_1, r_2, \ldots, r_n)$ ist äquivalent dazu, r_i unabhängig, uniform zufällig zu wählen

Matrizen versus Vektoren

- Matrix-Matrix: $\Theta(n^\omega), \omega > 2$
- Matrix-Vektor: $\Theta(n^2)$
- Grundidee: $\mathbf{A} \cdot (\mathbf{B}\vec{r}) = \mathbf{C}\vec{r}\ \forall r \in \mathbb{F}_2^n$, wenn $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$
- Bezeichnung: Freivalds-Methode

Theorem

Wenn $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{C}$ und $\vec{r}$ uniform zufällig aus $\mathbb{F}_2^n$ gewählt wird, gilt

$$\mathbb{P}(\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}) \leq \frac{1}{2}$$

Beweis: Siehe Tafel

Lemma

Uniforme, zufällige Wahl von $\vec{r} = (r_1, r_2, \ldots, r_n)$ ist äquivalent dazu, r_i unabhängig, uniform zufällig zu wählen

Law of total probability

Seien $E_1, E_2, \ldots, E_n$ gegenseitig disjunkte Ereignisse aus Ω, und sei $\cup_{i=1}^{n} E_i = \Omega$. Dann gilt

$$\mathbb{P}(B) = \sum_{i=1}^{n} \mathbb{P}(B \cap E_i) = \sum_{i=1}^{n} \mathbb{P}(B|E_i)\mathbb{P}(E_i).$$

Beweis: Siehe Tafel.

Anwendung: Matrixmultiplikation

$$\mathbb{P}(\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}) = \sum_{\{x_2,\ldots,x_n\} \in \mathbb{F}_2^{n-1}} \mathbb{P}([\mathbf{AB}\vec{r} = \mathbf{C}\vec{r}] \cap [(r_2, \ldots, r_n) = (x_2, \ldots, x_n)]) \leq \frac{1}{2}$$

Beweis: Siehe Tafel.

Matrixmultiplikation: Zusammenfassung

Klassisch

- Multiplikation = Verifikation
- Komplexität $\approx \mathcal{O}(n^{2.37})$

Technik: Fingerprinting

- Problemraum (drastisch) verkleinern
- Identitäten bewahren
- $\Theta(kn^2)$ für Fehlerw'keit 2^{-k}

Randomisiert

- $\mathbf{A} \cdot (\mathbf{B}\vec{r}) = \mathbf{C}\vec{r}\ \forall r \in \mathbb{F}_2^n$
- Wenn $\mathbf{D} \equiv \mathbf{AB} - \mathbf{C} \neq \mathbf{0}$, aber $\mathbf{D}\vec{r} = 0$:

$$\sum_{j-1}^{n} d_{1j} r_j = 0 \Rightarrow r_1 = -\frac{\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}$$

- Prinzip der aufgeschobenen Entscheidungen

Matrixmultiplikation: Zusammenfassung

Klassisch

- Multiplikation = Verifikation
- Komplexität $\approx \mathcal{O}(n^{2.37})$

Technik: Fingerprinting

- Problemraum (drastisch) verkleinern
- Identitäten bewahren
- $\Theta(kn^2)$ für Fehlerw'keit 2^{-k}

Randomisiert

- $\mathbf{A} \cdot (\mathbf{B}\vec{r}) = \mathbf{C}\vec{r}\ \forall r \in \mathbb{F}_2^n$
- Wenn $\mathbf{D} \equiv \mathbf{AB} - \mathbf{C} \neq \mathbf{0}$, aber $\mathbf{D}\vec{r} = 0$:

$$\sum_{j-1}^{n} d_{1j} r_j = 0 \Rightarrow r_1 = -\frac{\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}$$

- Prinzip der aufgeschobenen Entscheidungen

Definition: Minimum Cut

- Gegeben: Ungerichteter, ungewichteter Graph mit n Knoten
- Gesucht: Aufspaltung des Graphen in zwei Teile, so dass möglichst wenig Kanten durchtrennt werden müssen.

Zutaten

- Graph: Menge aus Knoten und Kanten
- Cut: Zwei Graphen, die zusammengenommen den Ursprungsgraph ergeben
- Cut Set: Menge aller durchtrennten Kanten
- Exponentiell viele Cuts bezogen auf Anzahl der Knoten

Definition: Minimum Cut

- Gegeben: Ungerichteter, ungewichteter Graph mit n Knoten
- Gesucht: Aufspaltung des Graphen in zwei Teile, so dass möglichst wenig Kanten durchtrennt werden müssen.

Zutaten

- $G = (V, E)$
- Cut: Zwei Graphen, die zusammengenommen den Ursprungsgraph ergeben
- Cut Set: Menge aller durchtrennten Kanten
- Exponentiell viele Cuts bezogen auf Anzahl der Knoten

Definition: Minimum Cut

- Gegeben: Ungerichteter, ungewichteter Graph mit n Knoten
- Gesucht: Aufspaltung des Graphen in zwei Teile, so dass möglichst wenig Kanten durchtrennt werden müssen.

Zutaten

- $G = (V, E)$
- $V = S \cup T, S \cap T = \emptyset$
- Cut Set: Menge aller durchtrennten Kanten
- Exponentiell viele Cuts bezogen auf Anzahl der Knoten

Definition: Minimum Cut

- Gegeben: Ungerichteter, ungewichteter Graph mit n Knoten
- Gesucht: Aufspaltung des Graphen in zwei Teile, so dass möglichst wenig Kanten durchtrennt werden müssen.

Zutaten

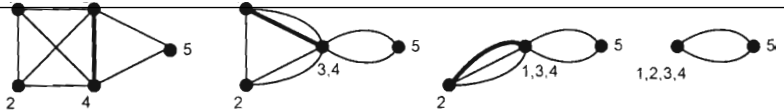
- $G = (V, E)$
- $V = S \cup T, S \cap T = \emptyset$
- Cut Set $C \equiv \{(u, v) \in E : u \in S, v \in T\}, w(S, T) \equiv |C|$
- Exponentiell viele Cuts bezogen auf Anzahl der Knoten

Definition: Minimum Cut

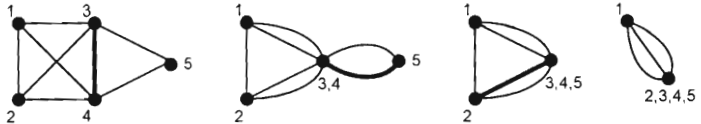
- Gegeben: Ungerichteter, ungewichteter Graph mit n Knoten
- Gesucht: Aufspaltung des Graphen in zwei Teile, so dass möglichst wenig Kanten durchtrennt werden müssen.

Zutaten

- $G = (V, E)$
- $V = S \cup T, S \cap T = \varnothing$
- Cut Set $C \equiv \{(u, v) \in E : u \in S, v \in T\}, w(S, T) \equiv |C|$
- $|C| = \frac{1}{2}(2^{|V|} - 2)$



(a) A successful run of min-cut.



(b) An unsuccessful run of min-cut.

Randomisierter Min-Cut-Algorithmus (Karger)

- Wähle zufällige Kante mit Knoten $\{u, v\}$
- Kontrahiere $\{u, v\}$ zu neuem Knoten $\langle u.v \rangle$
- Entferne alle Kanten zwischen u und v (Achtung: Entstehender Graph G' ist ein *Multigraph*)

Bildquelle: Mitzenmacher und Upfal, *Probability and Computing*

Theorem: Karger Min-Cut-Algorithmus

Der Karger-Min-Cut-Algorithmus findet ein minimales Cut-Set eines ungerichteten (Multi-)Graphen mit n Knoten mit

$$\mathbb{P}(\text{»Min-Cut-Set gefunden«}) \geq \frac{2}{n(n-1)}.$$

Beweis-Struktur

- Cut-Set C teilt $V(G)$ in disjunkte Mengen S und $T = V - S$
- Werden in $n-2$ Schritten nur Kanten in S oder T kontrahiert, ist das (bzw. ein) Cut-Set C gefunden.
- Beobachtung: $|E(C)|/|E(G)| \approx 0$.
- W'keit, dass Kante in C kontrahiert wird, ist gering.

Beweis des Karger-Theorems: Siehe Tafel

Theorem: Karger Min-Cut-Algorithmus

Der Karger-Min-Cut-Algorithmus findet ein minimales Cut-Set eines ungerichteten (Multi-)Graphen mit n Knoten mit

$$\mathbb{P}(\text{»Min-Cut-Set gefunden«}) \geq \frac{2}{n(n-1)}.$$

Lemma: Multigraph-Eigenschaften

In einem Multigraph mit n Knoten und Min-Cut-Größe k gilt $\deg v \geq k \; \forall v \in V$. Außerdem gilt $|E| \geq k\frac{n}{2}$.

Lemma: Cuts in Multigraphen

Die Min-Cut-Größe von $G/(x, y)$ ist mindestens so groß wie die Min-Cut-Größe von G.

Beweis des Karger-Theorems: Siehe Tafel

Monte Carlo

- Laufzeit vorab bestimmt
- Fehlerhaftes Resultat mit W'keit > 0.
- Polynomidentitäten, Verifikation Matrix-Multiplikation
- Ein- und zweiseitige Fehler:
  - Einseitig: $\mathbb{P}$(»Fehler«) = 0 für Ja- *oder* Nein-Antwort.
  - Beidseitig: $\mathbb{P}$(»Fehler«) ≠ 0 für beide Antworten.

Las Vegas

- Niemals fehlerhaftes Resultat
- Laufzeit variiert

Konversion

- Algorithmus laufen lassen
- Korrektheit des Ergebnisses überprüfen
- Wenn inkorrekt, dann wiederholter Versuch

Methoden-Arsenal

- Sampling (Polynomgleichheit)
- Fingerprinting (Verifikation Matrixmultiplikation)
- Aufgeschobene Entscheidungen (dito)
- Verbesserung Fehlerw'keit (Sampling mit und ohne Zurücklegen)
- Fehlerfreiheit versus Laufzeit (Transformation MC↔LV)

Die probabilistische Methode

Übersicht

- Abzähl- und Erwartungs(wert)methoden
- Anwendung: Large Cuts
- Derandomisierung über bedingte Erwartungswerte
- Lovász Local Lemma
- Effizienter Algorithmus für k-Sat

Die probabilistische Methode

Grundprinzip

- Existenzbeweis für Objekte mit bestimmten Eigenschaften
- Zufällige Wahl aus Ereignisraum für Objekt besitzt W'keit $> 0 \Rightarrow$ Objekt existiert
- Beispiel: $\mathbb{P}(\text{»Gewinn in Los-Spiel«}) > 0 \Rightarrow \exists$ Gewinn-Los

Anwendungen

- Einfaches Prinzip, Anwendung oft raffiniert
- Abzähl- und Erwartungs(wert)argumente
- Existenzbeweis kann (oft) in effizienten probabilistischen Algorithmus umgewandelt werden

Vorgehensweise

- Konstruiere Ereignisraum mit geeigneten Objekten
- Zeigen, dass gewünschtes Objekt mit W'keit > 0 auftritt.
- Achtung: W'keit muss *strikt* größer Null sein!

