5. Algebraische Methoden
    (a) Verifikation Matrizenmultiplikation
    (b) Tool: Schwarz-Zippel Theorem
    (c) String Matching Problem
    (d) Perfekte Matchings
        i. bipartite Graphen
        ii. beliebige, gewichtete Graphen
        iii. Tool: Isolation Lemma
6. Derandomisierung
    (a) Methode der bedingten Erwartungswerte
    (b) Erweiterung: Pessimistische Schätzer
    (c) Methode der k-fach unabhängigen Zufallsbits
7. Online-Algorithmen
    (a) Motivation & Definition der kompetitiven Analyse
    (b) Paging Problem

Motivation

Warum betrachtet man randomiserte Algorithmen?

- Randomisierte Algorithmen sind oftmals leichter zu beschreiben / zu implementieren als deterministische Algorithmen.
- Für einige Probleme kennt man randomisierte Algorithmen, die beweisbar schneller sind als jeder deterministische Algorithmus.

Nachteile:

- Man kann nur erhält nur Aussagen, der Form:
  — Algorithmus liefert (höchst-)wahrscheinlich das richtige Ergebnis, und/oder
  — Erwartungswert der Laufzeit des Algorithmus verhält sich gutartig.
- Analyse von randomisierten Algorithmen ist oftmals technisch recht trickreich.

Algorithmenanalyse

Las Vegas Algorithmus:

Antwort ist immer korrekt, Laufzeit ist eine Zufallsvariable

Monte Carlo Algorithmus:

Antwort darf (manchmal) inkorrekt sein, Laufzeit ist deterministisch

Komplexitätsklassen

RP (Randomised Polynomial Time):

$L \in$ **RP** $\iff$ Es gibt einen randomisierten Algorithmus A mit polynomieller Laufzeit (deterministisch!), so dass

$$\begin{aligned} x \in L \quad &\Rightarrow \quad \mathsf{Pr}[A(x) = \mathsf{true}] \geq \frac{1}{2} \\ x \notin L \quad &\Rightarrow \quad \mathsf{Pr}[A(x) = \mathsf{true}] = 0. \end{aligned}$$

(D.h. Es gibt einen Monte Carlo Algorithmus mit Fehlerw'keit $1/2$.)

ZPP (Zero Error Probabilistic Polynomial Time):

$L \in$ **ZPP** $\iff$ Es gibt einen randomisierten Algorithmus A mit erwarteter polynomieller Laufzeit. (D.h. Es gibt einen Las Vegas Algorithmus .)

Tools

Markov-Ungleichung

Für jede Zufallsvariable X mit $X \geq 0$ gilt

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

Chebyshev-Ungleichung

Für jede Zufallsvariable X gilt

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

Coupon Collector Problem

Gegeben: Kiste mit n verschiedenen Bällen; Ziehen mit Zurücklegen

$\Longrightarrow$ Im Erwartungswert muss man $n \ln n$ Bälle ziehen, bis jeder Ball mindestens einmal gezogen wurde.

Tools (2)

First Moment Method

(X_n) Folge von Zufallsvariablen mit Werten in $\mathbb{N}_0$, dann gilt

$$\mathbb{E}[X_n] \to 0 \qquad \Longrightarrow \qquad Pr[X_n = 0] \to 1$$

Second Moment Method

(X_n) Folge von Zufallsvariablen mit Werten in $\mathbb{N}_0$, dann gilt

$$\mathbb{E}[X_n] \to \infty \quad \underline{\text{und}} \quad \mathbb{E}[(X_n)^2] = (1 + o(1))(\mathbb{E}[X_n])^2$$

$$\Longrightarrow \qquad Pr[X_n = 0] \to 0$$

Tools (3)

Lovász Local Lemma

Gegeben Ereignisse $E_1, \ldots, E_n$ und Mengen $S_i \subseteq \{E_1, \ldots, E_n\}$ und Zahlen $0 \leq x_i \leq 1$, so dass

- E_i unabhängig von der Menge der Ereignisse in S_i,
- Pr$[E_i] \leq x_i \prod\limits_{E_j \notin S_i} (1 - x_j)$

dann gilt

$$\text{Pr}[\ \bigcap_{i=1}^{n} \overline{E_i}\ ] \geq \prod_{i=1}^{n} (1 - x_i).$$

Anwendungsbeispiel: k-SAT

(1) Jede k-SAT Formel mit weniger als 2^k Klauseln ist erfüllbar.

