Randomisierte Algorithmen

11. Randomisiertes Approximieren

Thomas Worsch

Fakultät für Informatik
Karlsruher Institut für Technologie

Wintersemester 2017/2018

Überblick

Polynomielle Approximationsschemata

Zählen von Lösungen von Formeln in DNF

Überblick

Polynomielle Approximationsschemata

Zählen von Lösungen von Formeln in DNF

Zählprobleme

- im folgenden: Π ein *Zählproblem*, das für jede Eingabe x eine gewisse Anzahl $\#_\Pi(x)$ (oder kurz $\#(x)$) von „Lösungen“ besitzt.
- Zu Π gehöriges Entscheidungsproblem E_Π:
  Gibt es für eine Eingabe x eine Lösung, d. h. ist $\#(x) > 0$?

Komplexitätsklasse #P

- Problem Π gehört zu **#P**, wenn
  - es eine nichtdeterministische Turingmaschine gibt,
  - die in Polynomialzeit arbeitet und
  - für jede Eingabe x genau $\#_\Pi(x)$ akzeptierende Berechnungen besitzt.
- Problem Π ist *#P-vollständig*, wenn
  - es in **#P** ist und
  - jedes Problem $\Pi' \in$ **#P**
  - von einer deterministischen Turingmaschine
  - in Polynomialzeit
  - auf Π reduziert werden kann
    - in dem ein oder anderen hier nicht näher spezifizierten Sinne

Lemma

Wenn man ein #**P**-vollständiges Problem deterministisch in Polynomialzeit lösen kann, dann ist **P** = **NP**.

Deswegen sind schnelle Algorithmen interessant, die zumindest (gute) Näherungslösungen finden …

#**P**-vollständige Probleme

Die folgenden Probleme sind #**P**-vollständig:

- Wieviele erfüllende Belegungen hat eine DNF-Formel?
- Wieviele erfüllende Belegungen hat eine 2SAT-Formel?

- Wieviele perfekte Matchings hat ein bipartiter Graph?
- Was ist die Permanente einer Booleschen Matrix?
- Wieviele topologische Sortierungen eines DAG gibt es?

Approximationsschemata (deterministisch)

- *Approximationsschema (AS)* für Π :
  - deterministischer Algorithmus A, der
  - für jede Eingabe x der Größe $n = |x|$ und
  - für jedes $\varepsilon > 0$
  - eine Ausgabe $A(x)$ erzeugt, für die gilt:

$$(1 - \varepsilon)\#(x) \leq A(x) \leq (1 + \varepsilon)\#(x)$$

- $A(x)$ heißt dann eine *ε-Approximation* von $\#(x)$.
- *polynomielles Approximationsschema (PAS)*:
  ein AS, dessen Laufzeit polynomiell in n ist.
- *voll polynomielles Approximationsschema (FPAS)*:
  ist ein AS, dessen Laufzeit polynomiell in n und $1/\varepsilon$ ist.

Anmerkung

- Für #**P**-vollständige Probleme kennt man keine PAS oder gar FPAS.
- also Approximationsschemata mit Zufallsentscheidungen betrachten …

Randomisierte Approximationsschemata

- *randomisiertes Approximationsschema (RAS)* für Π:
  - ein randomisierter Algorithmus A, der
  - für jede Eingabe x der Größe $n = |x|$ und
  - für jedes $\varepsilon > 0$
  - eine Ausgabe $A(x)$ erzeugt, für die gilt:

$$\mathbf{P}((1-\varepsilon)\#(x) \leq A(x) \leq (1+\varepsilon)\#(x)) \geq \frac{3}{4}$$

- *polynomielles randomisiertes Approximationsschema (PRAS)*:
  ein RAS, dessen Laufzeit polynomiell in n ist.
- *voll polynomielles randomisiertes Approximationsschema (FPRAS)*:
  ein RAS, dessen Laufzeit polynomiell in n und $1/\varepsilon$ ist.

Anmerkungen

- Wahl der Wahrscheinlichkeit $3/4$ nicht wesentlich
- Jede Konstante echt größer $1/2$ ist „genauso gut“.

Anmerkungen

- Wahl der Wahrscheinlichkeit $3/4$ nicht wesentlich
- Jede Konstante echt größer $1/2$ ist „genauso gut“.

- aus PRAS für Π kann man randomisierten Algorithmus für E_Π konstruieren, so dass man sieht: $E_\Pi \in \mathbf{BPP}$.