Beispiel

- Färben der Kanten eines vollständigen Graphen mit zwei Farben
- Keine großen monochromatischen Cliquen (vollständige Teilgraphen G_k mit k Knoten) sollen vorkommen

Theorem: Graphen 2-färben

Sei G ungerichteter, vollständiger Graph mit n Knoten. Es ist möglich, die Kanten mit zwei Farben so einzufärben, dass keine monochromatischen Cliquen der Größe k entstehen, wenn

$$\binom{n}{2} 2^{-\binom{k}{2}+1} < 1$$

gilt.

Beweis: Siehe Tafel.

Abzählmethoden III: Konstruktion Algorithmus

Bedingungen

- Auswahl (*Sampling*) einer Färbung aus dem Ereignisraum effizient möglich
- Erfolgsw'keit $p \Rightarrow$ Erwartungswert für Versuchsanzahl, bis gültiges Sample gefunden, ist $\frac{1}{p}$ (geometrische Verteilung!).
- $\frac{1}{p}$ muss polynomial in Eingabelänge sein, um Algorithmus mit erwarteter polynomialer Laufzeit zu erhalten!

Las Vegas

- Effizienter Verifikationsalgorithmus erforderlich
- Graphen färben: Alle $\binom{n}{k}$ Cliquen testen (problematisch, wenn k mit n skaliert)
- So lange testen, bis passendes Sample gefunden

Definition: Zufallsvariable

- Zufallsvariable $X : \Omega \to \mathbb{R}$
- $X = a$ repräsentiert Menge $\{s \in \Omega | X(s) = a\}$
- $\mathbb{P}(X = a) = \sum_{s \in \Omega : X(s) = a} \mathbb{P}(s)$

Erwartungswert

$$\mathbb{E}(X) = \sum_{x \in \mathrm{im}(X)} x\mathbb{P}(X = x)$$

Definition: Unabhängige Zufallsvariablen

$$\mathbb{P}((X = x) \cap (Y = y)) = \mathbb{P}(X = x) \times \mathbb{P}(Y = y)$$

Formelsammlung

X, Y bzw. $X_1, X_2, \ldots, X_n$: Zufallsvariablen

- $\mathbb{E}(X) = \sum_{i=1}^{\infty} \mathbb{P}(X \geq i)$ (für Zufallsvariable mit nicht-negativen Ganzzahlwerten)
- $\mathbb{E}(X) = \sum_{y \in \mathrm{im}(Y)} \mathbb{P}(Y = y)\mathbb{E}(X|Y = y)$
- Wechselseitige Unabhängigkeit (analog zu Ereignissen) genau dann, wenn

$$\mathbb{P}\left(\bigcap_{i \in I} X_i = x_i\right) = \prod_{i \in I} \mathbb{P}(X_i = x_i) \text{ für jedes } I \subseteq [1, n] \text{ und jedes } x_i$$

- k-fache Unabhängigkeit: $|I| \leq k$
- Paarweise Unabhängigkeit: 2-fache Unabhängigkeit

Beweise: Siehe beispielsweise *Mitzenmacher und Upfahl, Kapitel 2.1*

Prinzip

- Erwartungswert μ einer Zufallsvariable sei bekannt
- Mindestens ein Objekt nicht größer als μ
- Mindestens ein Objekt nicht kleiner als μ

Formalisierung

Sei X eine Zufallsvariable mit Erwartungswert $\mathbb{E}(X) = \mu \in \mathbb{R}$. Dann gilt

$$\mathbb{P}(X \geq \mu) > 0 \wedge \mathbb{P}(X \leq \mu) > 0.$$

Beweis: Siehe Tafel

k-Cut

- Gegeben Graph G und $k \in \mathbb{N}$, existiert Cut $\geq k$?
- **NP**-vollständiges Entscheidungsproblem.
  - Zertifikat: Cut selbst
- **APX**-hart (nicht beliebig gut approximierbar)

Lemma: Linearität von $\mathbb{E}(\cdot)$

Für alle diskreten Zufallsvariablen $X_1, X_2, \ldots, X_n$ mit endlichem Erwartungswert gilt

$$\mathbb{E}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathbb{E}(X_i)$$

Beweis: Siehe Übungsaufgabe

Large Cut

Sei G ungerichteter Graph mit $V(G) = n$ und $E(G) = m$. Dann existieren $A, B \subset V(G)$, die durch mindestens $\frac{m}{2}$ Kanten verbunden sind.

Beweis: Siehe Tafel

Umwandlung des Existenz-Arguments in einen effizienten Algorithmus

Prinzip

- Berechnung einer unteren Schranke p für Erfolgsw'keit
- Auswahl zufälliger Partitionen A, B: n zufällige Bits
- Sampling durchführen, bis erfolgreiches Ergebnis zu erwarten.
- Effizient testen, ob Ziel erreicht (Kanten des Cuts zählen). Wenn nicht, weitersampeln.

Zutaten

- p(»Large Cut gefunden«)

$$\mathbb{P}\left(C(A,B) \geq \frac{m}{2}\right) \geq \frac{1}{m/2+1}$$

Beweis: Siehe Tafel.

Large Cut: Randomisierter Algorithmus

Umwandlung des Existenz-Arguments in einen effizienten Algorithmus

Prinzip

- Berechnung einer unteren Schranke p für Erfolgsw'keit
- Auswahl zufälliger Partitionen A, B: n zufällige Bits
- Sampling durchführen, bis erfolgreiches Ergebnis zu erwarten.
- Effizient testen, ob Ziel erreicht (Kanten des Cuts zählen). Wenn nicht, weitersampeln.

Zutaten

- p(»Large Cut gefunden«)

$$\mathbb{P}\left(C(A,B) \geq \frac{m}{2}\right) \geq \frac{1}{m/2+1}$$

Beweis: Siehe Tafel.

Allgemeine Konstruktion

- Randomisierte Elemente durch strukturierte Auswahl bzw. Konstruktion ersetzen
- Effizienz darf nicht verlorengehen
- Hier: Methode der bedingten Erwartungswerte (*conditional expectations*)

Bedingte Erwartungswerte

Y, Z: Zufallsvariablen, E: Ereignis

$$\mathbb{E}(Y|Z=z) = \sum_y y\mathbb{P}(Y=y|Z=z)$$

$$\mathbb{E}(Y|E) = \sum_y y\mathbb{P}(Y=y|E)$$

Nützliche Identität

$$\mathbb{E}(X) = \sum_y \mathbb{P}(Y=y)\mathbb{E}(X|Y=y)$$

Allgemeine Konstruktion

- Randomisierte Elemente durch strukturierte Auswahl bzw. Konstruktion ersetzen
- Effizienz darf nicht verlorengehen
- Hier: Methode der bedingten Erwartungswerte (*conditional expectations*)

Bedingte Erwartungswerte

Y, Z: Zufallsvariablen, E: Ereignis

$$\mathbb{E}(Y|Z=z) = \sum_y y\mathbb{P}(Y=y|Z=z)$$

$$\mathbb{E}(Y|E) = \sum_y y\mathbb{P}(Y=y|E)$$

Nützliche Identität

$$\mathbb{E}(X) = \sum_y \mathbb{P}(Y=y)\mathbb{E}(X|Y=y)$$

Derandomisierung von Large Cut I

Strategie

- Zufalls-Plazierung Knoten: Großer Cut mit hoher W'keit
- Alternative: Anfangsknoten deterministisch setzen, weitere Knoten zufällig
- Gesucht: Verfahren mit steigendem Erwartungswert für Cut-Größe

Formalisierung

- Partitionierung $V(G) = A \cup B$, $\mathbb{E}(C(A,B) \geq \frac{m}{2})$
- $x_i \in \{A, B\}$: Position i-ter Knoten
- $\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k)$: Erwartungswert Cut gegeben Position der ersten k Knoten
- Gesucht: Strategie zur Plazierung des $k+1$-ten Knotens, so dass

$$\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k) \leq \mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_{k+1})$$

- $\mathbb{E}(C(A,B)) \leq \mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_n)$

Beweis: $\mathbb{E}(C(A,B)) \leq \mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_n)$

Induktionsanfang

$$\mathbb{E}(C(A,B)|x_1) = \mathbb{E}(C(A,B))$$

Gilt wegen Symmetrie zwischen A und B.

Induktionsschritt

Sei $Y_{k+1} \in \{A, B\}$ eine Zufallsvariable, die Plazierung von Knoten $k+1$ angibt (W'keit jeweils $\frac{1}{2}$).

$$\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k) = \frac{1}{2} \underbrace{\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k, Y_{k+1} = A)}_{①}$$

$$+ \frac{1}{2} \underbrace{\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k, Y_{k+1} = B)}_{②}$$

$$\max(①, ②) \geq \mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k)$$

Derandomisierung von Large Cut III

Analyse

- Wert von $\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k, Y_{k+1} = A)$?
- $k+1$ plazierte Knoten: Cut Size leicht berechenbar
- Nicht plazierte Knoten: Tragen mit W'keit $\frac{1}{2}$ zum Cut bei.
- Dito für $Y_{k+1} = B \Rightarrow$ Berechne $\mathbb{E}(\cdot)$ für beide Fälle, verwende günstigeren.
- *Effektiv ausreichend*: Plazierung von Knoten $k+1$ in die Menge mit *weniger* Nachbarknoten

Derandomisierung von Large Cut III

Analyse

- Wert von $\mathbb{E}(C(A,B)|x_1, x_2, \ldots, x_k, Y_{k+1} = A)$?
- $k+1$ plazierte Knoten: Cut Size leicht berechenbar
- Nicht plazierte Knoten: Tragen mit W'keit $\frac{1}{2}$ zum Cut bei.
- Dito für $Y_{k+1} = B \Rightarrow$ Berechne $\mathbb{E}(\cdot)$ für beide Fälle, verwende günstigeren.
- *Effektiv ausreichend*: Plazierung von Knoten $k+1$ in die Menge mit *weniger* Nachbarknoten

Derandomisierung von Large Cut IV

Deterministischer Algorithmus

1: **procedure** Large Cut(V, E) ▷ Deterministisch!
2: Allocate $\vec{x} \in \mathbb{F}_2^{|V|}$; $A \leftarrow 0, B \leftarrow 1$
3: **for** $i \leftarrow 1, |V|$ **do** ▷ Reihenfolge der Knoten beliebig, aber fest
4: **if** $i = 1$ **then**
5: $x_1 \leftarrow A$
6: **else**
7: **if** NumNeigh$(V_i, A) \geq$ NumNeigh(V_i, B) **then**
8: $x_i \leftarrow B$
9: **else**
10: $x_i \leftarrow A$
11: **end if**
12: **end if**