Bew.: First Moment Method

(2) Jede k-SAT Formel, in der jede Variable in höchstens $\frac{2^k}{e} - 1$ Klauseln vorkommt, ist erfüllbar.

Bew.: Lovász Local Lemma

Anwendungsbeispiel: Median

Es gibt einen Las Vegas Algorithmus, der den Median einer n-elementigen Menge in erwarteter Laufzeit $O(n)$ bestimmt.

Der Algorithmus benötigt (im Erwartungswert) $\frac{3}{2}n$ Vergleiche.
Zum Vergleich: Jeder deterministische Algorithmus benötigt mindestens $(2 - \epsilon)n$ Vergleiche.

Bew.: Chebyshev-Ungleichung

Anwendungsbeispiel: Verteilung von Aufgaben an Server

Modellierung:

Gegeben: m Bälle (= Jobs)

n Urnen (= Server)

Gesucht: Protokoll, dass die Bälle möglichst gleichmäßig verteilt — ohne zentrale Instanz

Randomisiertes Protokoll: Wähle für jeden Ball eine Urne zufällig.

$X_i :=$ Anzahl Bälle in i-ter Urne. Dann gilt:

$$\max_i X_i \approx \begin{cases} \frac{\ln n}{\ln \ln n} & m = \Theta(n) \\ \frac{m}{n} + \Theta(\sqrt{(m \ln n)/n}) & m \gg n \end{cases}$$

Bew.: First and Second Moment Method

Verbessertes Protokoll: [Power of two Choices]

Wähle für jeden Ball *zwei* Urnen zufällig. Platziere Ball in derjenigen Urne, die weniger Bälle enthält.

$$\max_i X_i \approx \frac{m}{n} + O(\ln \ln n).$$

Bew.: Siehe Literatur.

Anwendungsbeispiel: Permutation Routing

Permutation Routing Protocol:

1. Phase: $\forall v \in V$:

- Wähle $r(v) \in V$ zufällig.
- Verwende Bit-Fixing Strategie um Paket von v nach $r(v)$ zu schicken.
- Warte, bis (insgesamt) genau $7n$ Runden vergangen sind.

2. Phase: $\forall v \in V$:

- Verwende Bit-Fixing Strategie um Paket von $r(v)$ nach $\pi(v)$ zu schicken.

Satz: Mit W'keit $\geq 1 - 2^{-4n}$ erreicht bei diesem Protokoll jedes Paket spätestens nach 14n Runden sein Ziel.

Bew.: Chernoff Schranken

Tools (4)

Eine Folge von Zufallsvariablen $Z_0, Z_1, \ldots, Z_n, \ldots$ heißt **Martingal**, falls $\mathbb{E}[Z_k \mid Z_1, \ldots, Z_{k-1}] = Z_{k-1}$ für alle k.

Ungleichung von Azuma-Hoeffding

Sei $Z_0, Z_1, \ldots, Z_n, \ldots$ ein Martingal mit $\left|Z_k - Z_{k-1}\right| < c_k$ für alle k, dann gilt für alle t, λ:

$$\Pr\left[\ \left|Z_t - Z_0\right| \geq \lambda\ \right] \ \leq\ 2e^{-\frac{1}{2}\lambda^2 / \sum_{i=1}^{t} c_i^2}.$$

Anwendungsbeispiel: Zufällige Graphen

Zufälliger Graph $G_{n,p}$: Graph auf n (markierten) Knoten. Jede der $\binom{n}{2}$ möglichen Kanten wird mit W'keit p in den Graphen eingefügt (unabhängig von einander).

Edge Exposure Martingal

Idee: Graph wird Kante für Kante vorgezeigt ...

Vertex Exposure Martingal

Idee: Graph wird Knoten für Knoten vorgezeigt ...

Anwendung: Chromatische Zahl

$$\Pr[\ |\chi(G_{n,p}) - \mathbb{E}[\chi(G_{n,p})]| \geq \lambda\ ] \ \leq\ 2e^{-\lambda^2/(2n)}.$$

Tools (5)

Target Shooting

Gegeben: Universum U, Teilmenge $S \subseteq U$ (in Form eines *Orakels*, das gegeben ein $u \in U$ wahrheitsgemäß antwortet, ob $u \in S$.)

