Approximationsalgorithmen

Wintersemester 2013/14

HERZLICH WILLKOMMEN!

Worum geht's?

Eine Bemerkung von Vasek Chvatal

„In den kommunistischen Ländern des Ostblocks in den 60'er und 70'er Jahren war es möglich, intelligent, ehrenhaft oder ein Mitglied der kommunistischen Partei zu sein, aber es war nicht möglich alle drei Eigenschaften gleichzeitig zu verkörpern.“

Wir verlangen von Algorithmen für NP-harte Optimierungsprobleme, dass sie

1. optimale Lösungen bestimmen,
2. in polynomieller Zeit rechnen
3. und dies für jede Instanz tun.

Wir müssen approximieren!

- Welche Optimierungsprobleme können **exakt** durch effiziente Algorithmen gelöst werden?

  Die lineare Programmierung ist ein mächtiges, effzient lösbares Optimierungsproblem.

- Welche erfolgreichen Methoden gibt es im Entwurf von **Approximationsalgorithmen**?
  - Greedy Algorithmen,
  - Dynamische Programmierung,
  - Lokale Suche,
  - Local Ratio,
  - Lineare Programmierung,
  - Branch & Bound und Branch & Cut.
- Lassen schwierige Optimierungsprobleme überhaupt beliebig gute Approxitionen zu?

LINEARE PROGRAMMIERUNG

- Bedingungen werden durch ein System von linearen Ungleichungen definiert.
- Eine optimale Lösung muss alle Bedingungen erfüllen und eine lineare Zielfunktion optimieren.

Formal:

- Eine Instanz wird durch die Matrix $A = (a_{i,j})_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m}$, die „rechte Seite“ $b \in \mathbb{R}^m$ und den Vektor $c \in \mathbb{R}^n$ beschrieben.
- Maximiere $\sum_{i=1}^{n} c_i \cdot x_i$, so dass

$$\sum_{j=1}^{n} a_{i,j} \cdot x_j \leqslant b_i \text{ für } i = 1, \ldots, m$$

$$x_1, \ldots, x_n \geqslant 0.$$(*Web*)

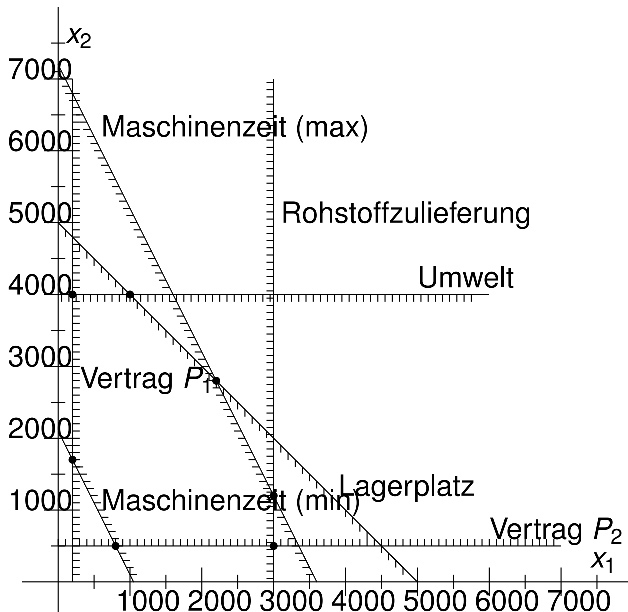
Ein Beispiel

- Ein Unternehmen verfügt über eine Maschine, auf der zwei Produkte P_1 und P_2 produziert werden können.
- Eine Einheit von P_1 bringt 7 Euro Gewinn, eine Einheit P_2 bringt 4 Euro. Die Zielfunktion ist $7 \cdot x_1 + 4 \cdot x_2$.
- Die Produktion einer Einheit von P_1 nimmt die Maschine 4 Minuten in Anspruch, eine Einheit P_2 ist in 2 Minuten fertig. Die Maschine steht im Monat für 240 Stunden, also für 14400 Minuten, zur Verfügung. Wir erhalten die Ungleichung $4 \cdot x_1 + 2 \cdot x_2 \leqslant 14400$.
- Weitere Bedingungen wie Rohstoffverbrauch, Absatzmöglichkeiten etc. führen auf weitere Ungleichungen.