Anmerkungen

- Wahl der Wahrscheinlichkeit $3/4$ nicht wesentlich
- Jede Konstante echt größer $1/2$ ist „genauso gut“.

- aus PRAS für Π kann man randomisierten Algorithmus für E_Π konstruieren, so dass man sieht: $E_\Pi \in \mathbf{BPP}$.

- Wenn es ein PRAS für die „Zählvariante“ eines **NP**-vollständigen Problems gibt, so wäre $\mathbf{NP} \subseteq \mathbf{BPP}$.
- Experten erwarten das nicht …

noch eine Definition

(ε, δ)-FPRAS:

- ein FPRAS, das für jede Eingabe x
- mit einer Wahrscheinlichkeit größer oder gleich $1 - \delta$
- eine ε-Approximation berechnet und
- dessen Laufzeit polynomiell in n, $1/\varepsilon$ und $\log 1/\delta$ ist.

Überblick

Problemstellung

- DNF-Formel $F(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_m$
  - mit $C_1 = \ell_{1,1} \wedge \cdots \wedge \ell_{1,r_1}, \ldots, C_m = \ell_{m,1} \wedge \cdots \wedge \ell_{m,r_m}$
  - jedes Literal $\ell_{i,j}$ ist Variable x_k oder ihre Negation $\overline{x_k}$
  - jede Variable kommt in jedem C_i höchstens einmal vor
- Es sei $n \geq 1$ und $m \geq 1$.
- sei $\#F$ die Anzahl der F erfüllenden Variablenbelegungen $\mathbf{x} : \{x_1, \ldots, x_n\} \to \{0, 1\}$.
- Aufgabe: Man bestimme zu gegebenem F die Zahl $\#F$.

$\#F$ zu berechnen ist schwer

- Das Problem ist #**P**-vollständig.
- Existenz eines deterministischen Polynomialzeit-Algorithmus wird (analog wie bei **NP**) von vielen „bezweifelt“.

- werden im folgenden aber ein (ε, δ)-FPRAS sehen und analysieren.

- zunächst für allgemeinere Problemstellung ein erster einfacher Algorithmus

Aufgabenstellung

- gegeben:
  - U endliches Universum
  - Teilmenge $G \subseteq U$ gegeben mittels charakteristischer Funktion $g : U \to \{0, 1\}$
- gesucht:
  - Größe $|G|$
  - gelegentlich $\rho = |G| / |U|$
- Annahmen:
  - Man kann schnell zufällig gleichverteilt ein u aus U auswählen.
  - g ist schnell zu berechnen, d. h. Frage „$u \in G$?“ schnell zu entscheiden.

Berechnung von $\#F$

- U: Menge aller 2^n Variablenbelegungen $\mathbf{x}$
- G: Menge der erfüllenden Variablenbelegungen $\mathbf{x}$
- g: Auswertung $F(\mathbf{x})$

Ein naiver Algorithmus

Wie könnte man Randomisierung einsetzen?

Ein naiver Algorithmus

Wie könnte man Randomisierung einsetzen?
z. B. so:

$z \leftarrow 0$ ⟨*Variable zum Zählen der Lösungen*⟩
for $i \leftarrow 1$ **to** N **do**
    $u \leftarrow$ ⟨*zufällig gleichverteilt aus* U *gewählt*⟩
    if $g(u) = 1$ **then**
        $z \leftarrow z + 1$
    fi
od

Ein naiver Algorithmus

Wie könnte man Randomisierung einsetzen?
z. B. so:

$z \leftarrow 0$ ⟨*Variable zum Zählen der Lösungen*⟩
for $i \leftarrow 1$ **to** N **do**
    $u \leftarrow$ ⟨*zufällig gleichverteilt aus* U *gewählt*⟩
    if $g(u) = 1$ **then**
        $z \leftarrow z + 1$
    fi
od
return $\frac{z}{N} \cdot |U|$

Analyse des naiven Algorithmus

Lemma

- sei Y_i die Zufallsvariable mit

$$Y_i = \begin{cases} 1 & \text{falls im } i\text{-ten Schleifendurchlauf } g(u) = 1 \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

- sei $Z = |U| \frac{\sum_{i=1}^{N} Y_i}{N}$
- Dann ist …?…

Analyse des naiven Algorithmus

Lemma

- sei Y_i die Zufallsvariable mit

$$Y_i = \begin{cases} 1 & \text{falls im } i\text{-ten Schleifendurchlauf } g(u) = 1 \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

- sei $Z = |U| \frac{\sum_{i=1}^{N} Y_i}{N}$
- Dann ist $\mathbf{E}[Z] = |G|$.

