Approximations-Algorithmen

Jan Johannsen

Vorlesung im Wintersemester 2013/14

Einordnung

Algorithmik

- Entwurf und
- Analyse von Algorithmen

Komplexitätstheorie

- Analyse der Komplexität von Problemen
- Einteilung in Klassen ähnlicher Komplexität
- Untersuchung der Struktur der Klassen

Analyse von Algorithmen

Analyse des Verbrauchs an Ressourcen

- Zeit
- Speicher
- ...

Abhängig von der Größe des Input

- Komplexität im *worst case*

Asymptotische Analyse – vernachlässigt

- konstante Faktoren
- endliche Anfangsstücke

Komplexität von Problemen

Obere Schranken

- durch Angabe eines Algorithmus
- und dessen Analyse

Untere Schranken

- durch Analyse des Problems
- Beweis, dass jeder effizientere Algorithmus versagt

Beispiel: Sortieren

- HeapSort, MergeSort brauchen $O(n \log n)$ Vergleiche
- untere Schranke: $\Omega(n \log n)$ Vergleiche
- $\rightsquigarrow$ Komplexität: $\Theta(n \log n)$

Problemtypen

Problem: Erreichbarkeit

Instanz: Graph G, Knoten s, t.
Lösung: Weg von s nach t.

Entscheidung: Gibt es einen Weg von s nach t ?

Suche: Finde einen Weg von s nach t.

Optimierung: Finde einen *kürzesten* Weg von s nach t.

Zählen: Bestimme die *Anzahl* der Wege von s nach t.

Optimierungsprobleme

Eigenschaften:

- mehrere potentielle Lösungen zu jedem Input
- mit Kosten- / Nutzenmaß versehen
- gesucht: Lösung mit minimalen Kosten / maximalem Nutzen

Beispiele:

- kürzeste Wege
- Travelling Salesperson
- Scheduling
- ...

Approximation

Viele Optimierungsprobleme sind NP-schwer

- effizienter Algorithmus nur, wenn $\mathsf{P} = \mathsf{NP}$
- nur exponentielle Algorithmen zur optimalen Lösung

Deshalb: Approximations-Algorithmen

- effizient
- liefern eine nicht notwendig optimale Lösung
- Abweichung vom Optimum a priori abzuschätzen

Approximierbarkeit

NP-schwere Optimierungsprobleme unterscheiden sich bzgl. ihrer Approximierbarkeit:

- es gibt beliebig gute Approximations-Algorithmen
- es gibt Approximations-Algorithmen einer gewissen Güte, aber keine besseren.
- es gibt überhaupt keine guten Approximations-Algorithmen

Ziele:

- Methoden zum Entwurf
- Grundlagen zur Untersuchung dieser Unterschiede

Übersicht

Übersicht

Komplexitätsklassen

Entscheidungsproblem ist gegeben durch:

- Menge von Instanzen I
- Teilmenge der positiven Instanzen $Y \subseteq I$

Beispiel: SAT

- I_{SAT} = Formeln in KNF
- Y_{SAT} = erfüllbare Formeln

Formel in KNF	$F = C_1 \wedge \ldots \wedge C_m$
Klausel	$C = a_1 \vee \ldots \vee a_k$
Literal	$a = x$ oder $a = \neg x$.

Komplexitätsklassen

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

NP als Verifikation von Lösungen

Problem P ist in NP gdw.

- es gibt
  - Menge W_P
  - Problem $Q \in \mathrm{P}$
  - Polynom $q()$
- I_Q sind Paare (x, w) mit
  - $x \in I_P, \quad w \in W_P$
  - $|w| \leq q(|x|)$
- $\forall x \in I_P : \quad x \in Y_P \leftrightarrow \exists w \in W_P : (x, w) \in Y_Q$

Beispiel: *SAT*

- W_{SAT} = Bewertung der Variablen
- $(F, \alpha) \in Y_Q$ gdw. α erfüllt F

Reduktion

Funktion f reduziert P auf Q, wenn

- $f : I_P \longrightarrow I_Q$ in polynomieller Zeit berechenbar
- für alle $x \in I_P$ ist $x \in Y_P \leftrightarrow f(x) \in Y_Q$

$P \leq_m Q$ falls es eine Reduktion f von P auf Q gibt.