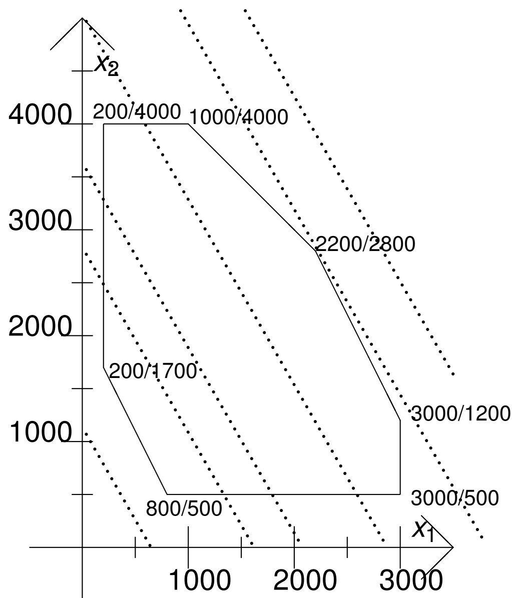
Wie kann eine optimale Lösung bestimmt werden?

Zuerst eine geometrische Interpretation.

Die Ungleichungen



Der Lösungsraum und die Zielfunktion



Effiziente Algorithmen für die Lineare Programmierung

Man bezeichnet den Lösungsraum auch als **Polytop**.

- Um eine optimale Lösung zu bestimmen, verschiebe die Gerade $7 \cdot x_1 + 4 \cdot x_2 = 0$ so weit wie möglich „nach oben“.
- Die Gerade wird dann auf eine Ecke treffen. (Ecken erfüllen zwei oder mehrere Ungleichungen exakt.)
- Die Situation im n-dimensionalen ist ungleich komplizierter, aber es gibt schnelle Algorithmen.
  - Der Simplex-Algorithmus sucht nach einer optimalen Ecke.
  - Interior-Point Verfahren durchstoßen das Innere des Polytops und sind häufig schneller.

BIPARTITES MATCHING

- n Angestellte und m Aufgaben sind gegeben.
- Weise jeder Aufgabe j höchstens einen Angestellten $i(j)$ und jedem Angestellten höchstens eine Aufgabe zu.
- $k_{i,j}$ ist die Kompetenz des Angestellten i für die Aufgabe j. Maximiere die Kompetenz

$$\sum_{j=1}^{m} k_{i(j),j}.$$

Die Formulierung als ein lineares Program:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} k_{i,j} \cdot x_{i,j} \quad \text{so dass} \quad \begin{aligned} &\sum_{i} x_{i,j} \leqslant 1 \text{ für alle Aufgaben } j, \\ &\sum_{j} x_{i,j} \leqslant 1 \text{ für alle Angestellten } i, \\ &x_{i,j} \geqslant 0 \text{ für alle } i, j \end{aligned}$$

Integrale und fraktionale Lösungen

- Wenn $x_{i,j}$ stets entweder 0 oder 1 ist, dann beschreibt x eine legale Zuordnung.
- Überraschende(!) Eigenschaft: Im Zuordnungsproblem ist jede Ecke integral, das heißt jede Komponente ist ganzzahlig.
  - Eine jede Ecke hat nur 0- und 1-Komponenten.
  - Die lineare Programmierung löst das Zuordnungsproblem.
- Im Allgemeinen wird es aber fraktionale Ecken geben.
  - **Zentrale Frage:** Wie erhält man aus einer optimalen fraktionalen Lösung eine gute integrale Lösung?

0-1 PROGRAMMIERUNG

- Eine Instanz wird durch die Matrix $A = (a_{i,j})_{1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m}$, die „rechte Seite“ $b \in \mathbb{R}^m$ und den Vektor $c \in \mathbb{R}^n$ beschrieben.
- Maximiere $\sum_{i=1}^{n} c_i \cdot x_i$, so dass

$$\sum_{j=1}^{n} a_{i,j} \cdot x_j \leqslant b_i \text{ für } i = 1, \ldots, m$$

$$x_1, \ldots, x_n \in \{0, 1\}.$$

- Die 0-1 Programmierung ist NP-vollständig.
- Im allgemeinen Fall lassen sich keine vernünftigen Approximationen berechnen, es denn die **Relaxation**

  ersetze $x_1, \ldots, x_n \in \{0, 1\}$ durch $x_1, \ldots, x_n \in [0, 1]$

  ist eine gute Approximation.

