Randomisierte Algorithmen

Randomisierte Algorithmen

Ein randomisierter Algorithmus A wählt in jedem Schritt eine von mehreren Optionen gleichwahrscheinlich aus.

(a) Die Wahrscheinlichkeit $\mathrm{pr}[B]$ einer Berechnung B von A ist das Produkt der Wahrscheinlichkeiten über alle Schritte der Berechnung.

(b) Die Akzeptanz-Wahrscheinlichkeit für eine Eingabe x ist

die Summe der Wahrscheinlichkeiten akzeptierender Berechnungen für x.

(c) Die von A akzeptierte Sprache $L(M)$ ist

$$L(A) = \{x \in \{0,1\}^* : A \text{ akzeptiert } x \text{ mit W-keit mindestens } \frac{1}{2} \}.$$

Für $0 \leqslant p < q \leqslant 1$ heißt A ein (p,q)-Algorithmus, wenn A jede Eingabe mit Wahrscheinlichkeit höchstens p oder mindestens q akzeptiert.

Randomisierte Algorithmen: Beispiele

- Randomisierte Algorithmen
  - Quicksort mit zufälliger Pivot-Wahl
  - Randomisierung für geometrische Probleme
    - Volumenmessung von konvexen Mengen
    - das „Closest-Point-Problem“
  - Randomisierte Primzahltests:
    Die Algorithmen von Rabin und Solovay-Strassen
  - Anwendungen im Entwurf von Online-Algorithmen:
    Randomisierung gegen die Zukunft
- Randomisierte Datenstrukturen:
  Bloom-Filter, randomisierte Skip-Listen, Treaps, universelles Hashing
- Anwendungen in der Kryptographie:
  Zufällige Wahl eines öffentlichen Schlüssels
- Randomisierung in Beweissystemen:
  Interaktive Beweise und probabilistically-checkable-proofs

Probabilistische Komplexitätsklassen (1/2)

(a) Für die Funktion $t : \mathbb{N} \to \mathbb{N}$ und alle Paare (p, q) mit $0 \leqslant p < q \leqslant 1$ ist

$$\mathtt{RTIME}_{p,q}(t) := \{\, L \subseteq \{0,1\}^* \quad : \quad \text{es gibt einen } (p,q)\text{-Algorithmus } A \text{ mit } L(A) = L \text{ und } t_A = \mathcal{O}(t) \,\}$$

die Komplexitätsklasse aller in Zeit $\mathcal{O}(t)$ berechenbaren Sprachen.

(b) Die Klasse $\mathtt{RTIME}^*_{p,q}(t)$ wird analog definiert, allerdings wird die Worst-Case-Laufzeit durch die erwartete Laufzeit ersetzt.

(c) Die Komplexitätsklasse

$$\mathtt{BPP} := \bigcup_{k \in \mathbb{N}} \mathtt{RTIME}_{\frac{1}{3},\frac{2}{3}}(n^k)$$

(*Bounded-Error-Probabilistic-Polynomial-Time*) ist die Klasse aller mit beschränktem Fehler, in polynomieller Zeit akzeptierbaren Sprachen.

Probabilistische Komplexitätsklassen (2/2)

Weitere fundamentale Klassen:

1. $\texttt{WeakBPP} := \bigcup_{k\in\mathbb{N}} \texttt{RTIME}_{\frac{1}{2}-n^{-k},\frac{1}{2}+n^{-k}}(n^k)$ und
   $\texttt{StrongBPP} := \bigcap_{r\in\mathbb{N}} \bigcup_{k\in\mathbb{N}} \texttt{RTIME}_{2^{-n^r},1-2^{-n^r}}(n^k)$
2. $\texttt{ZPP} := \bigcup_{k\in\mathbb{N}} \texttt{RTIME}^*_{0,1}(n^k)$ (*Zero-Error-Probabilistic-Polynomial-Time*) die Klasse aller fehlerfrei durch randomisierte Algorithmen in polynomieller erwarteter Laufzeit akzeptierbaren Sprachen,
3. $\texttt{RP} := \bigcup_{k\in\mathbb{N}} \texttt{RTIME}_{0,\frac{1}{2}}(n^k)$ (*Randomized-Polynomial-Time*) die Klasse aller durch randomisierte Algorithmen A in polynomieller Zeit akzeptierbaren Sprachen, wobei A beim Verwerfen fehlerfrei sein muss,
4. $\texttt{coRP} := \bigcup_{k\in\mathbb{N}} \texttt{RTIME}_{\frac{1}{2},1}(n^k)$ (*co-Randomized-Polynomial-Time*) die Klasse aller durch randomisierte Algorithmen A in polynomieller Zeit akzeptierbaren Sprachen, wobei A beim Akzeptieren fehlerfrei sein muss,
5. $\texttt{PP} := \bigcup_{k\in\mathbb{N}} \texttt{RTIME}_{\frac{1}{2},\frac{1}{2}}(n^k)$ (*Probabilistic-Polynomial-Time*) die Klasse aller durch randomisierte Algorithmen in polynomieller Zeit akzeptierbaren Sprachen, wobei die Akzeptanzwahrscheinlichkeit für keine Eingabe genau $\frac{1}{2}$ betragen darf.

Beziehungen zwischen den Komplexitätsklassen

Die folgenden Inklusionen gelten:

(a) $\mathtt{P} \subseteq \mathtt{ZPP} = \mathtt{RP} \cap \mathtt{coRP} \subseteq \mathtt{RP} \subseteq \mathtt{NP} \subseteq \mathtt{PP} \subseteq \mathtt{PSPACE} \subseteq \mathtt{EXP}$ und $\mathtt{coRP} \subseteq \mathtt{coNP}$.

(b) $\mathtt{RP}, \mathtt{coRP} \subseteq \mathtt{BPP} \subseteq \mathtt{PP}$.

Bounded Error Probabilistic Polynomial Time

Die Klasse BPP

(a) Amplifizierung: $\mathtt{WeakBPP} = \mathtt{BPP} = \mathtt{StrongBPP}$.

Eine schwach von $\frac{1}{2}$ weg-getrennte Fehlerwahrscheinlichkeit ist ausreichend, um „fast-fehlerfrei“ zu rechnen.

(b) $\mathtt{BPP} \subset \mathtt{P/poly}$

Randomisierte Algorithmen lassen sich effizient derandomisieren, wenn nicht-uniforme Algorithmen erlaubt sind.

(c) $\mathtt{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$

Sprachen mit effizienten randomisierten Algorithmen liegen in der polynomiellen Hierarchie.

Später: Wenn es eine Sprache $L \in E$ gibt, so dass L nur durch Schaltkreise der Größe $2^{\Omega(n)}$ berechenbar ist, dann folgt

$$\mathtt{P} = \mathtt{BPP}.$$

Randomisierung in der Kryptographie, im Entwurf von Online-Algorithmen oder im Entwurf von Beweissystemen ist hingegen nicht durch Determinismus ersetzbar.

Fehlerreduktion

WeakBPP = StrongBPP (1/2)

Es gelte $L = L(A)$ für einen randomisierten Algorithmus A mit

- Laufzeit $t_A(n) = \mathcal{O}(n^k)$ und
- Schwellenwerten $\frac{1}{2} - n^{-k}, \frac{1}{2} + n^{-k}$ für die Akzeptanzwahrscheinlichkeiten.

Ein „starker“ Algorithmus A^* für L:

- A^* simuliert A auf Eingabe w genau ℓ-mal
- und akzeptiert x wenn A mindestens $\lceil \frac{\ell}{2} \rceil$-mal akzeptiert.

Sei $X_i \in \{0, 1\}$ die Zufallsvariable mit $X_i = 1$ genau dann, wenn w in der $i$ten Simulation akzeptiert wird. Für $\mu = \sum_{i=1}^{\ell} \mathbb{E}[X_i]$ folgt

$$\text{pr}\left[\left|\sum_{i=1}^{\ell} X_i - \mu\right| \geqslant \beta \cdot \mu\right] \leqslant \exp(-\Omega(\beta^2 \cdot \mu))$$

mit der Chernoff-Schranke.