Eigenschaft:

Ist $Q \in \mathrm{P}$ und $P \leq_m Q$, dann auch $P \in \mathrm{P}$.

Vollständigkeit

Definition:

- Problem P ist NP-schwer, falls $Q \leq_m P$ für alle $Q \in$ NP.
- P ist NP-vollständig, falls $Q \in$ NP und NP-schwer.

Q NP-vollständig $\Rightarrow$ $Q \in$ P gdw. P = NP

Satz (Cook)

SAT ist NP-vollständig.

Problem 3-*SAT*: Jede Klausel $C = a_1 \vee a_2 \vee a_3$

Satz (Cook)

3-SAT ist NP-vollständig.

Lineare Programmierung

Problem *ILP* (Integer Linear Programming)

Instanz:	Menge I von linearen Ungleichungen über $\mathbb{Z}$ in Variablen $z_1, \ldots, z_n$
Frage:	Gibt es eine Lösung von I in $\mathbb{Z}$ d.h. Zuweisung von Werten $z_1, \ldots, z_n \in \mathbb{Z}$ so dass Ungleichungen I gelten.

Satz

ILP ist NP-vollständig.

Problem *LP*:

Wie ILP, aber gesucht sind Lösungen in $\mathbb{Q}$.

Im Gegensatz zum obigen Satz ist *LP* in P.

Graphenprobleme in NP

Problem *VC*

Instanz:	Graph $G = (V, E)$, $k \in \mathbb{N}$
Frage:	Gibt es vertex cover U mit $\|U\| \leq k$?

Vertex cover: $U \subseteq V$ mit $e \cap U \neq \emptyset$ für alle $e \in E$.

Problem $\textsc{Clique}$

Instanz:	Graph $G = (V, E)$, $k \in \mathbb{N}$
Frage:	Gibt es Clique U mit $\|U\| \geq k$?

Clique: $U \subseteq V$ mit $\{x, y\} \in E$ für alle $x, y \in U$.

Problem *HC*

Instanz:	Graph $G = (V, E)$
Frage:	Gibt es einen Hamilton-Kreis in G ?

Hamilton-Kreis: Kreis, der jedes $v \in V$ genau einmal besucht.

NP-vollständige Graphenprobleme

Offensichtlich: VC, $\textsc{Clique}$, HC sind in NP.

Reduktionen (Karp)

- $3\text{-}SAT \leq_m VC$
- $VC \leq_m \textsc{Clique}$
- $VC \leq_m HC$

Also: VC, $\textsc{Clique}$, HC sind NP-vollständig.

Optimierungsprobleme

Ein *Optimierungsproblem* ist gegeben durch:

- Menge I von Instanzen.
- Für jedes $x \in I$ eine Menge $S(x)$ von potentiellen Lösungen.
- Funktion m, die jedem (x, y) mit $x \in I$ und $y \in S(x)$ ein Maß $m(x, y)$ zuordnet.
- Ziel $\mathrm{goal} \in \{\min, \max\}$.

Eine Lösung $y^* \in S(x)$ heißt optimal, wenn

$$m(x, y^*) = m^*(x) := \mathrm{goal}\{\, m(x, y)\ ;\ y \in S(x)\,\}$$

$$S^*(x) := \{\, y^* \in S(x)\ ;\ m(x, y^*) = m^*(x)\,\}$$

Beispiele

Minimum Path

Instanz:	ungerichteter Graph $G = (V, E)$, Knoten s, t
Lösung:	Weg von s nach t
Maß:	Länge des Weges

Minimum Vertex Cover

Instanz:	ungerichteter Graph $G = (V, E)$
Lösung:	vertex cover $U \subseteq V$
Maß:	$\lvert U \rvert$

Maximum Clique

Instanz:	ungerichteter Graph $G = (V, E)$
Lösung:	Clique $U \subseteq V$
Maß:	$\lvert U \rvert$

Problemvarianten

Zu einem Optimierungsproblem P sind definiert:

konstruktives Problem	P_C:	Gegeben $x \in I$, finde $y \in S^*(x)$.
Auswertungsproblem	P_E:	Gegeben $x \in I$, berechne $m^*(x)$.
Entscheidungsproblem	P_D:	Gegeben $x \in I$ und $k \in \mathbb{N}$, ist $\mathrm{goal}(m^*(x), k) = m^*(x)$?