Das Rucksack-Problem

RUCKSACK

- Eine Instanz besteht aus n Objekten, wobei Objekt i das Gewicht $g_i \geqslant 0$ und den Wert $w_i \geqslant 0$ besitzt. Ein Rucksack trägt Objekte mit Gesamtgewicht höchstens G.
- Bepacke den Rucksack, so dass das `Gesamtgewicht` nicht überschritten, aber der `Wert` eingepackter Objekte maximal ist.

Also: Maximiere $\sum_{i=1}^{n} w_i \cdot x_i$, so dass

$$\sum_{j=1}^{n} g_j \cdot x_j \leqslant G \text{ und } x_1, \ldots, x_n \in \{0, 1\}.$$

- RUCKSACK ist ein Problem der 0-1 Programmierung mit nur **einer einzigen** Ungleichung.
- RUCKSACK führt auf ein NP-vollständiges Sprachenproblem, kann aber sehr scharf approximiert werden.

Graph-Probleme

INDEPENDENT SET (und CLIQUE)

- Eine Instanz ist durch einen ungerichteten Graphen $G = (V, E)$ und eine Gewichtung $w_u \geqslant 0$ für die Knoten $u \in V$ gegeben.
- Bestimme eine möglichst schwere unabhängige Menge (bzw. Clique) $V' \subseteq V$:

  keine (bzw. je) zwei Knoten sind durch eine Kante verbunden.

Ein 0-1 Programm

Maximiere $\sum_{u \in V}^{n} w_u \cdot x_u$, so dass

$$x_u + x_v \leqslant 1 \text{ für alle Kanten } \{u, v\} \text{ und}$$
$$x_1, \ldots, x_n \in \{0, 1\}.$$

Wie gut ist die Relaxation?

- Clique und Independent Set sind äquivalente Probleme,
- leider sind beide nicht vernünftig durch effiziente Algorithmen approximierbar.

VERTEX COVER

- Eine Instanz ist durch einen ungerichteten Graphen $G = (V, E)$ und eine Gewichtung $w_u \geqslant 0$ für die Knoten $u \in V$ gegeben.
- Bestimme eine möglichst leichte Knotenüberdeckung, also eine Teilmenge $V' \subseteq V$, so dass jede Kante einen Endpunkt in V' hat.
- Eine Anwendung
  - Wiederhole ein Experiment mehrmals, einige Testergebnisse sind allerdings verfälscht.
  - Setze eine Kante $\{i, j\}$ ein, wenn die Ergebnisse der Tests i und j zu stark voneinander abweichen.
  - Die verfälschten Tests bilden eine Knotenüberdeckung.
- V' ist genau dann eine Knotenüberdeckung, wenn $V \setminus V'$ eine unabhängige Menge ist.

  Erstaunlicherweise (!?) kann VERTEX COVER wesentlich schärfer als INDEPENDENT SET approximiert werden.

Kombinatorische Probleme

HITTING SET und SET COVER

- HITTING SET:
  - Eine Instanz ist durch ein Universum U und Teilmengen $T_1, \ldots, T_k$ gegeben. Alle Elemente u besitzen ein Gewicht $w_u \geqslant 0$.
  - Bestimme eine möglichst leichte Teilmenge $H \subseteq U$, so dass jede Teilmenge T_i von H getroffen wird.
- SET COVER:
  - Eine Instanz ist durch ein Universum U und Teilmengen $T_1, \ldots, T_k$ gegeben. Alle Teilmengen T_i besitzen ein Gewicht $w_i \geqslant 0$.
  - Bestimme möglichst leichte Teilmengen $T_{i_1}, \ldots, T_{i_r}$, so dass das Universum überdeckt wird.

- HITTING SET ist eine Verallgemeinerung von VERTEX COVER.
- HITTING SET und SET COVER sind äquivalente Probleme,
- beide sind deutlich schwieriger als VERTEX COVER, aber auch deutlich einfacher als INDEPENDENT SET.