Fall 1: $x \in L$. Dann ist $\mu \geqslant \ell(\frac{1}{2} + n^{-k})$ und $1 - \frac{\ell}{2\mu} \geqslant \frac{\ell}{\mu} \cdot n^{-k} \geqslant n^{-k}$ folgt.

- Setze $\beta := 1 - \frac{\ell}{2 \cdot \mu}$, und es ist $\mu \cdot (1 - \beta) = \frac{\ell}{2}$.
- Des weiteren ist $\beta \geqslant n^{-k}$.

Wenn A^* verwirft, dann sind höchstens $\frac{\ell}{2}$ Berechnungen akzeptierend $\Longrightarrow$

$$\begin{aligned} \text{pr}[\, A^* \text{ verwirft } x \,] &\leqslant \text{pr}[\sum_{i=1}^{\ell} X_i \leqslant \frac{\ell}{2}] = \text{pr}[\sum_{i=1}^{\ell} X_i \leqslant \mu \cdot (1 - \beta)] \\ &\leqslant \exp(-\Omega(\beta^2 \cdot \mu)) \leqslant \exp(-\Omega(\frac{\ell}{n^{2k}})), \end{aligned}$$

denn $\beta^2 \cdot \mu \geqslant n^{-2k} \cdot \frac{\ell}{2}$ folgt aus $\beta \geqslant n^{-k}$ und $\mu \geqslant \frac{\ell}{2} \Longrightarrow$ Für $\ell = \Omega(n^{2k+r})$ wird A^* die Eingabe x mit Wahrscheinlichkeit höchstens 2^{-n^r} verworfen.

Fall 2: $x \notin L$. Die Argumentation verläuft analog.

Nicht-uniforme Derandomisierung,

$\mathtt{BPP} \subset \mathtt{P/poly}$

Nicht-uniforme Derandomisierung: $\mathtt{BPP} \subset \mathtt{P/poly}$

Es ist $\mathtt{BPP} = \mathtt{StrongBPP}$.

Sei $L \in \mathtt{BPP}$. Dann ist $L = L(A)$ für einen randomisierten Algorithmus A mit Laufzeit $t_A = \text{poly}(n)$ und Fehlerwahrscheinlichkeit höchstens 2^{-2n}.

- A mit Eingabe x lässt sich als *deterministischer* Algorithmus auffassen, der die Eingaben x und $y =$ eine Folge von Zufallsbits erhält.
- Für jede Eingabe $x \in \{0,1\}^n$ wird A für einen Anteil von höchstens 2^{-2n} aller Zufallsstrings y die falsche Antwort liefern.
  - Für wie viele Zufallsstrings erhält A die falsche Antwort auf *irgendeiner* Eingabe in $\{0,1\}^n$? Für höchstens den Anteil

$$2^n \cdot 2^{-2n} = 2^{-n} < 1$$

    aller Zufallsstrings.
  - Es gibt einen Zufallsstring y_n für den A auf *allen* Eingaben der Länge n die richtige Klassifikation bestimmt.
- L wird von einem nicht-uniformen, deterministischen Algorithmus erkannt, der für Eingaben der Länge n mit y_n als Zufallsstring arbeitet. □

BPP und die polynomielle Hierarchie

$\mathtt{BPP}$ ist unter Komplementbildung abgeschlossen $\Longrightarrow$ Zeige: $\mathtt{BPP} \subseteq \Sigma_2^p$.

- $\mathtt{BPP} = \mathtt{StrongBPP} \Longrightarrow$ für $L \in \mathtt{BPP}$ gibt es einen randomisierten Algorithmus A mit Fehlerwahrscheinlichkeit höchstens 2^{-n}.
- Simuliere A durch einen effizienten Σ_2^p-Algorithmus A^*.

Für Eingabe x

- rät A^* in der Ratephase „Zufallstrings“ $y_1, \ldots, y_k$ und
- setzt in der Verifikationsphase einen String y „dagegen“.
  - A^* akzeptiert die Eingabe $x :\Longleftrightarrow$ Es gibt Zufallstrings $y_1, \ldots, y_k$, so dass es *für jedes* y mindestens ein i gibt und $A(x, y \oplus y_i)$ akzeptiert

A^* versucht mit der Wahl von y sicherzustellen, dass nicht ungerechtfertigter Weise akzeptiert wird.