Grundidee

- Ereignisraum mit »schlechten« Ereignissen
- Gesucht: Beweis, dass ein Element existiert, das nicht in den schlechten Ereignissen enthalten ist
- Einfach, wenn $\{E_i\}$ wechselseitig unabhängig
- Reale Welt: Ereignisse oft nicht komplett wechselseitig unabhängig, aber mit »milden« Abhängigkeiten

Formalisierung

- »Schlechte« Ereignisse: $E_1, E_2, \ldots, E_n$
- Wechselseitig unabhängig:

$$\mathbb{P}\left(\bigcap_{i\in I} E_i\right) = \prod_{i\in E} \mathbb{P}(E_i)$$

$\forall I \subseteq [1,n]$

- W'keit »gutes« Ereignis:

$$\mathbb{P}\left(\bigcap_{i=1}^{n} \bar{E}_i\right) = \prod_{i=1}^{n} \mathbb{P}(\bar{E}_i) > 0$$

(ausser wenn $\exists i : E_i = 1$)

Das Lovász Local Lemma (LLL) I

Grundidee

- Ereignisraum mit »schlechten« Ereignissen
- Gesucht: Beweis, dass ein Element existiert, das nicht in den schlechten Ereignissen enthalten ist
- Einfach, wenn $\{E_i\}$ wechselseitig unabhängig
- Reale Welt: Ereignisse oft nicht komplett wechselseitig unabhängig, aber mit »milden« Abhängigkeiten

Formalisierung

- »Schlechte« Ereignisse: $E_1, E_2, \ldots, E_n$
- Wechselseitig unabhängig:

$$\mathbb{P}\left(\bigcap_{i\in I} E_i\right) = \prod_{i\in E} \mathbb{P}(E_i)$$

  $\forall I \subseteq [1,n]$
- W'keit »gutes« Ereignis:

$$\mathbb{P}\left(\bigcap_{i=1}^{n} \bar{E}_i\right) = \prod_{i=1}^{n} \mathbb{P}(\bar{E}_i) > 0$$

  (ausser wenn $\exists i : E_i = 1$)

Wechselseitige Unabhängigkeit

- E ist wechselseitig unabhängig von $\{E_i\}$, wenn

$$\mathbb{P}\left(E \middle| \bigcap_{j \in I} E_j\right) = \mathbb{P}(E)$$

für jedes $I \subseteq [1, n]$ gilt.

- E ist abhängig von $\{E_i\}$, wenn die Identität *nicht* gilt.

Stilfrage

Wie kann man die Abhängigkeiten und Unabhängigkeiten zwischen verschiedenen Ereignissen am besten formalisieren?

Definition: Abhängigkeitsgraph

Ein Abhängigkeitsgraph für Ereignisse $F_1, F_2, \ldots, F_n$ ist ein gerichteter Graph $G = (V, E)$, so dass

- $V = \{1, \ldots, n\}$ (entsprechend den Ereignissen F_i)
- F_i (für $1 \leq i \leq n$) wechselseitig unabhängig von den Ereignissen $\{F_j | (i, j) \notin E\}$ ist.

Das Lovász Local Lemma (LLL) IV

Lemma: Lovász Local, symmetrische Variante

Wenn folgende Eigenschaften gelten:

- Die Wahrscheinlichkeit der »schlechten« Ereignisse ist begrenzt (gleiche Grenze für alle Ereignisse)
- Der Grad des Abhängigkeitsgraphen ist begrenzt (d.h. ein Ereignis hängt maximal von einer festen Anzahl anderer Ereignisse ab)
- Ein bestimmer Trade-Off zwischen den beiden Bedingungen erfüllt ist

dann tritt ein »gutes« Ereignis sicher auf.

Das Lovász Local Lemma (LLL) IV

Lemma: Lovász Local, symmetrische Variante

Wenn folgende Eigenschaften gelten:

- Für alle i gilt $\mathbb{P}(E_i) \leq p$
- Der Grad des Abhängigkeitsgraphen ist begrenzt (d.h. ein Ereignis hängt maximal von einer festen Anzahl anderer Ereignisse ab)
- Ein bestimmer Trade-Off zwischen den beiden Bedingungen erfüllt ist

dann tritt ein »gutes« Ereignis sicher auf.

Lemma: Lovász Local, symmetrische Variante

Wenn folgende Eigenschaften gelten:

- Für alle i gilt $\mathbb{P}(E_i) \leq p$
- $\deg(G) \leq d \in \mathbb{N}$
- Ein bestimmer Trade-Off zwischen den beiden Bedingungen erfüllt ist

dann tritt ein »gutes« Ereignis sicher auf.

Lemma: Lovász Local, symmetrische Variante

Wenn folgende Eigenschaften gelten:

- Für alle i gilt $\mathbb{P}(E_i) \leq p$
- $\deg(G) \leq d \in \mathbb{N}$
- $4dp \leq 1$ (implizit folgt $p \leq \frac{1}{4}$)

dann tritt ein »gutes« Ereignis sicher auf.

Lemma: Lovász Local, symmetrische Variante

Wenn folgende Eigenschaften gelten:

- Für alle i gilt $\mathbb{P}(E_i) \leq p$
- $\deg(G) \leq d \in \mathbb{N}$
- $4dp \leq 1$ (implizit folgt $p \leq \frac{1}{4}$)

$$\text{dann gilt } \mathbb{P}\left(\bigcap_{i=1}^{n} \bar{E}_i\right) > 0.$$

Lemma

Es gelten die Voraussetzungen des LLL. Sei $S \subset [1,n]$, $|S| \leq n-1$. Dann gilt $\forall k \notin S$:

$$\mathbb{P}\left(E_k | \bigcap_{j \in S} \bar{E}_j\right) \leq 2p.$$

Noch zu beweisen (schwieriger Teil des LLL)

Lemma: Iterative Anwendung der bedingten W'keit

$$\mathbb{P}\left(\bigcap_{i=1}^{n} E_i\right) = \prod_{i=1}^{n} \mathbb{P}\left(E_i | \bigcap_{j=1}^{i-1} E_j\right)$$

Siehe Beweis des Polynomgleichheits-Algorithmus.

Voraussetzungen

$$\mathbb{P}\left(E_k \middle| \bigcap_{j\in S} \bar{E}_j\right) \leq 2p \qquad \mathbb{P}\left(\bigcap_{i=1}^{n} E_i\right) = \prod_{i=1}^{n} \mathbb{P}\left(E_i \middle| \bigcap_{j=1}^{i-1} E_j\right)$$

Beweis des LLL

$$\begin{aligned}\mathbb{P}\left(\bigcap_{i=1}^{n} \bar{E}_i\right) &= \prod_{i=1}^{n} \mathbb{P}\left(\bar{E}_i \middle| \bigcap_{j=1}^{i-1} \bar{E}_j\right) \\ &= \prod_{i=1}^{n} \left[1 - \mathbb{P}\left(E_i \middle| \bigcap_{j=1}^{i-1} \bar{E}_j\right)\right] \\ &\geq \prod_{i=1}^{n} (1-2p) > 0.\end{aligned}$$

Lemma zum LLL

$$\mathbb{P}\left(E_k \middle| \bigcap_{j \in S} \bar{E}_j\right) \leq 2p.$$

Beweisstruktur: Lemma zum LLL

Induktion nach $s = 0, \ldots, n-1$.

- Induktionsanfang: $s = 0$, einfacher Fall.
- Induktionsschritt: Zunächst $\mathbb{P}(\cap_{j \in S} \bar{E}_j) > 0$ zu beweisen, damit Lemma wohldefiniert ist. Danach: Aufteilung in $S_1 = \{j \in S | (k, j) \in E\}$ und $S_2 = S - S_1$.
  - Fall 1: $S_2 = S$, einfacher Fall.
  - Fall 2: $|S_2| < s$.

Beweis: Siehe Tafel.

Theorem: k-SAT

Gegeben eine logische Formel mit Klauseln, in denen jeweils genau k Variablen auftreten. Klauseln, in denen eine Variable und deren Negation enthalten ist, sind ausgeschlossen (da trivial erfüllt).
Wenn keine Variable dieser k-SAT-Formel in mehr als $T = \frac{2^k}{4k}$ Klauseln auftritt, gibt es eine Belegung der Variablen, die die Formel erfüllen.

Interpretation von k-SAT

Zwei unterschiedliche Konventionen in der Literatur:

- *Genau k* Variablen pro Klausel
- *Bis zu k* Variablen pro Klausel

Beide Varianten führen zu identischen Komplexitätsklassen. k-SAT ist für $k \geq 3$ **NP**-vollständig.

Beweis

Betrachte eine uniform zufällige Zuweisung von Werten an die Literale.

- Sei E_i das Ereignis »i-te Klausel ist nicht erfüllt«
- Jede Klausel hat genau k Variablen, also $\mathbb{P}(E_i) = 2^{-k}$
- E_i ist wechselseitig unabhängig zu E_j, $j \neq i$, wenn Klausel i und j keine gemeinsamen Variablen besitzen.
- Jede Variable in i kann in maximal T Klauseln vorkommen, ergo $d \leq kT \leq 2^{k-2}$
- Also ergibt sich für das LLL: $4dp \leq 4 \times 2^{k-2} 2^{-k} \leq 1$.
- $\mathbb{P}\left(\bigcap_i \bar{E}_i\right) > 0$, d.h. es gibt eine erfüllende Belegung.

Viel Aufwand für ein Lemma

- Lovász und Erdős 1973: $4pd \leq 1$
- Lovász 1977: $ep(d+1) \leq 1$
- Shearer 1985: $epd \leq 1$, Schranke optimal
- Józef Beck 1991: Konstruktive Variante (unter restriktiven Bedingungen)
- Moser 2009, Moser & Gábor Tardos 2009: Konstruktive Variante unter allgemeinen Bedingungen

k-SAT: Las Vegas-Algorithmus

Sei $\mathcal{F}$ eine k-SAT-Formel, wobei k eine gerade, kleine Konstante ist. Jede Variable darf maximal $2^{\alpha k}$ mal auf, mit $\alpha > 0$ klein. Dann findet der im folgenden beschriebene Algorithmus in erwarteter linearer Zeit (in der Anzahl der Klauseln m) mit hoher Wahrscheinlichkeit eine erfüllende Belegung.

Struktur: Zwei Phasen

1. Teilmenge aller Variablen wird zufällig belegt.
2. Die verbleibenden Variablen werden deterministisch bestimmt.
   - LLL muss garantieren, dass Belegung existiert
   - Effizient ☞ Abhängigkeitsgraph darf keine großen zusammenhängenden Teilgraphen besitzen

Typische Struktur für LLL-basierte Algorithmen

Siehe *Mitzenmacher und Upfahl*, Kapitel 8.1

Nomenklatur

- l Variablen: $x_1, x_2, \ldots, x_l$
- m Klauseln: $C_1, C_2, \ldots, C_m$
- *Überlebende* und *gefährliche* Klauseln: Siehe Tafel
- Nach Teil 1 unbelegte Variablen: *Aufgeschobene Variablen*

Lemma 1

Es gibt eine Zuweisung von Werten an die aufgeschobenen Variablen, die eine erfüllende Belegung von $\mathcal{F}$ liefern

Lemma 2

Alle zusammenhängenden Teilgraphen von H' besitzen mit Wahrscheinlichkeit $1 - o(1)$ eine Größe von $\mathcal{O}(\log m)$.