Optimierungsprobleme für Zeichenketten

SHORTEST COMMON SUPERSTRING

- Eine Instanz besteht aus Strings $s_1, \ldots, s_n$.
- Gesucht ist ein möglichst kurzer String s, so dass jedes s_i ein **Teilstring** von s ist.

- SHORTEST COMMON SUPERSTRING ist ein wichtiges Optimierungsproblem in der DNA-Sequenzierung:
  - in der Shotgun Sequenzierung werden identische Kopien einer DNA-Sequenz s mehrfach in kleine, direkt sequenzierbare Teilstrings s_i zerlegt.
  - Rekonstruiere s aus den Teilstrings.
- SHORTEST COMMON SUPERSTRING hat vernünftige Approximationsalgorithmen.

Aussagenlogik

MAX SAT

- Eine Instanz ist durch eine aussagenlogische Formel ϕ in konjunktiver Normalform gegeben. Für jede Klausel k von ϕ ist ein Wert $w_k \geqslant$ gegeben.
- Bestimme eine Belegung der Variablen von ϕ, so dass Klauseln mit maximalem Wert erfüllt werden.

- Das Erfüllbarkeitsproblem ist NP-vollständig:
  MAX SAT kann nicht effizient bestimmt werden.
- Kann MAX SAT scharf approximiert werden?
  - **Für über 20 Jahre** war die Approximationskomplexität von MAX SAT unbekannt.
  - Gleiches galt für fast alle gerade erwähnten Probleme.
- Die Approximationskomplexität ist jetzt bekannt: Vernünftige, aber nicht beliebig scharfe Approximationen sind möglich.

Probabilistisch überprüfbare Beweise

Die Bestimmung der Approximationskomplexität vieler wichtiger Probleme macht eine völlig neue Sichtweise der Klasse NP erforderlich.

- **Bisher**: Akzeptiere eine Eingabe, wenn ein polynomiell langer Beweis effizient verifiziert werden kann.
- **Jetzt**: Akzeptiere eine Eingabe, wenn ein polynomiell langer Beweis bereits nach Offenlegen von wenigen, nämlich konstant vielen Beweisbits überzeugt!
  - Welche Beweisbits sind offenzulegen? Bestimme Positionen mit Hilfe relativ weniger Zufallsbits.

Die Klasse NP stimmt überein mit allen Sprachen, die **probabilistisch überprüfbare Beweise** besitzen.

Vorwissen

(1) Die Veranstaltung „Algorithmentheorie“, bzw. die Veranstaltung „Theoretische Informatik 1“.
Insbesondere sind Entwurfsmethoden für effiziente Algorithmen wie auch die NP-Vollständigkeit von Relevanz.

(2) Das Kapitel 1 im Skript stellt die wichtigsten Grundlagen aus der elementaren Stochastik zusammen.

Literatur

- Entwurf und Analyse von Approximationsalgorithmen:
  - V.V. Vazirani, Approximation Algorithms, *Springer Verlag* 2001.
  - D.P. Williamson und D.B. Shmoys, The Design of Approximation Algorithms, Cambridge University Press, 2011.
- Lineare Programmierung
  - B. Korte und J. Vygen, Combinatorial Optimization: Theory and Algorithms, 3rd edition, Springer Verlag, 2005.
  - R.J. Vanderbei, Linear Programming: Foundations and Extensions, *Kluwer Academic Publishers*, 2001.
- Probabilistisch überprüfbare Beweise:

  S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.
- Ein Skript wird zur Verfügung gestellt.

Weitere Informationen auf der **Webseite der Veranstaltung**

Administratives

Die Veranstaltung wird für die Studiengänge

- Bachelor Informatik und
- Master Informatik

angeboten.

Prüfungs- und Studienleistungen

- Mündliche Prüfung nach Absprache.
- Die in den Übungen erzielten Punkte verbessern das Resultat einer bestandenen mündlichen Prüfung um bis zu zwei „Schritte“.
  - Verbesserung um einen Schritt bei 40% der Übungspunkte,
  - um zwei Schritte bei 70% der Übungspunkte.

BITTE!

BITTE: AN DEN ÜBUNGEN TEILNEHMEN!