A^* akzeptiert die Eingabe $x :\Longleftrightarrow$ Es gibt Zufallstrings $y_1, \ldots, y_k$, so dass es *für jedes* y mindestens ein i gibt und $A(x, y \oplus y_i)$ akzeptiert

Fall 1: $x \in L$. Sage, dass y_i falsch für x und y ist, wenn $A(x, y \oplus y_i)$ verwirft.

- Es ist $\mathrm{pr}_{y_i}[\, y_i \text{ ist falsch für } x \text{ und } y \,] \leqslant \frac{1}{2}$.
- Würfel die y_i unabhängig voneinander aus $\Longrightarrow$

$$\mathrm{pr}_{y_1,\ldots,y_k}[\, \text{alle } y_i \text{ sind falsch für } x \text{ und } y \,] \leqslant 2^{-k}.$$

Sei $|x| = n$ und A arbeite in Laufzeit $t_A(n)$. Dann folgt

$$\mathrm{pr}_{y_1,\ldots,y_k}[\, \text{es gibt } y, \text{ so dass alle } y_i \text{ falsch für } x \text{ und } y \text{ sind} \,] \leqslant 2^{t_A(n)-k}.$$

Wähle $k := t_A(n) + 1$: Es gibt Zufallstrings $y_1, \ldots, y_k$, so dass es *für jedes* y mindestens ein i gibt und $A(x, y \oplus y_i)$ akzeptiert $\Longrightarrow$

$$A^* \text{ akzeptiert } x.$$

A^* akzeptiert die Eingabe $x :\Longleftrightarrow$ Es gibt Zufallstrings $y_1, \ldots, y_k$, so dass es *für jedes* y mindestens ein i gibt und $A(x, y \oplus y_i)$ akzeptiert

Fall 2: $x \notin L$. Sage, dass y_i falsch für x und y ist, wenn $A(x, y \oplus y_i)$ akzeptiert.

Die Strings $y_1, \ldots, y_k$ seien beliebig. Da Fehlerwahrscheinlichkeit $\leqslant 2^{-n}$, gilt

$$\mathrm{pr}_y[\, y_i \text{ ist falsch für } x \text{ und } y \,] \leqslant 2^{-n},$$

für jedes $i \leqslant k \Longrightarrow$

$$\mathrm{pr}_y[\, \text{es gibt } i \text{ und } y_i \text{ ist falsch für } x \text{ und } y \,] \leqslant k \cdot 2^{-n} < 1,$$

denn k ist polynomiell in n.

Für alle $y_1, \ldots, y_k$ gibt es y, so dass $A(x, y \oplus y_i)$ für jedes i verwerfend ist $\Longrightarrow$

A^* verwirft Eingabe x.

Randomisierte Reduktionen

UNIQUE-SAT

Im „*Promise-Problem*“ UNIQUE-SAT wird versprochen, dass eine Formel ψ in konjunktiver Normelform genau eine erfüllende Belegung α besitzt.

Bestimme α.

Wie schwierig ist UNIQUE-SAT? Eine Sprachenversion von UNIQUE-SAT:

> *Für eine KNF ψ ist zu bestimmen, ob ψ eindeutig erfüllbar oder unerfüllbar ist. Ist ψ mehrdeutig erfüllbar, dann ist jede Antwort korrekt.*

Es gibt eine effizient berechenbare, randomisierte Reduktion T, die KNF-Formeln ψ mit n Variablen auf KNF-Formeln $T(\psi)$ mit den folgenden Eigenschaften abbildet:

1. Wenn ψ zu KNF-SAT gehört, dann wird $T(\psi)$ genau eine erfüllende Belegung mit Wahrscheinlichkeit mindestens $\frac{8}{n}$ besitzen.
2. Wenn ψ nicht zu KNF-SAT gehört, dann wird $T(\psi)$ mit Wahrscheinlichkeit 1 unerfüllbar sein.

