Discrete Optimization 2010
Lecture 1
Introduction / Algorithms & Spanning Trees

Marc Uetz
University of Twente

m.uetz@utwente.nl

Outline

Organization

Lectures

- lecture of 2 $\times$ 45 min. approx., maybe sometimes:
- discussion of homework exercises

Your Assignments

- Following the lectures
- Reading the literature
- Solving homework assignments (in teams of 2-3, please)
  - Hand-In (next Lecture) or email (by Monday 13:00)
  - Homework is corrected by TA (Ruben Hoeksma)
- Written exam

Organization

Grading

- Homework exercises 40%
- Written exam 60%

Norm

- to pass, exam ≥ 5.5 and total grade ≥ 5.5

Online Material

http://www.math.utwente.nl/~uetzm/do/

Literature

Reader, with selcted chapters from

- R. K. Ahuja, T. L. Magnanti and J. B. Orlin: Network Flows, Prentice Hall, 1993.
- Cook, W. J., W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, Wiley, 1998.
- Cormen, T. H., C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
- U. Faigle, W. Kern, and G. Still: *Algorithmic Principles of Mathematical Programming*, Springer, 2002
- V. V. Vazirani, *Approximation Algorithms*, Springer, 2001.

For the reader, pay € 5,- in cash or transfer to account

ABN-AMRO bank (BIC: ABNANL2A)
Account nr. 405343663 (IBAN: NL33ABNA0405343663)
Reference: "OFI 500.30100, LNMB reader"

Overview & Schedule (Preliminary)

Date	Session Topics
20.09.	Introduction, Minimum Spanning Trees
27.09.	Matroids, Shortest Path Algorithms
04.10.	Maximum Flow Algorithms
11.10.	Minimum Cost Flow Algorithms
18.10.	Matchings & Total Unimodularity
25.10.	Glimpse of Integer Programming
01.11.	P, NP, NP-completeness
08.11.	NP-complete problems
15.11.	Approximation Algorithms
22.12.	Randomized Algorithms & Derandomization
29.11.	Primal Dual Approximation Algorithms
06.12.	Approximation Schemes & Inapproximability
17.01.	Exam (Location & Time TBA), Utrecht

Outline

Optimization Problems

Definition

An instance $I = (S, f)$ of an optimization problem P is

- $S =$ set of solutions
- $f : S \to \Re$ objective function

Any $s \in S$ is a solution, and $f(s)$ its value.

Definition

A solution s^* is an optimal solution for instance $I = (S, f)$ if

$$f(s^*) \ \leq (\geq) \ f(s) \quad \forall s \in S \, .$$

Optimization Problems

Definition

An optimization problem $P =$ set of all instances $I \in P$ (sharing the same description)

Examples:

- Linear Programming instances: $c \in \mathbb{Q}^n, A \in \mathbb{Q}^{n \times m}, b \in \mathbb{Q}^m$
- MST Instances: edge-weighted graph $G = (V, E, c)$

Definition

A combinatorial (discrete) optimization problem P is an optimization problem, so that for any instance $I = (S, f) \in P$ the solution set S is finite (or at most countable)

Example: Linear Programming a.k.a. LP

Each instance of LP (standard form) is given by

- $c \in \mathbb{Q}^n, A \in \mathbb{Q}^{n \times m}, b \in \mathbb{Q}^m$

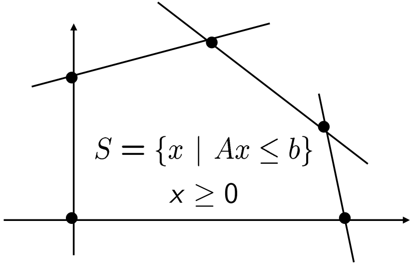
$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

- solutions $S = \{x \in \Re^n \mid x \geq 0, Ax \leq b \}$
- values $f(x) = cx$ for all $x \in S$