- Das erste Übungsblatt wird am 21.10 aus- und nach einwöchiger Bearbeitungszeit am 28.10 zurückgegeben.
- Die Übungsgruppe trifft sich zum ersten Mal am 29.10 um 10:15 im SR 307.

BITTE: Helfen Sie mir durch

- ihre Fragen,
- Kommentare
- und Antworten!

Die Vorlesung kann nur durch Interaktion interessant werden.

- Meine Sprechstunde: Dienstags 10-12.
- Kommen Sie vorbei oder sprechen Sie mich in der Vorlesung an.

NP-Optimierungsprobleme:
Die sinnvollen Optimierungsprobleme

Optimierungsprobleme

Wir betrachten Optimierungsprobleme der Form

$$\text{opt}_x f(x_0, x) \;\; \text{so dass} \;\; \text{Lösung}(x_0, x).$$

- Optimiere die `Zielfunktion` $f(x, x_0)$ bei gegebener `Instanz` x_0 über alle `Lösungen` x zu x_0.
  - x ist genau dann eine Lösung, wenn das Prädikat Lösung(x_0, x) wahr ist.
- Es ist $\text{opt} = \min$ oder $\text{opt} = \max$.

NP-Optimierungsprobleme $P = (\mathrm{opt}, f, L)$

(1) Es ist entweder „opt = min“ oder „opt = max“.

(2) Die Zielfunktion $f(x_0, x)$ ist in polynomieller Zeit (in $|x_0| + |x|$) berechenbar und die Frage, ob x_0 eine Instanz für P ist, kann in polynomieller Zeit in $|x_0|$ entschieden werden.

(4) Zu einer Instanz x_0 heisst x eine Lösung, falls das Prädikat $L(x_0, x)$ wahr ist. Wir fordern:

(4a) L besitzt nur polynomiell lange Lösungen:

$$L(x_0, x) \Rightarrow |x| \leqslant \mathrm{poly}(|x_0|).$$

(4b) Das Prädikat $L(x_0, x)$ ist in polynomieller Zeit (in $|x_0| + |x|$) auswertbar.

Approximationsalgorithmen

$P = (\mathrm{opt}, \mathrm{f}, \mathrm{L})$ sei ein Optimierungsproblem, $\varepsilon : \mathbb{N} \to \mathbb{R}$ eine Funktion.

(a) x^* ist eine **optimale Lösung** für Instanz x_0, falls

$$f(x_0, x^*) = \mathrm{opt}\ \{f(x_0, x) \mid x \text{ erfüllt das Prädikat } L(x_0, x)\ \}.$$

(b) x ist eine ε**-approximative** Lösung für Instanz x_0, falls

$$\varepsilon(|x_0|) \geqslant \max \left\{ \frac{\mathrm{opt}_P(x_0)}{f(x_0, x)}, \frac{f(x_0, x)}{\mathrm{opt}_P(x_0)} \right\}.$$

(c) A ist Approximationsalgorithmus für P, falls A für jede Instanz x_0 eine Lösung $A(x_0)$ berechnet. A ist ε**-approximativ**, falls $A(x_0)$ für jede Instanz x_0 eine ε-approximative Lösung ist.

Approximierbarkeitsklassen

$\varepsilon : \mathbb{N} \to \mathbb{R}$ sei eine Funktion.

- NPO ist die Klasse aller NP-Optimierungsprobleme.
- $\varepsilon -$ APX ist die Klasse aller Optimierungsprobleme in NPO mit effizienten $O(\varepsilon)$-approximativen Algorithmen.
- Wir kürzen $1 -$ APX durch APX ab.

  APX ist die Klasse aller Probleme mit effizienten c-approximativen Algorithmen für eine Konstante c.

Approximationsschemata

Ein polynomielles Approximationsschema

Ein Optimierungsproblem besitzt ein polynomielles Approximations-schema, wenn es einen Approximationsalgorithmus A gibt, der

- für jede Instanz und für jeden „Approximationsfaktor" $\delta > 1$ eine δ-approximative Lösung bestimmt
- und bei *fixiertem* δ eine Laufzeit erreicht, die polynomiell in der Beschreibungslänge der Instanz ist.

PTAS (polynomial time approximation scheme) ist die Klasse aller Probleme mit einem polynomiellen Approximationsschema.