Wie arbeitet T?

Die KNF ψ besitze genau n Variablen.

1. T wählt $i \in \{0, 1, \ldots, n-1\}$ zufällig aus:
   - T „wettet“ darauf, dass die Anzahl erfüllender Belegungen mindestens 2^i aber weniger als 2^{i+1} beträgt.
   - Sollte ϕ erfüllbar sein, dann wird T diese Wette natürlich nur mit W-keit $\frac{1}{n}$ gewinnen, aber eine kleine Gewinnwahrscheinlichkeit ist akzeptabel.
2. Für $m := i + 2$ wird eine zufällige Hashfunktion

$$h(x) = A \cdot x + b \text{ mit } A \in \mathbb{Z}_2^{m \times n} \text{ und } b \in \mathbb{Z}_2^m$$

ausgewürfelt. Für A, b, die Belegung x und *Hilfsvariablen* y ist

$$T(\psi)(x, y) := \psi(x) \wedge \phi(x, A, b, y).$$

T funktioniert, weil . . .

Annahme: ψ besitze b erfüllende Belegungen mit $2^{m-2} \leqslant b \leqslant 2^{m-1}$.
U_x ist das Ereignis, dass x die einzige erfüllende Belegung mit $h(x) = 0$ ist.

Mit welcher Wahrscheinlichkeit wird U_x für irgendeine Belegung x erfüllt?

$$\begin{aligned} \mathrm{pr}[\,U_x\,] &\geqslant \mathrm{pr}_{h\in\mathcal{A}_{m,n}}[\,h(x)=0\,] - \sum_{x',\psi(x') \text{ ist wahr}} \mathrm{pr}_{h\in\mathcal{A}_{m,n}}[\,h(x)=h(x')=0\,] \\ &\geqslant \frac{1}{2^m} - \frac{2^{m-1}}{2^{2m}} \geqslant \frac{1}{2^{m+1}}. \end{aligned}$$

Also folgt

$$\mathrm{pr}[\,\exists x \in \{0,1\}^n\, U_x\,] = \sum_{x,\psi(x) \text{ ist wahr}} \mathrm{pr}[U_x] \geqslant \frac{2^{m-2}}{2^{m+1}} \geqslant \frac{1}{8}.$$

- Ist ψ erfüllbar, dann ist $T(\psi)$ mit W-keit $\frac{1}{8n}$ erfüllbar.
- Ist ψ unerfüllbar, dann ist natürlich auch $T(\psi)$ unerfüllbar. □

Zusammenfassung

1. Randomisierung besitzt eine drastische Fehlerreduktion, *wenn* der Fehler merklich kleiner als $\frac{1}{2}$ ist.
   - $\mathtt{WeakBPP} = \mathtt{StrongBPP}$,
   - aber $\mathtt{NP} \subseteq \mathtt{PP}$.
2. Es ist $\mathtt{BPP} \subseteq \mathtt{P/poly}$. Die Inklusion $\mathtt{NP} \subseteq \mathtt{BPP}$ ist nicht plausibel, denn
   - $\mathtt{NP} \subseteq \mathtt{BPP} \subseteq \mathtt{P/poly}$.
   - Das Theorem von Karp und Lipton besagt, dass die polynomielle Hierarchie $\mathtt{PH}$ zu Σ_2^p kollabiert, wenn $\mathtt{NP} \subseteq \mathtt{P/poly}$ gilt.
3. Vermutlich gilt sogar $\mathtt{P} = \mathtt{BPP}$: Siehe dazu die folgende Diskussion über Pseudo-Random-Generatoren. Randomisierung zeigt aber seine Stärke
   - im Entwurf einfacher und schneller Offline-Algorithmen,
   - im Entwurf von Online-Algorithmen,
   - in der Kryptographie (Wahl des öffentlichen Schlüssels)
   - in interaktiven Beweisen (später).
4. Mit einer randomisierten Reduktion kann die Schwierigkeit von Promise-Problemen wie etwa UNIQUE-SAT nachgewiesen werden.