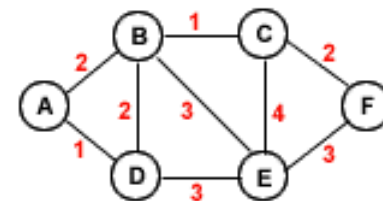




# Effiziente Algorithmen

Hartmut Klauck  
Universität Frankfurt  
SS 06

6.7.





# Lineare Programmierung

- Wir beschreiben nun eine „universelle Programmiersprache“ für alle Probleme in  $P$
- Geben einen praktisch effizienten Algorithmus an, um solche Probleme zu lösen



# Lineare Programmierung

## ○ Definition

- Ein Lineares Programm (LP) ist ein Optimierungsproblem über reellen Variablen  $x_1, \dots, x_n$
- Maximiert wird eine lineare Zielfunktion  $C(x) = \sum_i c_i x_i$  mit reellen Koeffizienten  $c_i$
- Zulässige Lösungen sind solche, die eine Menge von Ungleichungen erfüllen:
  - Sei  $A$  eine  $m \times n$  Matrix mit reellen Koeffizienten
  - Alle  $x$  mit  $Ax \leq b$  sind zulässig
- Gesucht wird eine zulässige Lösung mit maximaler Zielfunktion



# Ein Beispiel

- Das MaxFlow Problem
  - Gegeben Graph  $G$  mit Kapazitätsfunktion  $C(u,v)$
  - $G$  habe  $n^2$  Knotenpaare  $(u,v)$ , dazu verwenden wir  $n^2$  Variablen  $f(u,v)$
  - Verwende folgende Ungleichungen:
    - $f(u,v) \leq C(u,v)$  für alle  $u,v$
    - $f(u,v) = -f(v,u)$  für alle  $u,v$
    - $\sum_v f(u,v) = 0$  für alle  $u$  ausser  $s,t$
  - Maximiere  $\sum_v f(s,v)$
  - Das Programm hat  $n^2$  Variablen und  $2n^2+n-2$  Ungleichungen oder Gleichungen
  - Lösung entspricht offensichtlich einem maximalen Fluss



# Standardform

- Wir können allgemein mit  $\geq$ ,  $\leq$  und  $=$  Constraints arbeiten
- Einfache Reduktion auf Standardform:
  - $\max c^T x$
  - Bedingungen:
  - $Ax \leq b$
  - $x \geq 0$



# Slack Form

- Weitere spezielle Form von LPs
- Wir drücken nun viele Constraints als Gleichungen aus
- Sei  $\sum A[i,j] x_j \leq b_i$  eine Ungleichung
- Wir führen eine neue Variable  $x_{n+i}$  ein und setzen
- $x_{n+i} = b_i - \sum A[i,j] x_j$
- und  $x_{n+i} \geq 0$
- $x_{n+i}$  ist eine slack-Variable oder Schlupfvariable
- Wenn wir dies für alle Ungleichungen tun erhalten wir ein System mit  $n+m$  Variablen,  $m$  Gleichungen und ansonsten dem Constraint  $x \geq 0$
- Die Variablen, die links in Gleichungen stehen nennen wir Basisvariablen, die anderen Nichtbasisvariablen



# Lineare Programmierung

## ○ Fakten:

- LP werden in der Praxis meist mit dem Simplex Algorithmus gelöst
- LP-Solver in vielen Mathematik Bibliotheken
- Simplex hat im worst case exponentielle Laufzeit
- Man kann zeigen, daß nur für alle LP in einer „Umgebung“ des LP die meisten LP von Simplex in polynomieller Zeit gelöst werden (unter einer bestimmten „Pivotwahl“): smoothed Analysis, [Teng und Spielman]
- D.h. nur „notorische“ LP sind hart für Simplex
- Es gibt worst case Polynomialzeit Algorithmen:
  - Ellipsoid
  - Interior Point Methoden
- Zumindest Ellipsoid ist nicht praktikabel



# P-Vollständigkeit

- Jedes Problem in P kann auf das Problem LP reduziert werden, wobei die Reduktion durch eine Turingmaschine mit Arbeitsspeicher  $O(\log n)$  berechnet werden kann.
- Damit ist LP ein „ausdrucksstärkstes“ Problem in P
- Findet man keinen speziellen Algorithmus für ein Problem (oder hat keine Zeit dazu):  
Formuliere als LP verwende Standard LP Solver