- Algorithmen mit Laufzeit $n^{1/(\delta-1)}$ qualifizieren sich als polynomielles Approximationsschema:

  Die Laufzeit explodiert, wenn δ gegen 1 strebt.

- Und Algorithmen, die gutmütiger auf eine schärfere Approximationsanforderung reagieren?

Ein volles polynomielles Approximationsschema

Ein Optimierungsproblem besitzt ein volles polynomielles Approximationsschema, wenn es einen Approximationsalgorithmus gibt, der

- für jede Instanz und für jedes $\delta > 1$ eine δ-approximative Lösung bestimmt und
- eine Laufzeit erreicht, die polynomiell in n und $\frac{1}{\delta-1}$ ist.

FPTAS (fully polynomial time approximation scheme) ist die Klasse aller Probleme mit einem vollen polynomiellen Approximationsschema.

- PO sei die Klasse aller Optimierungsprobleme in NPO, die in polynomieller Zeit exakt lösbar sind.
- Wir erhalten eine Schwierigkeitshierarchie

$$\begin{aligned} \text{PO} &\subseteq \text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX} \\ &\subseteq \text{log-APX} \subseteq \text{poly(log)-APX} \subseteq \text{poly-APX} \subseteq \text{NPO}. \end{aligned}$$

Kleine Optima

Parametrisierte Algorithmen

Können wir schwierige Optimierungsprobleme schneller lösen, wenn wir wissen, dass das Optimum klein ist?

- Wie schnell können wir überprüfen, ob ein Graph G eine Knotenüberdeckung der Größe höchstens k besitzt?

  Zeit $O(n^2 \cdot \binom{n}{k})$ reicht aus: Teste alle Knotenmengen der Größe k.
- Wir zeigen, dass Zeit $O(2^k \cdot n)$ ausreicht.

Eine Vorüberlegung

Der Graph G habe n Knoten und eine Knotenüberdeckung der Größe k. Dann hat G höchstens $k \cdot n$ Kanten.

> Graphen mit kleinen Knotenüberdeckungen haben wenige Kanten.

- Eine Knotenüberdeckung $C \subseteq V$ muss einen Endpunkt für jede Kante des Graphen besitzen.
- Ein Knoten kann aber nur Endpunkt von höchstens $n - 1$ Kanten sein und deshalb hat G höchstens $k \cdot (n - 1) \leqslant k \cdot n$ Kanten.

VC(**G**, **k**): Ein schneller Algorithmus

(1) Setze $U = \emptyset$.

U wird eine Knotenüberdeckung der Größe $\leqslant k$ speichern.

(2) Wenn G mehr als $k \cdot n$ Kanten hat, dann hat G keine Knotenüberdeckung der Größe k: Halte mit der „erfolglos“ Meldung.

Wir können ab jetzt annehmen, dass G höchstens $k \cdot n$ Kanten besitzt.

(3) Wenn $k = 0$, dann halte mit einer „erfolgreich“ Meldung.

(4) Wähle eine beliebige Kante $\{u, v\} \in E$.

- Rufe **VC(G − u, k − 1)** auf. Wenn Antwort „erfolgreich“, dann

  setze $U = U \cup \{u\}$ und halte mit Nachricht „erfolgreich“.
- Ansonsten rufe **VC(G − v, k − 1)** auf. Wenn Antwort „erfolgreich“, dann

  setze $U = U \cup \{v\}$ und halte mit Nachricht „erfolgreich“.
- Ansonsten sind beide Aufrufe erfolglos. Halte mit Nachricht „erfolglos“.

Analyse

VC(G, k) überprüft in Zeit $\mathbf{O(2^k \cdot (n + m))}$,
ob ein Graph mit n Knoten und m Kanten eine Knotenüberdeckung der Größe höchstens k besitzt.

Warum?

- VC(G, k) erzeugt einen Rekursionsbaum B.
- B ist binär und hat Tiefe k.
- Es gibt höchstens 2^k rekursive Aufrufe, wobei ein Aufruf höchstens Zeit $O(n + m)$ benötigt.

Wir können VERTEX COVER effizient lösen, wenn k höchstens ein (nicht zu großes) Vielfaches von $\log_2 n$ ist.