Gesucht: Schätzer für $\rho := |S|/|U|$.

Vorgehensweise:
Erzeuge $x_1, \ldots, x_n \in U$ zufällig, gleichverteilt und unabhängig voneinander. Setze

$$f(x_i) = \begin{cases} 1 & x_i \in S \\ 0 & \text{sonst} \end{cases}$$

Gebe $\hat{\rho} := \sum_{i=1}^{n} f(x_i)/n$ als Schätzer für ρ aus.

Dann gilt:

$$\Pr\big[(1-\epsilon)\rho \leq \hat{\rho} \leq (1+\epsilon)\rho\big] \leq e^{-n\rho\frac{\epsilon^2}{2}}.$$

Bem: Erzeugt man die Punkte x_i mit einer Markov-Kette, so spricht man von einer Markov Chain Monte Carlo Methode.

Tools (6)

Markov-Ketten

Idee: Konstruiere Markov-Kette so, dass

- gesuchte Verteilung der stationären Verteilung entspricht und
- die Markov-Kette "möglichst schnell" in die stationäre Verteilung konvergiert.

Def: Eine Markov-Kette heißt "*schnell mischend*", falls die stationäre Verteilung in polynomieller Zeit erreicht wird.

Anwendungsbeispiel: Mischen von Karten

Wiederholt man die Operation “schiebe die oberste Karte irgendwo (zufällig) in den Stapel” $\approx n \log n$ mal, so sind die Karten danach gut gemischt.

Anwendungsbeispiel: Lösen von 3Sat

Wdh $(3/2)^n$ mal:

  Wähle Belegung zufällig.

  Wdh. $3n$ mal:

    Wähle eine unerfüllte Klausel. Falls es keine gibt: STOP, erfüllende Belegung gefunden.

    Wähle zufällig eine Variable der Klausel und ändere ihre Belegung.

Satz: Alg. ist Monte Carlo Algorithmus für 3SAT.

4. Kapitel: Verteilte Algorithmen

Byzantisches Problem [Informatik-Variante]

Gegeben: n Prozessoren, von denen max. t defekt sind; jeder Prozessor P_i startet mit Wert b_i.

Gesucht: Protokoll, so dass sich die (nicht defekten) Prozessoren möglichst schnell auf einen Wert einigen, wobei dieser dem Startwert entsprechen soll, falls dieser bei allen identisch ist.

Randomisierter Algorithmus: $\mathbb{E}[\#Runden] \leq 6$

Zum Vergleich: Jedes deterministische Verfahren benötigt mindestens $t + 1$ Runden

Choice Coordination Problem

Gegeben: n Prozessoren und n Speicherstellen; *Gesucht:* Protokoll, so dass am Ende in genau einer der Speicherstellen der Wert ✓ steht.

Schwierigkeit: Brechen von Symmetrien: bei gleicher Konfiguration soll nur *einer* der Prozessoren etwas tun.

Randomisierter Algorithmus: $\mathbb{E}[\#\text{Lese-/Schreib Operationen pro Prozessor}] = \mathcal{O}(1)$

5. Kapitel: Algebraische Methoden

Idee: Statt zwei Instanzen auf Gleichheit zu prüfen, wird zunächst von beiden Instanzen jeweils ein "*Fingerabdruck*" berechnet und dann nur diese Fingerabdrücke verglichen. Bei Gleichheit der Fingerabdrücke müssen die beiden Instanzen allerdings nicht notwendiger Weise ebenfalls gleich sind. Bei einer geschickten Wahl des "*Fingerabdruckes*" kann man aber die Wahrscheinlichkeit solch eines Irrtums klein halten.

Tool: Schwartz-Zippel Theorem

Wir betrachten folgende Beispiele für diese Methode:

1. Matrizenmultiplikation
2. String Matching
3. Matchings in Graphen

Matrizenmultiplikation

Gegeben: $n \times n$ Matrizen A, B, C

Frage: Gilt $AB = C$?

Standardansatz: Berechne $D = AB$ und vergleiche D und C.

Laufzeit: $O(n^3)$ (mit Schulmultiplikation zur Berechnung von D)

Randomisierter Ansatz: Wähle zufälligen Vektor $r \in \{0,1\}^n$ und berechne $x = ABr$ und Cr und vergleiche x und y.