Analyse des naiven Algorithmus (2)

Beweis

- offensichtlich $\mathbf{E}[Y_i] = |G| / |U|$
- wegen Linearität des Erwartungswertes:

$$\mathbf{E}[Z] = \frac{|U|}{N} \sum_{i=1}^{N} \mathbf{E}[Y_i] = \frac{|U|}{N} \cdot N \cdot \frac{|G|}{|U|} = |G|$$

Analyse des naiven Algorithmus (3)

- Frage: Wie groß sollte man die Zahl N der Schleifendurchläufe wählen, damit die Wahrscheinlichkeit für ein „stark“ von $|G|$ abweichendes Ergebnis „klein“ ist.
- zeigen: eine hinreichende Bedingung
- Mitteilung: sie ist auch „fast notwendig“

Analyse des naiven Algorithmus (4)

Satz

- $\rho = |G| / |U|$
- Seien δ und ε aus dem Intervall $(0; 1]$.
- Der naive Algorithmus liefert mit Wahrscheinlichkeit $\geq 1 - \delta$ eine ε-Approximation für $|G|$, falls gilt:

$$N \geq \frac{5}{\varepsilon^2 \rho} \ln \frac{2}{\delta} \ .$$

Beweis (1)

- seien δ und ε beliebig aber fest
- Y_i wieder die Zufallsvariable mit

$$Y_i = \begin{cases} 1 & \text{falls im } i\text{-ten Schleifendurchlauf } g(u) = 1 \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

- sei $Y = \sum_{i=1}^{N} Y_i$ und folglich $Z = |U|\, Y/N$.
- Y ist binomialverteilt mit Erwartungswert $N\rho$ und
- Ergebnisse über Chernoff-Schranken anwendbar
- man erhält …

Beweis (2)

- man erhält:

$$
\begin{aligned}
& \mathbf{P}((1-\varepsilon)\,|G| \leq Z \leq (1+\varepsilon)\,|G|) \\
= \quad & \mathbf{P}((1-\varepsilon)N\rho \leq Y \leq (1+\varepsilon)N\rho) \\
= \quad & 1 - \mathbf{P}((1-\varepsilon)N\rho > Y) - \mathbf{P}(Y > (1+\varepsilon)N\rho) \\
\geq \quad & 1 - e^{-\varepsilon^2 N\rho/2} - e^{-\varepsilon^2 N\rho/5} \\
\geq \quad & 1 - 2e^{-\varepsilon^2 N\rho/5}\,.
\end{aligned}
$$

Beweis (3)

- also

$$\mathbf{P}((1-\varepsilon)\,|G| \leq Z \leq (1+\varepsilon)\,|G|) \geq 1 - 2e^{-\varepsilon^2 N\rho/5}\ .$$

- Einsetzen von $N = \frac{5}{\varepsilon^2\rho}\ln\frac{2}{\delta}$ ergibt im Exponenten

$$-\frac{\varepsilon^2 N\rho}{5} = -\frac{\varepsilon^2 5\rho\ln\frac{2}{\delta}}{\varepsilon^2\rho 5} = -\ln\frac{2}{\delta} = \ln\frac{\delta}{2}$$

- also $\mathbf{P}((1-\varepsilon)\,|G| \leq Z \leq (1+\varepsilon)\,|G|) \geq 1-\delta$.

N hinreichend — leider nicht nur das

- Satz:
  - Wenn man (unter anderem) N umgekehrt proportional zu $\rho = |G| / |U|$ wählt,
  - dann erhält man eine „gute“ Approximation.

N hinreichend — leider nicht nur das

- Satz:
  - Wenn man (unter anderem) N umgekehrt proportional zu $\rho = |G| / |U|$ wählt,
  - dann erhält man eine „gute“ Approximation.
- Aber ρ kann 2^{-n} sein: exponentielle Laufzeit

N hinreichend — leider nicht nur das

- Satz:
  - Wenn man (unter anderem) N umgekehrt proportional zu $\rho = |G| / |U|$ wählt,
  - dann erhält man eine „gute“ Approximation.
- Aber ρ kann 2^{-n} sein: exponentielle Laufzeit
- „Leider“ sind Chernoff-Schranken recht gut:
  - bei obigem einfachen Algorithmus: große Laufzeiten nicht nur hinreichend, sondern auch notwendig.
- *polynomielles* RAS erfordert anderes Vorgehen.
- Problem: G sehr klein im Vergleich zu U