# Einige Begriffe

- Im  $n$ -dimensionalen Raum
- Hyperebene:  $n-1$  dimensional, durch eine Gleichung definiert
- Halbraum: Alle Punkte, die eine Ungleichung erfüllen
- Konvexes Polytop: konvexe Hülle einer endlichen Punktmenge  $p_1, \dots, p_m$ 
  - Ecke eines Polytops: einer der Punkte der endlichen Menge
  - Kanten: konvexe Hülle von 2 der Punkte
  - $d$ -Facetten: konvexe Hülle von  $d$  der Punkte
  - Seiten:  $m-1$  Facetten
- Simplex:  $n$ -dimensionales Polytop mit  $n+1$  Ecken



# Geometrie

- Betrachte  $\mathbb{R}^n$
- Jede Ungleichung entspricht einem Halbraum, der durch eine Hyperebene begrenzt wird
- Die zulässigen Lösungen liegen im Schnitt von  $m$  Halbräumen
- Diese Menge ist ein konvexes Polytop
- Die Zielfunktion „zeigt“ in eine Richtung
- Optima liegen (evtl. unter anderem) an den Ecken des Polytops



# Der Simplexalgorithmus

- Idee:
  - Bringe in slack form
  - Die Menge der zulässigen Lösungen ist ein Simplex
  - Wir starten an einer Ecke des Simplex
  - Wir führen dann Iterationen aus, bei denen wir von einer Ecke entlang einer Kante des Simplex zu einer besseren Ecke laufen
  - Dies wird wiederholt, solange möglich
  - Wir erreichen ein lokales Maximum
  - Dieses ist auch global optimal wegen der Konvexität der zul. Lösungen und der Linearität der Kostenfunktion



# Simplex Algorithmus

- Gegeben sei ein lineares Programm in slack form
- Es gebe  $m$  Basisvariablen und  $n$  Nichtbasisvariablen
- Definiere eine Startlösung:
  - Setze alle Nebenvariablen auf 0 und alle Hauptvariablen entsprechend ihrer Gleichung
  - Wir nehmen im Augenblick an, dies sei eine zulässige Lösung
  - Weiterhin ist dies eine Ecke des Simplex
- Wir wechseln von einer slack form des LP zu einer anderen, so daß die soeben definierte Lösung besser (oder zumindest nicht schlechter) wird
- Solange bis „lokales“ Optimum erreicht



# Wechsel der slack form

- Wir wählen eine Nichtbasisvariable, bei der eine Erhöhung ihres Wertes die Zielfunktion verbessert (d.h.  $x_e$  mit  $c(e) > 0$ )
- Dies beeinflusst die Werte der Basisvariablen
- Wir erhöhen  $x_e$ , bis eine der Hauptvariablen  $= 0$  sein muß (ist dies nicht der Fall, so ist die Zielfunktion unbeschränkt)
- Umschreiben des LP: Wechsel der Haupt- und Nebenvariablen
  
- Iteration



# Operation pivot

- Eingaben:
  - Ein slack form LP:
    - N: Indizes der Nichtbasisvariablen
    - H: Indizes der Basisvariablen
    - A,b: Matrix und Vektor des „normalen LP“
    - v: Konstante in der Zielfunktion
    - c: Koeffizienten der Zielfunktion
    - e: Index einer Nichtbasisvariablen
    - l: Index einer Basisvariablen
- Ausgabe wie Eingabe;
  - neues slack form LP mit  $x_e$  als Basisvariablen und  $x_l$  als Nichtbasisvariablen



# Operation pivot

1.  $b'(e) = b(l) / A(l, e)$
2. Für alle  $j$  aus  $N - \{e\}$ :  $A'(e, j) = A(l, j) / A(l, e)$
3.  $A'(e, l) = 1 / A(l, e)$
  
4. Für alle  $i$  aus  $H - \{l\}$ :
  1.  $b'(i) = b(i) - A(i, e)b'(e)$
  2. Für alle  $j$  aus  $N - \{e\}$ :
    1.  $A'(i, j) = A(i, j) - A(i, e)A'(e, j)$
    3.  $A'(i, l) = -A(i, e)A'(e, l)$
  
5.  $v' = v + c(e)b'(e)$
6. Für alle  $j$  aus  $N - \{e\}$ :
  1.  $c'(j) = c(j) - c(e)A'(e, j)$
7.  $c'(l) = -c(e)A'(e, l)$
8. Update der Mengen  $H, N$