4-Baum

Sei G ein zusammenhängender Graph. Eine Teilmenge $T \subseteq V(G)$ induziert einen sogenannten *4-Baum*, wenn

1. Die Entfernung zwischen allen Knoten in T ist mindestens 4
2. Wenn ein neuer Graph gebildet wird, in dem alle Klauseln adjazent sind, die in G genau Entfernung 4 haben, ist T zusammenhängend.

Inhalt

Definition: Probabilistische Turing-Maschine (PTM)

Eine *probabilistische Turing-Maschine* (PTM) ist eine Turing-Maschine mit zwei Übergangsfunktionen δ_0, δ_1. In jedem Rechenschritt wird eine der beiden Funktionen mit W'keit $\frac{1}{2}$ ausgewählt, unabhängig von allen Auswahlen in den vorhergehenden Rechenschritten.

$\delta : Q \backslash q_{\text{accept}} \times \Gamma \to (Q \times \Gamma \times \{\text{L}, 0, \text{R}\}), (Q \times \Gamma \times \{\text{L}, 0, \text{R}\})$

Randomisierung versus Nichtdeterminismus

Eine probabilistische TM wählt *einen* zufälligen Pfad durch den Berechnungsbaum. Dies ist *nicht* äquivalent zum Verhalten einer nichtdeterministischen TM.

- NTM: Ein akzeptierender Pfad genügt zur sicheren »Gesamt«akzeptanz
- PTM: W'keit: Verhältnis akzeptierende zu gesamten Pfaden

Definition: Komplexitätsklasse BPTIME

Sei $L \subseteq \{0,1\}^*$ eine Sprache, und $T : \mathbb{N} \to \mathbb{N}$. Eine PTM M entscheidet L in Zeit $T(n)$, wenn gilt, dass $\forall x \in \{0,1\}^*$

- M hält nach $T(|x|)$ Schritten, unabhängig von den Zufallswahlen
- $\mathbb{P}(M(x) = L(x)) \geq \frac{2}{3}$ (gilt für $M(x)$ = »akzeptieren« und $M(x)$ = »ablehnen«)

Definition: Komplexitätsklasse BPP

BPP (*bounded error probabilistic polynomial*) ist definiert durch

$$\mathbf{BPP} \equiv \bigcup_{c \in \mathbb{N}} \mathbf{BPTIME}(n^c)$$

Anmerkungen

- **BPP** ist invariant gegenüber Änderung der Konstante $\frac{2}{3}$, solange diese $> \frac{1}{2}$.
- *Zweiseitiger* Fehler
- **BPP** definiert auf Worst-Case-Basis!
- *Semantische* anstelle *syntaktischer* Definition

Excluded Middle

Eine **BPP**-PTM akzeptiert jede Eingabe $x \in \{0,1\}^*$ mit W'keit $\geq \frac{2}{3}$, oder lehnt mit W'keit $\geq \frac{2}{3}$ ab.

Randomisierte Komplexitätsklassen III

Triviale Beziehungen BPP

$$\mathbf{P} \subseteq \mathbf{BPP}$$

$$\mathbf{BPP} \subseteq \mathbf{EXPTIME}$$

Beweis

- **PTM** führt gleiche Rechnung in jedem randomisierten Zweig durch.
- **DTM** durchläuft $2^{\text{poly}(n)}$ Pfade.

Definition: Komplexitätsklassen RTIME und RP

RTIME$(T(n))$ enthält alle Sprachen L, für die es eine PTM M gibt, die nach $T(n)$ Zeitschritten hält, und für die gilt

- $x \in L : \mathbb{P}(M(x) = 1) \geq \frac{2}{3}$
- $x \notin L : \mathbb{P}(M(x) = 1) = 0$

Es gilt $\mathbf{RP} \equiv \bigcup_{c \in \mathbb{N}} \mathbf{RTIME}(n^c)$

Komplementärklasse

Sei **C** eine Komplexitätsklasse und $L \in \mathbf{C}$ die darin enthaltene Sprache. Dann ist

$$\mathbf{coC} \equiv \{L | \bar{L} \in \mathbf{C}\}$$

die zu **C** komplementäre Komplexitätsklasse.

RP und coRP

RP

	Entscheidung	
Korrekt	Akz.	Abl.
Akzeptieren	$\geq 2/3$	$\leq 1/3$
Ablehnen	0	1

Definition: Komplexitätsklassen RTIME und RP

RTIME$(T(n))$ enthält alle Sprachen L, für die es eine PTM M gibt, die nach $T(n)$ Zeitschritten hält, und für die gilt

- $x \in L : \mathbb{P}(M(x) = 1) \geq \frac{2}{3}$
- $x \notin L : \mathbb{P}(M(x) = 1) = 0$

Es gilt $\mathbf{RP} \equiv \bigcup_{c \in \mathbb{N}} \mathbf{RTIME}(n^c)$

Komplementärklasse

Sei **C** eine Komplexitätsklasse und $L \in \mathbf{C}$ die darin enthaltene Sprache. Dann ist

$$\mathbf{coC} \equiv \{L | \bar{L} \in \mathbf{C}\}$$

die zu **C** komplementäre Komplexitätsklasse.

RP und coRP

coRP

	Entscheidung	
Korrekt	Akz.	Abl.
Akzeptieren	1	0
Ablehnen	$\leq 1/3$	$\geq 2/3$

Definition: Komplexitätsklassen ZTIME und ZPP

ZTIME enthält alle Sprachen L, für die es eine PTM M gibt, die in erwarteter Zeit $\mathcal{O}(T(n))$ läuft und stets ein korrektes Ergebnis ausgibt.
Es gilt $\mathbf{ZPP} \equiv \bigcup_{c \in \mathbb{N}} \mathbf{ZTIME}(n^c)$

Theorem: Zusammenhang RP, coRP und ZPP

$$\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$$

Beweis: Siehe Tafel

Zutat: Markov-Ungleichung

Für eine Zufallsvariable X mit nicht-negativen Werten gilt

$$\forall a > 0 : \mathbb{P}(X \geq a) \leq \frac{\mathbb{E}(X)}{a}$$

Inklusionen

- $\mathbf{RP} \subseteq \mathbf{NP}$
- $\mathbf{P} \subseteq \mathbf{ZPP}$ ✓
- $\mathbf{P} \subseteq \mathbf{BPP}$ ✓
- $\mathbf{BPP} = \mathbf{coBPP}$
- $\mathbf{RP} \subseteq \mathbf{BPP}$
- $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$ ✓
- $\mathbf{BPP} \subseteq \mathbf{EXPTIME}$ ✓
- Ansonsten: $\mathbf{BPP} \overset{?}{\subseteq} \mathbf{NP}$

Beweise: Siehe Tafel

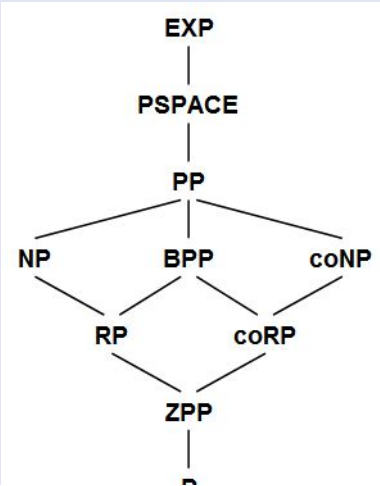
Bildquelle: www.scottaaronson.com

Randomisierte Komplexitätsklassen VI

Inklusionen

- RP $\subseteq$ NP
- P $\subseteq$ ZPP ✓
- P $\subseteq$ BPP ✓
- BPP = coBPP
- RP $\subseteq$ BPP
- ZPP = RP $\cap$ coRP ✓
- BPP $\subseteq$ EXPTIME ✓
- Ansonsten: BPP $\overset{?}{\subseteq}$ NP

Illustration



Offene Fragen

- $\mathbf{BPP} \overset{?}{\subseteq} \mathbf{NP}$
- $\mathbf{P} \overset{?}{=} \mathbf{NP} \cap \mathbf{coNP}$
- $\mathbf{P} \overset{?}{=} \mathbf{BPP}$, $\mathbf{NP} \overset{?}{=} \mathbf{BPP}$

Pseudozufall und Derandomisierung

Zufall konstruieren

- **BPP**: Polynomiale Laufzeit *mit echtem Zufall*
- Annahme: Pseudo-Zufallsstrom kann nicht algorithmisch in polynomialer Zeit von echtem Zufall unterschieden werden
- Deterministische Simulation möglich: Generator und Algorithmus hintereinander schalten → weiterhin polynomiale Laufzeit
- Resultat: **BPP** = **P**

Aufgabe

Konstruktion eines PRNG, dessen Ausgabe nicht in poly-Zeit von echtem Zufall unterschieden werden kann.
Problem: Muss für alle Unterscheidungsalgorithmen (einschließlich bislang unbekannter!) funktionieren.

Oxymoron (Contradictio in adiecto)

- Problematische Terminologie: *Deterministischer Zufalls*(zahlen)generator
- Besser: Pseudozufalls{zahlen,bit,}generator

Randomisierter Algorithmus

- Gegeben: Randomisierter Algorithmus A'
- Zutaten: Deterministischer Alg. A, zufällige Bits r, Eingabe x.
- Vereinfachung: $A(x, r) = A(r)$ (wird später aufgehoben)

PRNG

- $g : \{0,1\}^k \to \{0,1\}^l$
- Seed (Samen) s mit $|s| = k$ gestreckt auf $|g(s)| = l$ zufällige Bits, wobei $l > k$

Wann ist Pseudozufall ausreichend?

- Erfolgs/akzeptanzw'keit mit l echten Zufallsbits:

$$\mathbb{P}(\text{»Ja«}) = \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})$$

- Erfolgs/akzeptanzw'keit mit k echten Zufallsbits, gestreckt auf l Bits:

$$\mathbb{P}(\text{»Ja«}) = \mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«})$$

Definition: Ununterscheidbarkeit Zufall/Pseudozufall

Sei g eine Funktion $\{0,1\}^k \to \{0,1\}^l, k < l, k = k(l)$. Sei A ein deterministischer Algorithmus mit l Zufallsbits als Eingabe. g »ϵ-täuscht« A, wenn

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})\right| \leq \epsilon$$

Definition: ϵ-starker Pseudozufallsgenerator

g ist ein ϵ-starker Pseudozufallsgenerator, wenn er alle Algorithmen $A \in \mathbf{P}$ ϵ-täuscht.