Beispiele:

$P = \textsc{Minimum Vertex Cover}$ $\qquad P_D = \mathrm{VC}.$
$P = \textsc{Maximum Clique}$ $\qquad P_D = \textsc{Clique}$

Die Klassen NPO und PO

Ein Optimierungsproblem ist in der Klasse NPO, falls

1. I ist in P.
2. für $x \in I$ und $y \in S(x)$ ist $|y| \leq q(|x|)$
3. für x, y mit $|y| \leq q(|x|)$ ist die Frage "$y \in S(x)$?" in P
4. m ist in polynomialer Zeit berechenbar.

Eigenschaft: P in NPO $\Longrightarrow$ P_D in NP

$P \in$ NPO ist in der Klasse PO, falls
das konstruktive Problem P_C in polynomialer Zeit lösbar ist.

Analog: P in PO $\Longrightarrow$ P_D in P

Turing-Reduktion

Eine Turing-Reduktion von P auf Q ist

- Algorithmus, der P löst
- benutzt Subroutine zur Lösung von Q
- Laufzeit ohne Subroutine ist polynomiell

$P \leq_T Q$ falls es eine Turing-Reduktion von P auf Q gibt.

Eigenschaft:

Ist $Q \in \mathrm{P}$ und $P \leq_T Q$, dann auch $P \in \mathrm{P}$.

Relationen zwischen Problemvarianten

Offensichtlich: $P_D \leq_T P_E \leq_T P_C$

Fakt

Für alle Optimierungsprobleme ist $P_E \leq_T P_D$.

Beweis:

Da $|m(x,y)| \leq p(|x|)$ für alle $y \in S(x)$, ist $m^*(x) < 2^{p(|x|)}$

$\rightsquigarrow$ binäre Suche findet $m^*(x)$ in Zeit $O(p(|x|))$.

Relationen zwischen Problemvarianten

Beispiel

Für $P = \textsc{Maximum Clique}$ ist $P_C \leq_T P_E$.

Für $G = (V, E)$ definiere

- $N(v) := \{ u \in V ; \{u, v\} \in E \}$
- $G(v)$: induzierter Teilgraph auf $\{v\} \cup N(v)$
- $G^-(v)$: induzierter Teilgraph auf $N(v)$

Algorithmus:

$MaxClique(G)$
  $k := \textsc{Maximum Clique}_E(G)$
  `if` $k = 1$ `then return any` $v \in V$
  `finde` $v \in V$ `mit` $\textsc{Maximum Clique}_E(G(v)) = k$
  `return` $\{v\} \cup MaxClique(G^-(v))$

Relationen zwischen Problemvarianten

Ob $P_C \leq_T P_D$ allgemein gilt, ist unbekannt.

Satz

$P_C \leq_T P_D$ gilt, falls P_D NP-vollständig ist.

NP-schwere Optimierungsprobleme

Definition

Problem P in NPO ist NP-schwer,
falls $Q \leq_T P_C$ für alle $Q \in$ NP.

Eigenschaft:

$$P_D \text{ NP-vollständig} \quad \Longrightarrow \quad P \text{ NP-schwer.}$$

$$\mathrm{P} \neq \mathrm{NP} \quad \Longrightarrow \quad \mathrm{PO} \neq \mathrm{NPO}$$

Ein NP-schweres Problem

Minimum Traveling Salesperson

Instanz:	n Städte $c_1, \ldots, c_n$, Distanzmatrix $D \in \mathbb{N}^{n \times n}$
Lösung:	Tour: Permutation $c_{i_1}, \ldots, c_{i_n}$ der Städte
Maß:	Kosten $\sum_{k=1}^{n-1} D(i_k, i_{k+1}) + D(i_n, i_1)$

Satz

Minimum Traveling Salesperson *ist NP-schwer.*