LP problem

Find solution $x \in S$ minimizing cx

LP is a Combinatorial Problem



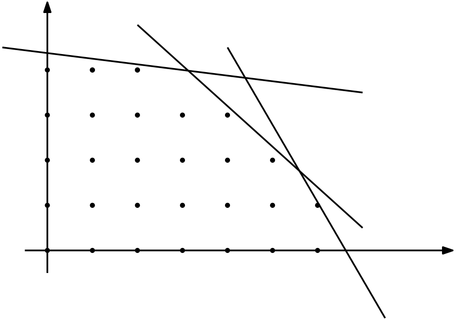
Suffices to consider finitely many solutions

$$\begin{aligned} S &= \{\bullet\} = \text{basic feasible solutions} \\ &= \text{vertices of the polyhedron } \{x \in \Re^n \mid x \geq 0, Ax \leq b\ \} \end{aligned}$$

Solved combinatorially by Dantzig's simplex algorithm (1949)

Integer Programming a.k.a. IP

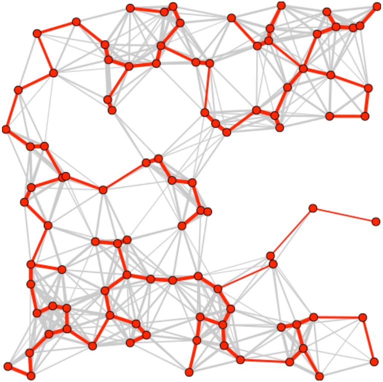
Just as LP, only restrict to x integer



Clearly: solutions set S finite (or, at most countable) by definition, as vertices in general not integer, not solved via LP

Spanning Trees

Definition: Minimal connected subgraph of given $G = (V, E)$



Minimum Spanning Tree a.k.a. MST

Each instance of MST is given by

An edge-weighted undirected graph $G = (V, E, c)$ with

- V = set of vertices / nodes
- E = set of edges $\subseteq \{\{v, w\} \mid v, w \in V\}$
- edge costs c_e for all $e = \{v, w\} \in E$

- solutions $S = \{T \subseteq E \mid T \text{ is a spanning tree}\}$
- values $f(T) = \sum_{e \in T} c_e$

MST problem

Find a minimum cost spanning tree T

Algorithms

Definition

Informally: Set of rules specifying a computational procedure, where *rules* are arithmetic and logical instructions (such as $:=$, $\pm$, $\cdot$, / , $\leq$, AND, OR, IF, THEN, ...) which, given some initial state (input), terminate in some final state (output)

Etymology

Persian Al-Khwarizmi (780-850), father of Algebra, wrote "On Calculation with Hindu Numerals"; this was translated into latin "Algoritmi on...", later evolving into "algorithm" = "calculation method"

Algorithm Design

Algorithm = think of a procedure that computes solutions to (combinatorial) problems

Main Issues

- Efficiency: How long does it take to compute a solution?
- Quality: Claims about the quality of the solution?

Possibilities for analyzing algorithms

- empirical
- average case
- worst case

Why so Pessimistic (Worst Case)?

Empirical Analysis

- Computer dependent (Moore's Law: factor 2 per 18 months)
- Language dependent (C++, C#, Java, C, FORTRAN,...)
- Compiler & Programmer dependent (Code optimization,...)

Average case

- Distribution of problem instances?
- Difficult analytically

Worst Case

- Simple & sound statements: Performance guarantees!
- Downside: too pessimistic ("pathological instances")

Computation Time of an Algorithm

Definition

The computation time of an algorithm is the number of basic instructions that the algorithm performs until termination.

Want ...

- asses computation time for problems, rather than instances
- focus only on behavior for large instances (size$\longrightarrow \infty$)

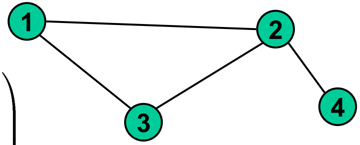
Need to capture

1. how to asses the problem size?
2. if size$\longrightarrow \infty$, what computation time? (asymptotic analysis)

Problem Size & Data Structures

\# bytes to represent the instance, depends on data structure

Example: Given a graph $G = (V,E)$. $|V|=n$, $|E|=m$.