Anmerkungen zur Unterscheidung Zufall/Pseudozufall

- $\epsilon = o(l^{-c}) \forall c \in \mathbb{R}^+ \Rightarrow$ super-polynomiale Anzahl an Wiederholungen erforderlich, um Unterscheidung durchzuführen, d.h. unmöglich für $A \in \mathbf{P}$. Definiert *starken* PRNG.
- Keine explizite Quantifizierung der zugrundeliegenden Verteilung (Entropie etc.) erforderlich
- Wesentlich mächtiger als jeder spezifische konkrete (effiziente) Test
- Ein einziger Algorithmus A, der zwischen r und $g(s)$ unterscheidet, reicht aus, um g zu »entlarven«

Derandomisierung von BPP I

Wenn ein Algorithmus aus **BPP** l zufällige Bits benötigt, existiert eine deterministische Version, die in Zeit $2^l \text{poly}(n)$ abläuft.

Derandomisierung von BPP II

Wenn ein Algorithmus aus **BPP** von einem Pseudozufallsgenerator mit Seed-Länge k getäuscht wird, existiert eine deterministische Version, die in Zeit $2^k \text{poly}(n)$ abläuft.

Wanted...

Pseudozufallsgenerator g,

- der Seeds mit logarithmischer Länge $\log(l)$ in der Anzahl benötigter Zufallsbits l verwendet
- dessen Ausgabe nicht in poly-Zeit von echtem Zufall unterscheidbar ist.

Derandomisierung von BPP I

Wenn ein Algorithmus aus **BPP** l zufällige Bits benötigt, existiert eine deterministische Version, die in Zeit $2^l \text{poly}(n)$ abläuft.

Derandomisierung von BPP II

Wenn ein Algorithmus aus **BPP** von einem Pseudozufallsgenerator mit Seed-Länge k getäuscht wird, existiert eine deterministische Version, die in Zeit $2^k \text{poly}(n)$ abläuft.

Wanted...

Pseudozufallsgenerator g,

- der Seeds mit logarithmischer Länge $\log(l)$ in der Anzahl benötigter Zufallsbits l verwendet
- dessen Ausgabe nicht in poly-Zeit von echtem Zufall unterscheidbar ist.

Zutaten: Blum-Micali-Generator

- Einweg-Funktion f: Berechnung von $y = f(x)$ einfach; Berechnung von $x = f^{-1}(y)$ effizient unmöglich.
- Einweg-Permutation: Bijektive Einwegfunktion
- b: poly-Zeit-Funktion, die Bit $b(x)$ aus Bitstring x berechnet
- Hard-Core-Bit bzw. -Prädikat: Berechnung von $b(y)$ genau so schwierig wie Berechnung von x aus $y = f(x)$.

Beispiele für (vermutete!) Einwegfunktionen

- Exponentiation und Logarithmus über endlichen Körpern (diskreter Logarithmus)
- Multiplikation und Faktorisierung
- Kryptographische Hashfunktionen (bsp. SHA256)

Definition: Blum-Micali-Yao-Generator

Sei s ein zufälliger Bit-String mit $|s| = k$, und f eine Einweg-Permutation mit Hard-Core-Bit $b(x)$. Dann ist ein starker Pseudozufallsgenerator $g_l : \{0,1\}^k \to \{0,1\}^l$ gegeben durch

$$g_l(s) = \langle b(s), b(f(s)), b(f(f(s))), b(f^3(s)), \ldots, b(f^{l-1}(s)) \rangle,$$

wenn $l = \text{poly}(k)$ gilt.

- Für jede Konstante a kann aus k Seed-Bits ein pseudozufälliger Bitstring der Länge $l = k^a$ erzeugt werden
- Zufallsstring kann nach und nach bitweise erzeugt werden
- Intention von Blum und Micali: $g : \{0,1\}^{\sqrt{n}} \to \{0,1\}^n$.
- Korrektheit: Noch zu zeigen

Definition: Einwegfunktion und Hard-Core-Bit

Seien $f : \{0,1\}^k \to \{0,1\}^k$ und $b : \{0,1\}^k \to \{0,1\}$ in Zeit poly(k) berechenbar. Dann ist f eine Einwegfunktion mit Hard-Core-Bit b, wenn für alle $A \in \mathbf{BPP}$ und alle Konstanten $c \in \mathbb{R}$ gilt, dass

$$\mathbb{E}(\mathbb{P}(A(f(x)) = b(x))) = \frac{1}{2} + o(k^{-c}).$$

(Der Erwartungswert wird über $x \in \{0,1\}^k$ berechnet)

Interpretation

- Einzelnes Ergebnisbits kann durch randomisierten Algorithmus (oder, schwächer, deterministischen Algorithmus) nur unwesentlich besser als durch Raten (W'keit ½) ermittelt werden.
- Die Komplexität gilt für den *durchschnittlichen* Fall (und nicht nur im Worst Case)

Theorem: Konstruktion PRNG aus Einweg-Permutation

Sei f eine Einweg-Permutation mit Hard-Core-Bit b. Dann ist

$$g(x) = \langle f(x), b(x) \rangle$$

ein starker Pseudozufallsgenerator mit $l = k + 1$.

Beweisstruktur

Beweis durch Widerspruch:

- Annahme: Es gibt einen poly-Zeit Algorithmus A, der $g(x)$ von einem gleichlangen Zufallsstring unterscheiden kann.
- Dann existiert ein Algorithmus, den Wert $b(x)$ aus $f(x)$ mit besserer W'keit als Raten zu berechnen.
- Widerspruch zur Voraussetzung, dass f eine Einwegfunktion bzw. b ein Hard-Core-Bit ist.

Beweis: Siehe Tafel.

Blum-Micali-Yao-Generator

Ein starker Pseudozufallsgenerator ist gegeben durch

$$g_l(s) = \langle b(s), b(f(s)), b(f(f(s))), b(f^3(s)), \ldots, b(f^{l-1}(s)) \rangle,$$

wenn $l = \text{poly}(k)$ gilt, f eine Einwegpermutation und b ein Hard-Core-Bit ist.

Beweis (Struktur)

- Widerspruchsbeweis: $\exists$ Tester A, der zwischen zufälligem String und $g_l(s)$ unterscheidet.
- Konstruktion: Schrittweise Transition zwischen uniformem r und $g_l(s)$. Eines der ausgetauschten Bits muss nach Definition einen Unterschied machen.
- Erwartungswert mittels Teleskopsumme (Differenz zwischen Einzelschritten) berechnen
- Konstruktion eines Algorithmus B, der Hard-Core Bit mit besserer W'keit als bloßem Raten berechnet.
- Details: Siehe *Moore & Mertens*, Kapitel 11.4.3

Blum-Micali-Yao-Generator

$$g_l(s) = \langle b(s), b(f(s)), b(f(f(s))), b(f^3(s)), \ldots, b(f^{l-1}(s)) \rangle$$

Beweis (Struktur)

- Widerspruchsbeweis: $\exists$ Tester A, der zwischen zufälligem String und $g_l(s)$ unterscheidet.
- Konstruktion: Schrittweise Transition zwischen uniformem r und $g_l(s)$. Eines der ausgetauschten Bits muss nach Definition einen Unterschied machen.
- Erwartungswert mittels Teleskopsumme (Differenz zwischen Einzelschritten) berechnen
- Konstruktion eines Algorithmus B, der Hard-Core Bit mit besserer W'keit als bloßem Raten berechnet.
- Details: Siehe *Moore & Mertens*, Kapitel 11.4.3

Blum-Micali und logarithmische Seeds

- Blum-Micali-Generator: Angenommen $k = \log(n)$ möglich
- Einwegfunktion f kann in poly-Zeit $t(k)$ berechnet werden
- Berechnung von f^{-1}: Alle möglichen Eingaben durchprobieren in Zeit $2^k t(k) = 2^{\log(n)} t(k)$
- Widerspruch: Invertierung von f in poly-Zeit möglich $\Rightarrow$ f ist keine Einwegfunktion.
- $\Rightarrow$ Mindestens polynomial lange Seeds erforderlich

Mildern der exponentiellen Zeitkomplexität

- $\forall k \in \mathbb{N} : l = k^a$ pseudo-zufällige Bits erzeugbar
- k Bits werden in $l = k^a$ Bits *expandiert*
- Umkehrung: l zufällige Bits können mit einer Seed-Länge von $k = l^\epsilon$ Bits simuliert werden ($l = k^2 \rightarrow k = \sqrt{l} = l^{\frac{1}{2}}, l = k^{10} \rightarrow k = l^{\frac{1}{10}}, \ldots$)

Theorem: Einweg-Funktionen und BPP

Wenn Einweg-Funktionen existieren, dann gilt

$$\mathbf{BPP} \subseteq \mathbf{TIME}(2^{n^\epsilon}) \forall \epsilon > 0.$$

Blum-Micali: Derandomisierung von BPP III

Konditionales Ergebnis

- Überlegungen zur Blum-Micali-Methode setzen bislang *unbewiesene* (aber plausible) Existenz von Einwegfunktionen voraus
- Viele Aussagen der fortgeschrittenen Komplexitätstheorie folgen »Wenn-Dann-Muster«
- Praktische Bedeutung: Viele Techniken im realen Einsatz sind nur unter ähnlichen konditionellen Voraussetzungen tatsächlich wohlfundiert
- Einschätzung der Praktikabilität *erfordert solides theoretisches Grundlagenwissen*!

Zufall und Probleminstanz

- Vereinfachung bislang: $A(x, r) = A(r)$ bzgl. Unterscheidung zwischen echtem und Pseudo-Zufall
- Täuschung durch Pseudozufall muss für jede mögliche Eingabe gelingen:

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})\right| \leq \epsilon$$

wird ersetzt durch

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(x, g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(x, r) = \text{»Ja«})\right| \leq \epsilon' \ \forall x$$

Existenz problematischer Werte x

Wenn A nicht von g getäuscht wird, gibt es eine Instanz x, für die sich die Akzeptanzw'keiten deutlich unterscheiden.

Probleme

- Instanz x nicht notwendigerweise effizient konstruierbar
- Es existiert daher kein allgemeiner Meta-Tester: $\nexists A'(r) = A(x, r)$
- Verschärfung der Bedingungen: Tester soll weiterhin in **P** enthalten sein, aber dennoch problematische Instanzen »kennen«.
- Information über problematische Instanzen in zusätzlichem »advice« kodiert.
- Konsequenz: Pseudozufallsgenerator muss gegen Algorithmen mit zusätzlicher Information/Ratschlag bestehen

Komplexitätsklasse P/poly

- Poly-Zeit DTM mit Ratschlag (*advice*) a_n mit $|a_n| = \text{poly}(n)$ bei Eingabe w mit $|w| = n$
- Ratschlag hängt nur von Länge, aber *nicht* vom Inhalt der Eingabe ab
- Ratschlag a_n *nicht* notwendigerweise effizient zu berechnen!
- Äquivalent: Unterschiedliche spezialisierte Algorithmen werden angepasst an n eingesetzt

Anwendung

- A erhält problematische Instanz x als Advice (kann mit n Bits spezifiziert werden)
- Advice kann sich von n zu n' beliebig ändern
- PRNG muss gegen Algorithmen aus P/poly bestehen!