N hinreichend — leider nicht nur das

- Satz:
  - Wenn man (unter anderem) N umgekehrt proportional zu $\rho = |G| / |U|$ wählt,
  - dann erhält man eine „gute“ Approximation.
- Aber ρ kann 2^{-n} sein: exponentielle Laufzeit
- „Leider“ sind Chernoff-Schranken recht gut:
  - bei obigem einfachen Algorithmus: große Laufzeiten nicht nur hinreichend, sondern auch notwendig.
- *polynomielles* RAS erfordert anderes Vorgehen.
- Problem: G sehr klein im Vergleich zu U
- Ziel: Mache ggf. U kleiner …
  - DNF-Problem: betrachte nicht mehr *alle* Variablenbelegungen sondern nur manche.

Neue Aufgabenstellung

- sei V ein endliches Universum mit
- m Teilmengen $H_1, \ldots, H_m \subseteq V$, so dass
  - jedes $|H_i|$ in Polynomialzeit berechenbar
  - aus jedem H_i zufällig gleichverteilt ein Element auswählbar
  - Für alle $v \in V$ und alle i kann man in Polynomialzeit feststellen, ob $v \in H_i$ ist oder nicht.
- Aufgabe: Bestimme die Größe von $H = H_1 \cup \cdots \cup H_m$.

Neue Aufgabenstellung

- sei V ein endliches Universum mit
- m Teilmengen $H_1, \ldots, H_m \subseteq V$, so dass
  - jedes $|H_i|$ in Polynomialzeit berechenbar
  - aus jedem H_i zufällig gleichverteilt ein Element auswählbar
  - Für alle $v \in V$ und alle i kann man in Polynomialzeit feststellen, ob $v \in H_i$ ist oder nicht.
- Aufgabe: Bestimme die Größe von $H = H_1 \cup \cdots \cup H_m$.