Adjacency matrix (a_{ij}):

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

n^2 = 16 bytes

Adjacency list L :

$$L_1 \rightarrow \{2,3\}$$
$$L_2 \rightarrow \{1,3,4\}$$
$$L_3 \rightarrow \{1,2\}$$
$$L_4 \rightarrow \{2\}$$

$n+2m$ = 12 bytes

Why Data Structures Matter

Consider Problem "EDGE"

- Given graph $G = (V, E)$, two vertices $i, j \in V$
- Question: Does edge $\{i, j\}$ exist?

Adjacency matrix A: If $(a_{ij} == 1)$ return "yes"; else return "no";
one basic instruction only $O(1)$

Adjacency lists L:
for all $k \in L_i\{$
  if $(k == j)$ return "yes";$\}$
return "no";
at most $|L_i| + 2$ basic instructions $O(n)$

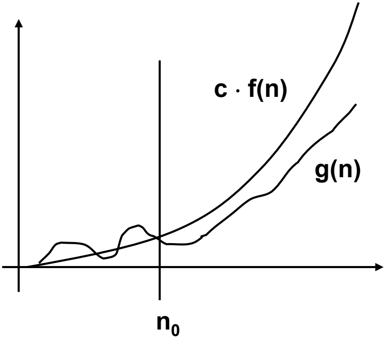
Side remark: Why would I ever use adjacency lists then?

Asymptotic upper bounds (Big-O)

Definition

$g(n) \in O(f(n)) \quad \Leftrightarrow \quad$ There is constant $c > 0$ and $n_0 \in \mathbb{N}$ so that

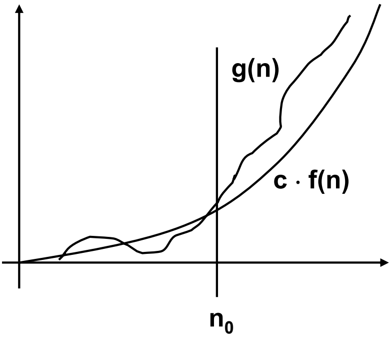
$$g(n) \leq c \cdot f(n) \;\forall\; n \geq n_0$$



Asymptotic lower bounds (big-Ω)

Definition

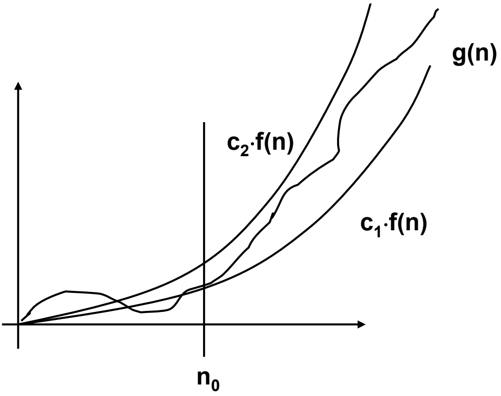
$$g(n) \in \Omega(\, f(n)\,) \quad \Leftrightarrow \quad \text{There is constant } c > 0 \text{ and } n_0 \in \mathbb{N} \text{ so that } g(n) \geq c \cdot f(n) \; \forall \; n \geq n_0$$



Asymptotic equivalent (big-Θ)

Definition

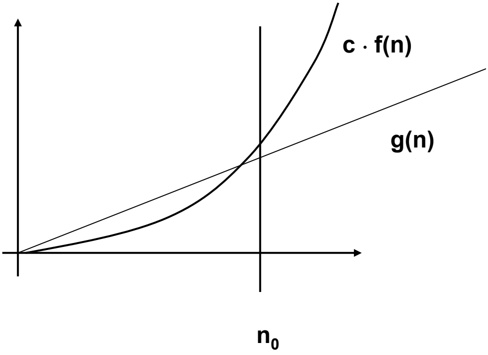
$$g(n) \in \Theta(\, f(n)\,) \quad \Leftrightarrow \quad g(n) \in O(\, f(n)\,) \text{ and } g(n) \in \Omega(\, f(n)\,)$$