Komplexitätsklasse P/poly

- Poly-Zeit DTM mit Ratschlag (*advice*) a_n mit $|a_n| = \text{poly}(n)$ bei Eingabe w mit $|w| = n$
- Ratschlag hängt nur von Länge, aber *nicht* vom Inhalt der Eingabe ab
- Ratschlag a_n *nicht* notwendigerweise effizient zu berechnen!
- Äquivalent: Unterschiedliche spezialisierte Algorithmen werden angepasst an n eingesetzt

Anwendung

- A erhält problematische Instanz x als Advice (kann mit n Bits spezifiziert werden)
- Advice kann sich von n zu n' beliebig ändern
- PRNG muss gegen Algorithmen aus **P/poly** bestehen!

Eigenschaften von P/poly

- Lookup-Tabellen (2^n Einträge) ausgeschlossen wegen $|a_n| = \text{poly}(n)$
- Beschreibt Algorithmen mit Vorausberechnung
- Unrealistisches Berechnungsmodell (enthält nicht-rekursive Probleme!), aber obere Schranke für praktische effiziente Berechenbarkeit
- Äquivalent: Boole'sche Schaltkreise (*Boolean Circuits*), aufgebaut aus polynomial vielen AND/OR/NOT-Gattern. Formal: **PSIZE = P/poly**.
- Verbindung von »unrealistischer« und hochgradig praktischer Komplexitätsklasse
- **BPP** $\subseteq$ **P/poly** und **P** $\subseteq$ **P/poly**
- Gebräuchliches Bedrohungsmodell in vielen Krypto-Szenarien (bsp. Regenbogen-Tabellen)

Beweise/Details: Siehe *Moore & Mertens*, Kapitel 6.5.

Definition: Exponentiell starker PRNG

Sei g eine Funktion $\{0,1\}^k \to \{0,1\}^l$ mit $k = \mathcal{O}(\log l)$, die in Zeit poly(l) berechnet wird. g ist ein *exponentiell starker PRNG*, wenn für jeden nicht-uniformen Algorithmus A mit Laufzeit und Ratschlag $\mathcal{O}(l)$ gilt, dass

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})\right| = o(1).$$

- A hängt nicht von x ab, ist aber aus **P/poly**
- Laufzeit von g polynomial in $l \Rightarrow$ exponentielle Laufzeit in k
- Konstante $o(1)$ unterscheidet sich von bisheriger Definition

Exponentiell starke Pseudozufallsgeneratoren II

Exponentiell starker PRNG

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})\right| = o(1).$$

- Iteration über alle Seeds verhindern: Konstante in $k = \mathcal{O}(\log(n))$ hinreichend groß wählen.
- Beispiel: A läuft in Zeit $\mathcal{O}(l)$ mit $\mathcal{O}(l)$ Bits Ratschlag.
  - Sei $k = 10 \log l = \mathcal{O}(l)$
  - $\exists\, 2^k = l^{10}$ mögliche Seeds
  - Algorithmus kann nur $\mathcal{O}(l)$ Seeds überprüfen
  - Unterscheidung $g(s)$ von echt zufälligem String nur mit geringer W'keit $\mathcal{O}(l^{-9})$

Exponentiell starker PRNG

$$\left|\mathbb{P}_{s\in\{0,1\}^k}(A(g(s)) = \text{»Ja«}) - \mathbb{P}_{r\in\{0,1\}^l}(A(r) = \text{»Ja«})\right| = o(1).$$

- Abschwächung der Täuschungsbedingung: $|\cdot| = o(1)$ okay, solange Erfolgsw'keit $p > \frac{1}{2}$ ist.
- Beispiel: $p \geq \frac{2}{3}$ für A (Definition **BPP**!) $\Rightarrow$ Veränderung $< \frac{1}{6}$ akzeptabel, da $\frac{2}{3} - x > \frac{1}{2}$ für $x < \frac{1}{6}$.

Exponentiell starke Pseudozufallsgeneratoren IV

Theorem: BPP und exponentiell starke PRNGs

$$\exists \text{ exponentiell starker PRNG } \Rightarrow \mathbf{BPP} = \mathbf{P}$$

- Algorithmus $A \in \mathbf{BPP}$ mit Laufzeit $t = \text{poly}(n)$ soll unkonditionell derandomisiert werden
- Setze $l = t$ (obere Schranke für verwendbaren Zufall)
- Nicht-uniformer Algorithmus A braucht $n < l$ Bits an Ratschlag
- Laufzeit und Ratschlag sind daher $\mathcal{O}(l)$
- Theorem folgt unter Beachtung der vorher angegebenen Randbedingungen, indem man über alle $2^k = \text{poly}(n)$ Seeds iteriert und eine Mehrheitsentscheidung fällt.

Kleines Detail…

Existieren exponentiell starke PRNGs?

Theorem: BPP und exponentiell starke PRNGs

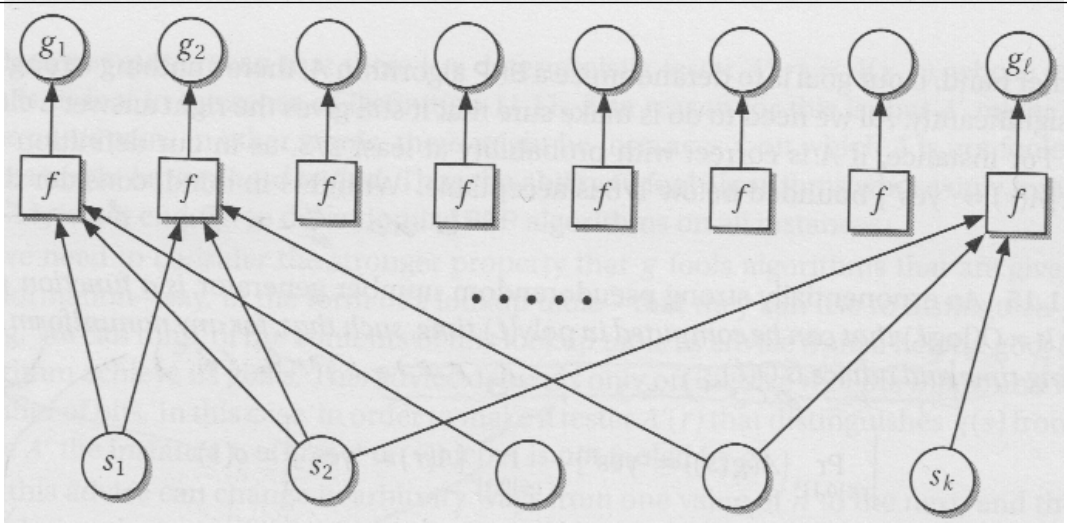
$$\exists \text{ exponentiell starker PRNG } \Rightarrow \mathbf{BPP} = \mathbf{P}$$

- Algorithmus $A \in \mathbf{BPP}$ mit Laufzeit $t = \text{poly}(n)$ soll unkonditionell derandomisiert werden
- Setze $l = t$ (obere Schranke für verwendbaren Zufall)
- Nicht-uniformer Algorithmus A braucht $n < l$ Bits an Ratschlag
- Laufzeit und Ratschlag sind daher $\mathcal{O}(l)$
- Theorem folgt unter Beachtung der vorher angegebenen Randbedingungen, indem man über alle $2^k = \text{poly}(n)$ Seeds iteriert und eine Mehrheitsentscheidung fällt.

Kleines Detail…

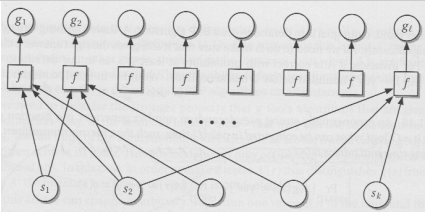
Existieren exponentiell starke PRNGs?

Konstruktion von Nisan-Wigderson I



Bildquelle: *Moore & Mertens*, Seite 550

Konstruktion von Nisan-Wigderson I



Zutaten

- Zufälliger Seed s der Länge k (d.h. $|s| = k$)
- Pseudozufallssequenz der Länge $l > k$ soll erzeugt werden.
- Gegeben: Random Bit-Generator $f : \{0,1\}^n \to \{0,1\}$

Algorithmus

- Wähle n-elementige Teilmenge aus Seed
- Wende Bit-Generator auf Teilmenge an
- Wiederhole die ersten beiden Schritte l-mal und konkateniere erzeugte Bits

Bildquelle: *Moore & Mertens*, Seite 550

Nisan-Wigderson: Formale Definition

Seien eine Funktion $f : \{0,1\}^n \to \{0,1\}$ und ein zufälliger Seed $s \in \mathbb{F}_2^k$ gegeben. Weiterhin existiere eine Familie von Teilmengen $S_i \subset [1,k]$, so dass $|S_i| = n \forall i \leq 1 \leq l$. Dann ist

$$g(s) = \langle f(S_1), f(S_2), \cdots, f(S_n) \rangle$$

ein pseudo-zufälliger String der Länge n, der den Anforderungen eines exponentiell harten Zufallsgenerators genügt ($f(S_j)$ ist als $f(s_{j_1}), f(s_{j_2}), \cdots, f(s_{j_n})$ zu verstehen, wenn $S_j = \{j_1, j_2, \ldots, j_n\}$).
Dies gilt unter den Voraussetzungen, dass

- f *exponentiell hart* ist
- $|S_i \cap S_j| \leq \alpha n$ für alle $i \neq j$, d.h. der Überlapp zwischen den Teilmengen ist beschränkt.

Definition: Exponentiell harte Funktion

Eine Funktion f ist exponentiell hart, wenn $\exists \epsilon > 0$, so dass für alle nicht-uniformen Algorithmen A mit Laufzeit und Ratschlag $\mathcal{O}(2^{\epsilon n})$ gilt, dass

$$\mathbb{P}_{x \in \{0,1\}^n}(A(x) = f(x)) = \frac{1}{2} + \mathcal{O}(2^{-\epsilon n})$$

- Selbst ein mächtiger nicht-uniformer **EXPTIME**-Algorithmus A kann f nicht besser als durch Raten invertieren!

Beweis (Struktur)

- Widerspruchsbeweis: $\exists$ Tester A, der zwischen zufälligem String und $g(s)$ unterscheidet.
- Konstruktion: Schrittweise Transition zwischen uniformem r und $g(s)$. Nach Annahme muss es ein Bit geben, dessen Austausch einen substantiellen Unterschied macht:

$$\mathbb{E}_{s,r}\Big(A(f(S_1),\dots,f(S_{i-1}),f(S_i),r_{i+1},\dots)- A(f(S_1),\dots,f(S_{i-1}),r_i,r_{i+1},\dots\Big) \geq C/l$$

- Aufspaltung Erwartungswert $s = \langle x, y\rangle$: $|x| = n$ Bits für S_i, $|y| = k - n$ Bits für Rest von s.