- DNF-Problem:
  - H_i: Variablenbelegungen, die Klausel C_i erfüllen

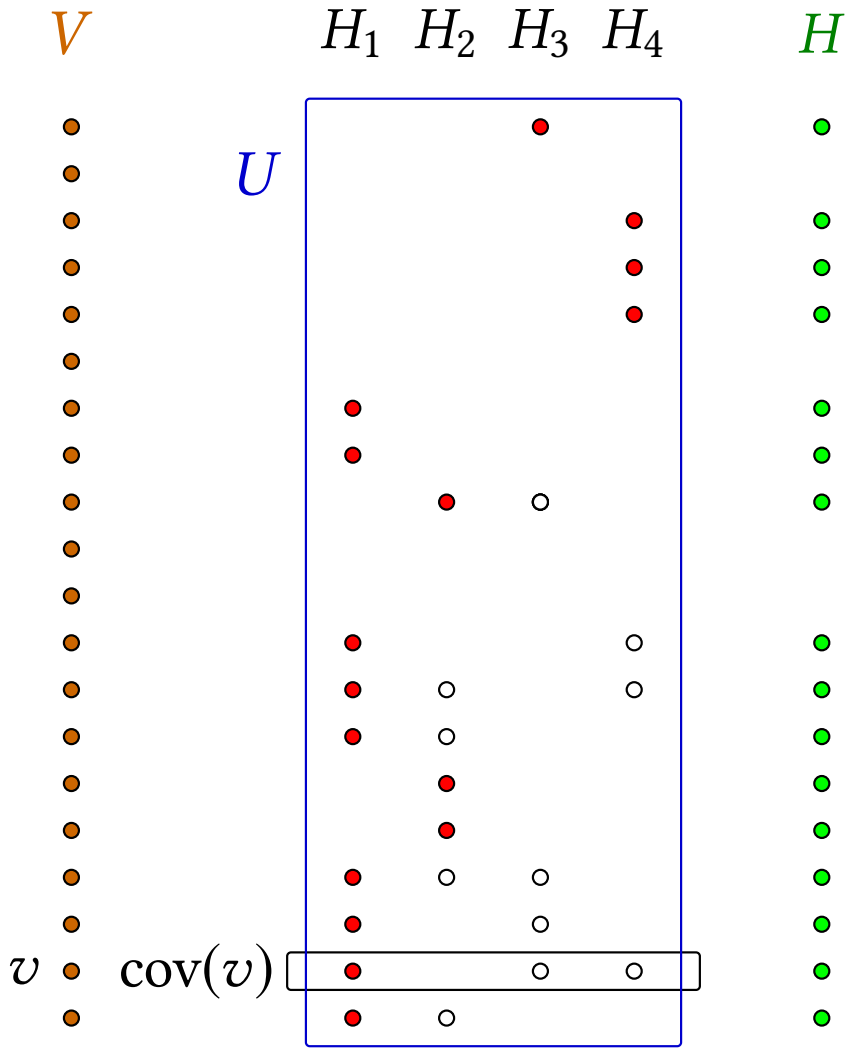
Der wesentliche Trick

- sei $U = \{(v, i) \mid v \in H_i\}$, also $U = H_1 \cup \cdots \cup H_m$
- $|U| \geq |H|$
- sei $\text{cov}(v) = \{(v, i) \mid (v, i) \in U\}$.
- dann:
  - Es gibt genau $|H|$ nichtleere Mengen $\text{cov}(v)$.
  - Die $\text{cov}(v)$ sind paarweise disjunkt, partitionieren U.
  - $|U| = \sum |\text{cov}(v)|$.
  - Für alle $v \in V$ ist $|\text{cov}(v)| \leq m$.
- sei

$$G = \left\{ (v, i) \,\middle|\, v \in H \wedge i = \min\{j \mid v \in H_j\} \right\} \subseteq U$$

- $|G| = |H|$
  - in G wird jedes $v \in H$ genau einmal gezählt
  - nämlich für das kleinste i mit $v \in H_i$.

Der wesentliche Trick im Bild



Der wesentliche Trick

- Statt die Größe von $H \subseteq V$ zu bestimmen,
- kann man die Größe von $G \subseteq U$ berechnen.
- im zweiten Fall sind die Größenverhältnisse „günstiger“:

Lemma

$$\rho = |G| / |U| \geq 1/m$$

Beweis

$$|U| = \sum_{v \in H} |\mathrm{cov}(v)| \leq \sum_{v \in H} m = m\,|H| = m\,|G|$$

Algorithmus

$\langle$*Formel* $F = C_1 \vee \cdots \vee C_m$ *mit Klauseln* $C_i = \ell_{i,1} \wedge \cdots \wedge \ell_{i,r_i}\rangle$
$sizeU \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
    $sizeHi \leftarrow 2^{n-r_i}$
    $sizeU \leftarrow sizeU + sizeHi$
od
$N \leftarrow \frac{5m}{\varepsilon^2} \ln \frac{2}{\delta}$; $z \leftarrow 0$
for $j \leftarrow 1$ **to** N **do**
    $i \leftarrow \langle$*zufällig aus* $[1;m]$ *mit W.keit* $sizeHi/sizeU$ *gewählt*$\rangle$
    $x \leftarrow \langle C_i$ *erfüllende Var.bel.; zufällig gleichverteilt*$\rangle$
    if $\neg\exists j < i : x \in H_j$ **then** $\quad\langle x \in G\ ?\rangle$
        $z \leftarrow z + 1$
    fi
od
return $z \cdot sizeU/N$

Theorem

Der Algorithmus von eben ist ein (ε, δ)-FPRAS für das DNF-Problem.

Beweis (1)

- Größe der Eingabe liegt in $\Omega(n+m)$ und in $O(nm)$.
- nach Lemma ist $\rho \geq 1/m$
- H_i: Menge aller Variablenbelegungen, die C_i erfüllen.
- $|H_i| = 2^{n-r_i}$
- noch zu zeigen: In der zweiten Schleife wird durch die Zuweisungen an i und x stets zufällig ein Element aus U *gleichverteilt* ausgewählt,
  - Wahrscheinlichkeit, ein bestimmtes (x, i) auszuwählen: $|H_i| / \sum |H_i| \cdot 1/|H_i| = 1/\sum |H_i| = 1/|U|$.
- Also ist der zweite Algorithmus tatsächlich ein Spezialfall des ersten Algorithmus.

Beweis (2)

- zweiter Algorithmus Spezialfall des ersten Algorithmus
- Also genügt eine Anzahl Schleifendurchläufe von

$$N = \frac{5m}{\varepsilon^2} \ln \frac{2}{\delta}$$

- Anzahl Schleifendurchläufe und Gesamtlaufzeit polynomial in $1/\varepsilon$, $\log 1/\delta$ und der Problemgröße