Satz: $$\text{Pr}[\text{“falsche Antwort”}] = \begin{cases} 0 & AB = C \\ \text{Pr}[x = y \mid AB \neq C] \leq 1/2 & AB \neq C \end{cases}.$$

Laufzeit: $O(n^2)$

Schwartz-Zippel Theorem

Satz: Sei $Q(x_1, \ldots, x_n)$ multivariates Polynom, nicht identisch Null, mit maximalem (totalem) Grad d. Dann gilt für jede endliche Menge S:

$$\text{Pr}_{r_1,\ldots,r_n \in S}[Q(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Beweis: Induktion über n.

“$n = 1$”: Klar: Ein Polynom in einer Variablem mit Grad d hat bekanntlich höchstens d Nullstellen.

“$n - 1 \Rightarrow n$”: vgl. Vorlesung 20.12.02

String Matching

Gegeben: Textstring $x_1 x_2 \ldots x_n$ und Muster/Pattern $y_1 y_2 \ldots y_m$

Frage: Kommt das Pattern im Text vor, d.h. gibt es ein i mit

$x_i x_{i+1} \ldots x_{i+m-1} = y_1 y_2 \ldots y_m$?

Deterministische Algorithmen:

naiver Ansatz: $O(nm)$

KMP, Boyer-Moore: $O(n+m)$

Randomisierter Algorithmus:

Idee: Berechne für jede Stelle des Textes einen Fingerabdruck, vergleiche diese mit dem Fingerabdruck des Musters.

<u>Ann:</u> $x_i, y_j \in \{0, 1\}$ [andere Alphabete: einfache Änderung]

String Matching (2)

String Matching: Randomisierter Algorithmus

Wähle zufällige Primzahl p;

$Y = \sum_{j=0}^{m-1} y_{m-j} 2^j$ mod p; //Fingerabdruck Muster

$X = \sum_{j=0}^{m-1} x_{m-j} 2^j$ mod p; //Fingerabdruck Text an Pos. 1

$i = 1$

while $i \leq n - m + 1$ **do**

  if $X = Y$ **then**

    if $x_i x_{i+1} \ldots x_{i+m-1}$ und $y_1 y_2 \ldots y_m$ **then**

      STOP(“Match an Position i”);

    end if

  end if

  $X = 2 \cdot [X - 2^{m-1} x_i] + x_{i+m}$ mod p; //Fingerabdruck Text an Pos. $i + 1$

end do

STOP(“No Match.”);

Satz: Algorithmus hat erwartete Laufzeit $O(n + m)$.

Matchings in bipartiten Graphen

Gegeben: bipartiter Graph $G = (A \cup B, E)$

Frage: Enthält G ein perfektes Matching? Falls ja, bestimme ein solches.

Deterministische Algorithmen:

augmentierende Pfade: $O(nm)$

Edmonds-Karp: $O(\sqrt{n}m)$

Randomisierter Algorithmus:

Idee: Definiere $\hat{M} = \hat{M}(x_{11}, \ldots, x_{nn}) = (\hat{m}_{ij})$, wobei $\hat{m}_{ij} = \begin{cases} x_{ij} & \text{falls } \{a_i, b_j\} \in E \\ 0 & \text{sonst} \end{cases}$

Dann gilt: $\det(\hat{M}) \equiv 0$ genau dann, wenn G *kein* perfektes Matching enthält.

Matchings in bipartiten Graphen (2)

Test, ob Graph perfektes Matching enthält

Wähle $r_{11}, \ldots, r_{nn} \in \{1, \ldots, 2n\}$ zufällig.

if det$\big(\hat{M}(r_{11}, \ldots, r_{nn}\big) \neq 0$ **then**

  return("G enthält perfektes Matching.")

else

  return("G enthält kein perfektes Matching.")

Satz: Laufzeit des Algorithmus ist $O(n^3)$ und es gilt

$$\text{Pr}\big[\text{“falsche Antwort”}\big] \quad \begin{cases} = 0 & \text{falls } G \text{ kein perfektes Matching enthält} \\ \leq 1/2 & \text{sonst} \end{cases}$$

Bem.: Wiederholt man den Algorithmus k mal, und gibt "G enthält perfektes Matching" aus, falls mindestens einer der k Aufrufe zu diesem Ergebnis kam, so reduziert sich die Fehlerwahrscheinlichkeit auf $1/2^k$.