Beweis (Struktur)

- Widerspruchsbeweis: $\exists$ Tester A, der zwischen zufälligem String und $g(s)$ unterscheidet.
- Konstruktion: Schrittweise Transition zwischen uniformem r und $g(s)$. Nach Annahme muss es ein Bit geben, dessen Austausch einen substantiellen Unterschied macht:

$$\mathbb{E}_{s,r}\Big(A(f(S_1),\dots,f(S_{i-1}),f(S_i),r_{i+1},\dots)- A(f(S_1),\dots,f(S_{i-1}),r_i,r_{i+1},\dots\Big) \geq C/l$$

- Aufspaltung Erwartungswert $s = \langle x, y\rangle$: $|x| = n$ Bits für S_i, $|y| = k - n$ Bits für Rest von s.

Beweis (Struktur)

- $\mathbb{E}_{s,r}(\cdot) \to \mathbb{E}_{x,y,r}(\cdot)$
- Aufspaltung Erwartungswert $s = \langle x, y \rangle$: $|x| = n$ Bits für S_i, $|y| = k - n$ Bits für Rest von s.

Beweis (Struktur)

- $\mathbb{E}_{s,r}(\cdot) \to \mathbb{E}_{x,y,r}(\cdot)$
- Aufspaltung Erwartungswert $s = \langle x, y \rangle$: $|x| = n$ Bits für S_i, $|y| = k - n$ Bits für Rest von s.
- $\exists y, r$, so dass $\mathbb{E}(\cdot)$ invariant, wenn Wert nur über x berechnet wird.
- Konstruktion eines Algorithmus B, der Hard-Core Bit mit besserer W'keit als bloßem Raten berechnet, mit Hilfe von Advice zur Berechnung von $S_1, \ldots, S_{i-1}$: Da y fix, hängt $S_{j \neq i}$ nur von Bits aus x ab $\Rightarrow$ Reduzierung Rechenaufwand, da $|S_i \cap S_j| \leq \alpha n$ und Lookup-Tabelle durch Nichtuniformität möglich wird.
- Tester, der nach Annahme nicht existiert $\Rightarrow$ Widerspruch.
- Details: Siehe *Moore & Mertens*, Kapitel 11.4.5

Inhalt

Quantenmechanische Komplexitätsklassen I

BQP: Bounded Error Quantum Polynomial Time

Probleme, die in poly-Zeit auf Quanten-Turingmaschine mit Fehlerw'keit max 1/3 gelöst werden können.

- Faktorisierung, Diskrete Logarithmen, Pell-Gleichung, …

$$\mathbf{BPP} \subseteq \mathbf{BQP} \qquad \mathbf{BQP} \subseteq \mathbf{P}^{\#\mathbf{P}}$$
$$\mathbf{BQP} \subseteq \mathbf{PP} \qquad \mathbf{BQP} \subseteq \mathbf{EXPTIME}$$

EQP: Exact Quantum Polynomial-Time

Wie **BQP**, aber Fehlerw'keit 0

MA: Merlin-Arthur

- Arthur: Verifizierer mit RNG
- Merlin: *Unehrlicher* Beweiser/Orakel mit (essentiell) unbeschränkter Rechenleistung

Formale Definition

$L \in$ **MA**, wenn es eine poly-Zeit DTM M und Polynome p, q gibt, so dass (x: Eingabe, z: Zertifikat von Merlin, y: Zufallsbits von Arthur)

- $x \in L : \exists z \in \{0,1\}^{q(n)} \ \Pr_{y \in \{0,1\}^{p(n)}}(M(x,y,z) = 1) \geq 2/3$
- $x \notin L : \forall z \in \{0,1\}^{q(n)} \ \Pr_{y \in \{0,1\}^{p(n)}}(M(x,y,z) = 0) \geq 2/3$

Relation mit anderen Klassen

- **MA**: "**NP** mit probabilistischer Verifikation" (Arthur: **BPP**)
- **NP** $\subseteq$ **MA**, **BQP** $\subseteq$ **MA**

Quantenmechanische Komplexitätsklassen II

QMA: Quantum Mechanical Merlin-Arthur

- Wie MA, aber polynomial großer Quantenzustand als Input von Merlin
- Verifizierbarkeit in poly-Zeit auf Quantencomputer (QTM)

Formale Definition

Es gilt $L \in \mathbf{QMA}$, wenn es eine poly-Zeit-QTM V und ein Polynom $p(x)$ gibt, so, dass

- $\forall x \in L \exists |\psi\rangle$: W'keit V akzeptiert $(|x\rangle, |\psi\rangle) \geq 2/3$
- $\forall x \notin L \exists |\psi\rangle$: W'keit V akzeptiert $(|x\rangle, |\psi\rangle) < 1/3$

Bekannte Inklusionen

$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{MA} \subseteq \mathbf{QCMA} \subseteq \mathbf{QMA} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$

Quantenmechanische Komplexitätsklassen III

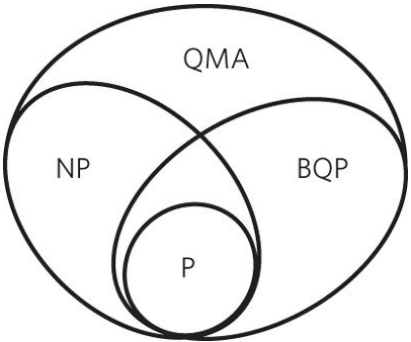
QMA: Eigenschaften

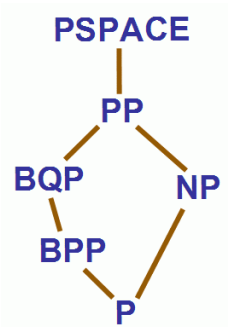
- **QMA**: Vollständige Probleme existieren
- k-lokale Hamiltonians (zusammengesetzt aus Operationen auf maximal k Qbits) sind **QMA**-vollständig für $k \geq 2$
- **QMA**-harter, physikalisch umsetzbarer Hamiltonian:

$$\hat{H} = \sum_i h_i \sigma_z^{(i)} + \sum_{i<j} J_{ij} \sigma_z^{(i)} \sigma_z^{(i)} + \sum_{i<j} K_{ij} \sigma_x^{(i)} \sigma_x^{(i)}$$

Umsetzbar durch adiabatisches Quantenrechnen!

Quantenmechanische Komplexitätsklassen IV





Quantenbits – Dirac'sche Ket-Darstellung

- Zwei Zustände: $|0\rangle, |1\rangle$
- Geometrische Interpretation: Siehe Tafel
- Erweiterbar auf n-Niveau-Systeme: $|0\rangle, |1\rangle, |2\rangle, \ldots, |n\rangle$

Quantenregister: Mehr-Qbit-Zustände

- 2-Qbit-Register: $|i\rangle \otimes |j\rangle \equiv |i, j\rangle \equiv |ij\rangle\,, i, j \in \mathbb{N}$
- 3-Qbit-Register: $|i\rangle \otimes |j\rangle \otimes |j\rangle \equiv |i, j, k\rangle \equiv |ijk\rangle\,, i, j, k \in \mathbb{N}$
- Allgemeines n-Qbit-Register:

$$\bigotimes_{k=1}^{n} |i_k\rangle = |i_1 i_2 \dots i_n\rangle\,, i_k \in \mathbb{N}\ \forall k \in [1, n]$$

Basis

- 2-Qbit-System: $\{|00\rangle\,, |01\rangle\,, |10\rangle\,, |11\rangle\}$
- n-Qbit-System: $\{|i_1 i_2 \dots i_n\rangle\}_{i_k \in \{0,1\}}$ ☞ 2^n Elemente
- Basis für 2-Qtrit-System (2 Drei-Niveau-Zustände)?

Berechnung von Superpositionen

- Sei $|x\rangle = \gamma_0 |0\rangle + \gamma_1 |1\rangle$ und $|y\rangle = \delta_0 |0\rangle + \delta_1 |1\rangle$.
- Berechnung von $|x\rangle \otimes |y\rangle \equiv |xy\rangle$: Komponentenweises "Ausmultiplizieren" (siehe Tafel)

Superposition

- Ein n-Qbit-Register R befindet sich allgemein im Zustand

$$R = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$$

mit $i \in [0, 2^n - 1]$ und $\alpha_i \in \mathbb{C}$ (Bits gesetzt entsprechend Wert der Binärzahl i). Normierung: $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$.

Zusammenfassung: Quantenschaltkreise I

Elementare Quantenoperatoren

- Hadamard-Gatter

$$\hat{H}\,|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad \hat{H}\,|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- Not-Gatter

$$\hat{N}\,|0\rangle = |1\rangle \qquad \hat{N}\,|1\rangle = |0\rangle$$

- Controlled Not-Gatter

$$\hat{CNOT}\,|x, y\rangle = |x, x \oplus y\rangle$$

Zusammenfassung: Quantenschaltkreise II

Operatoren/Matrizen

- Größen mit "Operator-Hütchen" ($\hat{X}$ statt X): *Operatoren*.
- Im endlichdimensionalen Fall durch Matrix darstellbar.
- *Achtung*: Rechenregeln unterschiedlich zu "normalen" skalaren Größen:
  - Typischerweise nicht kommutativ.
  - Typischerweise gilt $\hat{A}\hat{B} \neq \hat{B}\hat{A}$.

Vektorraum $\mathbb{C}^n$

- Qbits repräsentiert durch *Orthonormalbasis* für $\mathbb{C}^2$:

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

- $\alpha\,|0\rangle + \beta\,|1\rangle \equiv \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$
- Qtrits (3-Niveau-Zustände):

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad |2\rangle \equiv \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Quantenregister und Tensorprodukt

$$|\varphi\rangle \otimes |\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} \equiv \begin{pmatrix} a \cdot \begin{pmatrix} c \\ d \end{pmatrix} \\ b \cdot \begin{pmatrix} c \\ d \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a \cdot c \\ a \cdot d \\ b \cdot c \\ b \cdot d \end{pmatrix}$$

- Beispiele für Standardbasis: Siehe Tafel
- Achtung: Exponentieller Dimensionszuwachs! ☞ Simulation (vermutlich) nicht effizient durchführbar

Operatoren und Matrizen

Operationen auf n-Qbit-Registern werden durch komplexe $2^n \times 2^n$-Matrizen ($M \in \mathbb{C}^{2^n \times 2^n}$) beschrieben.

Beispiel: Darstellung von CNOT

- Wirkung des Operators auf die Basiszustände:

$$|00\rangle \to |00\rangle \qquad |01\rangle \to |01\rangle \qquad |10\rangle \to |11\rangle \qquad |11\rangle \to |10\rangle$$

- Darstellung als Matrix:

$$\text{CNOT} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- Beispiel und Illustration: Siehe Tafel

Quantenmechanische Zeitpropagation

Die zeitliche Entwicklung eines Quantensystems (ohne Messungen) ist durch unitäre Transformationen geregelt. Ausgangspunkt: Schödinger-Gleichung

$$i\hbar\partial_t |\Psi\rangle = \hat{H} |\Psi\rangle = \frac{\hat{p}^2}{2m} + \hat{V}(x,t)$$

Unitäre Matrix

Eine komplexe $n \times n$-Matrix A ist *unitär*, wenn

$$\bar{A}^{\mathrm{T}} \equiv A^{\dagger} = A^{-1}$$

gilt, wobei $\bar{A}$ die elementweise komplexe Konjugation und A^{-1} die inverse Matrix von A bezeichnet.

Illustration: Siehe Tafel

Quantennetzwerke und Mehr-Qbit-Hamiltonians

- Mehrere Qubits ☞ Tensorprodukt
- Konkatenation von Operationen ☞ Matrix-Multiplikation

Konstruktion: Siehe Tafel

Harmonischer Oszillator

- "Teilchen" in quadratischem Potential: $\hat{V} = \frac{1}{2}k\hat{x}^2$
- k: "Federkonstante", m: Masse
- Beschreibt Photonen (elektromagnetisches Feld) u.v.m.

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}k\hat{x}^2 = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega^2\hat{x}^2$$

Wellenfunktionen: Familie von Lösungen

$$\psi_n(x) = \frac{1}{\sqrt{2^n\, n!}} \cdot \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \cdot e^{-\frac{m\omega x^2}{2\hbar}} \cdot H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right) = |n\rangle$$

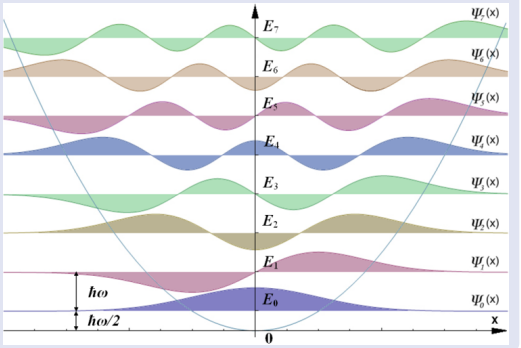
Energie

- $n = 0, 1, 2, 3, \ldots$: Diskrete Anregungsstufe
- Energieniveaus: $\hat{H} |n\rangle = E_n |n\rangle$

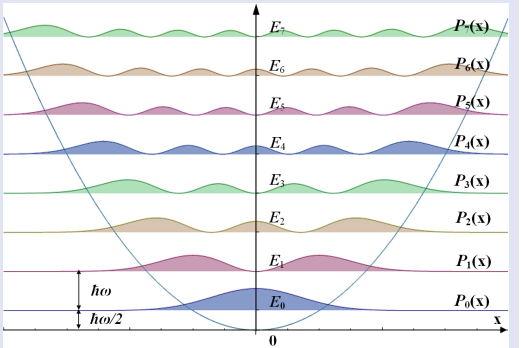
$$E_n = \hbar\omega \left(n + \frac{1}{2} \right)$$

Harmonischer Oszillator III

Wellenfunktion



Aufenthaltswahrscheinlichkeiten



Bidlquelle: Wikipedia

Klassisches Ising-Modell/QUBO

$$H(\vec{\sigma}) = H(\sigma_1, \sigma_2, \ldots, \sigma_N) = -\sum_{(i<j)} J_{ij}\sigma_i\sigma_j - \mu \sum_j h_j\sigma_j$$

- σ_i: Spin an Position i mit Einstellungen ± 1
- Hamilton-Funktion H: Energie (Temperatur) des Modells
- Verteilung 1D (Linie), 2D (Ebene), 3D (Raum) oder komplexer Graph
- J_{ij} Kopplung zwischen (benachbarten) Spins
  - $J_{i,j} > 0$: Ferromagnetisch (parellele Anordnung bevorzugt)
  - $J_{i,j} < 0$: Anti-Ferromagnetisch (abwechselnde Anordnung bevorzugt)
- h_i: Externes magnetisches Feld
  - $h_i > 0$: Anordnung nach oben (+) bevorzugt
  - $h_i < 0$: Anordnung nach unten (-) bevorzugt
- Statistische Physik: Energieverteilung, Phasenübergänge, etc.
- Rechnerisch: NP-hart.

Quantenmechanisches Ising-Modell

- Spinwerte durch Pauli-Operator $\hat{\sigma}_z$ ersetzen
- Hermitescher Operator (und unitär)! $\Rightarrow$ Hamiltonian
- $\hat{\sigma}_z$: Eigenwerte ± 1, Eigenkets $|0\rangle, |1\rangle$

$$\hat{H}(\vec{\hat{\sigma}}) = H(\hat{\sigma}_z^{(1)}, \hat{\sigma}_z^{(2)}, \ldots, \hat{\sigma}_z^{(N)}) = -\sum_{(i,j)} J_{ij} \hat{\sigma}_z^{(i)} \hat{\sigma}_z^{(j)} - \mu \sum_j h_j \hat{\sigma}_z^{(j)}$$

Ising-Modell & 3-Colouring III

3-Colouring als Spin-Glass

- Spins ±1 durch Pauli-Operator $\hat{\sigma}_z$ repräsentiert
- Dichotomische Variable: $\hat{x} := \frac{\mathbb{1}+\hat{\sigma}_z}{2}$ (EW + EV: Tafel?)
- Nebenbedingungen durch Energiepenalties abbilden
- Knoten v als Menge binärer Variablen $\{\hat{x}_{v,i}\}$ für jede Farbe i repräsentieren

Nebenbedingungen

1. Jeder Knoten erhält genau eine Farbe: Minimiere $\sum_v \left(1 - \sum_i \hat{x}_{v,i}\right)^2$
2. Knoten an Kante erhalten unterschiedliche Farbe: Minimiere $\sum_{(u,v)\in E} \sum_i \hat{x}_{u,i}\hat{x}_{v,i}$

3-Colouring als Spin-Glass

- Spins ±1 durch Pauli-Operator $\hat{\sigma}_z$ repräsentiert
- Dichotomische Variable: $\hat{x} := \frac{\mathbb{1}+\hat{\sigma}_z}{2}$ (EW + EV: Tafel?)
- Nebenbedingungen durch Energiepenalties abbilden
- Knoten v als Menge binärer Variablen $\{\hat{x}_{v,i}\}$ für jede Farbe i repräsentieren

Lösung

Zustand minimaler Energie ($E = 0$) $\Rightarrow$ Alle Bedingungen erfüllt $\Rightarrow$ Lösung

$$\hat{H} = C \sum_v \left(1 - \sum_i \hat{x}_{v,i}\right)^2 + C \sum_{(u,v)\in E} \sum_i \hat{x}_{u,i}\hat{x}_{v,i}$$

Das adiabatische Theorem I

Adiabatischer Prozess

- "Langsame" Transformation von Grundzustand $\hat{H}_0$ nach Grundzustand $\hat{H}_P$
- Starthamiltonian: Grundzustand
- Adiabatische (langsame) Änderung, *quasistatisch*: Grundzustand mit minimaler Energie → passt sich schrittweise an neuen Hamiltonian an.
- Nicht-adiabatische Änderung: Grundzustand bleibt erhalten → typischerweise nicht minimale-Energie für neuen Hamiltonian

Das adiabatische Theorem II

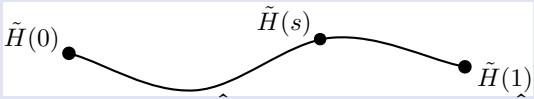
Spingläser: Typischer Start-Hamiltonian

$$\hat{H}_0 = -h_0 \sum_i \hat{\sigma}_x^{(i)}$$

- Grundzustand: $|+\rangle^{\otimes N}$. Tafel?
- Superposition über *alle* Basiszustände
- Einfach zu präparieren

Das adiabatische Theorem III

Stetig zeitlich variierender Hamiltonian



$$\hat{H}(0) = \hat{\tilde{H}}(0) \qquad \hat{H}(t) = \hat{\tilde{H}}(t/T) \qquad \hat{H}(T) = \hat{\tilde{H}}(1)$$

Diagonalisierung

- Matrixdarstellung von $\hat{H}$ *diagonal* in Energieeigenbasis: $\hat{H} = \sum_n E_n \, |E_n\rangle \, \langle E_n|$
- Funktioniert zu jedem Zeitpunkt s für $\hat{\tilde{H}}(s)$:

$$\hat{\tilde{H}}(s) = \sum_n E_n(s) \, |E_n(s)\rangle \, \langle E_n(s)|$$

  wobei $E_0(s) < E_1(s) \leq E_2(s) \leq \cdots \leq E_N(s)$. Eigenbasis zeitabhängig!
- Wenn $|\psi(0)\rangle = |E_0(0)\rangle$: $\lim_{T\to\infty} |\, \langle E_0(1)|\psi(T)\rangle \,|^2 = 1$

Laufzeit – Physik-Sicht

- Abhängig von Energie-Lücke

$$\Delta(s) = E_1(s) - E_0(s) \qquad \Delta = \min_{s\in[0,1]} \Delta(s)$$

- Grobe Schätzung (Standard-QM-Lehrbuch)

$$T \ll \frac{\Gamma^2}{\Delta^2} \qquad \Gamma^2 = \max_{s\in[0,1]} ||\dot{\hat{H}}(s)^2||$$

- Feinere Schätzung (Teufel 2003, Jansen et al. 2007)

$$T \geq \frac{4}{\epsilon}\left(\frac{||\dot{\hat{H}}(0)||}{\Delta(0)^2} + \frac{||\dot{\hat{H}}(1)||}{\Delta(1)^2} + \int_0^1 ds\left(10\frac{||\dot{\hat{H}}^2||}{\Delta^3} + \frac{||\ddot{\hat{H}}||}{\Delta}\right)\right)$$

mit $|| \, |\psi(T)\rangle - |E_0(1)\rangle \, || \leq \epsilon$

Das adiabatische Theorem V

Laufzeit – Informatik-Sicht

Wie groß ist Δ?

$$T \ll \frac{\Gamma^2}{\Delta^2} \text{ mit } \Gamma^2 = \max_{s\in[0,1]} ||\dot{\tilde{H}}(s)^2||$$

- $\Delta = 1/\text{poly(n)}$: Effizient
- $\Delta = 1/\exp(n)$: Nicht effizient

Δ ermitteln

- Aus physikalischer Struktur des Problem berechnen (typischerweise kompliziert)
  - Grover-Suche: $\Delta = \frac{1}{\sqrt{N}}$
  - Transverses Ising-Modell: $\Delta \propto \frac{1}{n}$
  - Fisher-Problem: $\Delta \approx \exp(-c\sqrt{n})$
- Messen bzw. empirisch bestimmen (einfach(?))