Matchings in bipartiten Graphen (3)

Bestimmung eines perfekten Matching

Eingabe: bipartiter Graph $G = (A \cup B, E)$, der ein perfektes Matching enthält

Ausgabe: perfektes Matching $M \subseteq E$.

$H := G$; $M := \emptyset$;

while $|E(H)| > \frac{|V(H)|}{2}$ **do**

  Wähle $e = \{u, v\} \in E(H)$ beliebig.

  if $H - e$ enthält (noch immer) ein perfektes Matching **then**

    $H := H - e$;   // lösche Kante e

  else

    $M := M \cup e$;

    $H := H[V(H) \setminus \{u, v\}]$;   // lösche Knoten u, v und alle inzidenten Kanten

  end if

end do

$M := M \cup E(H)$;

Satz: Algorithmus hat erwartete Laufzeit $O(mn^3 \log n)$.

Matchings in gewichteten Graphen

Gegeben: Graph $G = (V, E)$, $w : E \to \mathbb{Z}$

Frage: Enthält G ein perfektes Matching? Falls ja, bestimme eines mit minimalem Gewicht.

D.h., bestimme ein M mit

$$w(M) = \min\{w(M') \mid M' \text{ perfektes Matching}\}, \qquad \text{wobei } w(X) = \sum_{e \in X} w(e).$$

Deterministische Algorithmen:

Blossom-Algorithmus: $O(n^3)$; Alg. von Micali, Vazirani: $O(\sqrt{n}m)$

Randomisierter Algorithmus:

Idee: Definiere $\hat{M} = \hat{M}(x_{12}, \ldots, x_{1n}, x_{23}, \ldots, x_{2n}, \ldots, x_{n-1\,n}) = (\hat{m}_{ij})$, wobei

$$\hat{m}_{ij} = \begin{cases} x_{ij} & \text{falls } \{v_i, v_j\} \in E \wedge i < j \\ -x_{ji} & \text{falls } \{v_i, v_j\} \in E \wedge i > j \\ 0 & \text{sonst} \end{cases}$$

Satz von Tutte: $\det(\hat{M}) \equiv 0$ genau dann, wenn G *kein* perfektes Matching enthält.

Erweiterung des Algorithmus für gewichtete Graphen

Idee: Setze $x_{ij} := 2^{w_{ij}}$, wobei $w_{ij} = w(\{v_i, v_j\})$ das Gewicht der Kante $\{v_i, v_j\}$ sei. Also

$$A = (a_{ij}), \qquad \text{wobei } a_{ij} = \begin{cases} 2^{w_{ij}} & \text{falls } \{v_i, v_j\} \in E \wedge i < j \\ -2^{w_{ji}} & \text{falls } \{v_i, v_j\} \in E \wedge i > j \\ 0 & \text{sonst} \end{cases}$$

Folgerung aus dem Satz von Tutte: Sei G ein Graph und M ein perfektes Matching mit minimalem Gewicht. Dann gilt: Ist M_0 das *einzige* perfekte Matching in G mit Gewicht $w(M_0)$, so gilt:

$$w_0 = \max_{4^k \mid \det(A)} k \, .$$

Bestimmung eines perfekten Matching mit minimalem Gewicht

Eingabe: Graph $G = (V, E; w)$ mit einem *eindeutigen* minimalem perfekten Matching

Ausgabe: perfektes Matching mit minimalem Gewicht.

Bestimme A wie oben; $k_0 := \max\{k \mid 4^k$ teilt $\det(A)\}$;

$M := \emptyset$;

for all $e \in E$ **do**

// Bestimme die zum Graphen $G - e$ gehörende Matrix A':

Sei $e = \{v_k, v_l\}$; Setze $A' = (a'_{ij})$, wobei $a'_{ij} = \begin{cases} 0 & \text{falls } \{i,j\} = \{k,l\} \\ a_{ij} & \text{sonst} \end{cases}$

$k' := \max\{k \mid 4^k$ teilt $\det(A')\}$;

if $k_0 < k'$ **then** $M := M \cup \{e\}$;

end do

return(M).

Satz: Unter den oben genannten Voraussetzungen (Eindeutigkeit des minimalen perfekten Matching!) ist der Algorithmus korrekt und benötigt $O(mn^3)$ arithmetische Operationen, **aber ...**