Asymptotically Insignificant (little-o)

Definition

$g(n) \in o(\, f(n)\,) \quad \Leftrightarrow \quad \forall$ constants $c > 0$ there is $n_0 \in \mathbb{N}$ so that $g(n) < c \cdot f(n) \;\forall\; n \geq n_0$



Examples (you may try to prove them all)

- $7n^2 + 1000n \in O(n^2)$
- $1000n \in O(n^2)$
- $n \log n \in O(n^2)$
- $n! \in \Omega(2^n)$
- $7n^2 + 1000n \in \Theta(n^2)$
- $\log n \in o(n^\epsilon)$ for all $\epsilon > 0$
- $n \log n \in o(n^2)$
- $n^k \in o(c^n)$ for all k and all $c > 1$

Notation for Functions $f \in \ldots$

$\mathrm{O}(1)$	constant
$\mathrm{O}(\log n)$	logarithmic
$\mathrm{O}([\log n]^c)$	polylogarithmic
$\mathrm{o}(n)$	sublinear
$O(n)$	linear
$\mathrm{O}(n \log n)$	quasilinear
$\mathrm{O}(n^2)$	quadratic
$\mathrm{O}(n^c), c > 0$	polynomial
$\Omega(c^n)$	exponential
$\Omega(n!)$	factorial or combinatorial

Encoding Length of a Problem

Example: Given graph $G = (V, E)$ with $|V| = n$ and $|E| = m$

Adjacency matrix

Encoding length $\ell(G) \in \Theta(\, n^2 \,)$ $[\frac{1}{2}n^2]$

Adjacency lists

Encoding length $\ell(G) \in \Theta(\, n + m \,)$ $[n + 2m]$

Notice: both encoding length of problems and computation time of algorithms depend on data structures! Is that a problem?

Polynomial Equivalence

Given a combinatorial optimization problem P and two different encodings L_1 and L_2 (with encoding lengths $\ell_i, i = 1, 2$)

Definition

Encodings L_1 and L_2 are polynomially equivalent if there are two polynomial functions p_1 and p_2 such that, for all instances I,

- $\ell_1(I) \leq p_1(\ell_2(I))$
- $\ell_2(I) \leq p_2(\ell_1(I))$

Theorem

Adjacency lists and adjacency matrix are polynomially equivalent encodings for graphs. (Proof: Exercise) □

Encoding Numbers

Theorem

Given any (natural) number n, its encoding in binary is $\Theta(\log n)$

Recall, 1=1, 2=10, 3=11, 4=100, $5 = 101$, ..., $2^k = 1\,\underbrace{0000}_{k \text{ times}}$,...

Proof: Let k be such that

$$2^{k-1} \leq n < 2^k$$

then binary has exactly $k = \lfloor \log_2 n \rfloor + 1 \in \Theta(\log_2 n)$ digits $\square$

Remark

All k-ary encodings are polynomially equivalent with binary, as long as $k \in O(1)$ and $k > 1$, because $\log_k n = \log_2 n / \log_2 k$

Polynomial Time Algorithms

Definition

Given a combinatorial optimization problem P and encoding L.
Algorithm A is a polynomial time algorithm for P if

- for any instance $I \in P$, A terminates with a solution s of I
- there is a polynomial function p such that, if $n_I = |L(I)|$ is the encoding length of I, then

$$t_A(I) \in \mathrm{O}(\, p(n_I)\,) \quad \forall\, I \in P$$

where $t_A(I)$ is the number of basic instructions of A on I

- computation time is then said to be in $\mathrm{O}(\, p(n)\,)$
- A solves problem P if solution s is optimal for all $I \in P$
- encodings don't matter, as long as polynomially equivalent

Remarks on Polytime Algorithms

The definition

- addresses the worst case (true for all instances) $\surd$
- is relative to problem size, as it should be $\surd$
- is asymptotic (of interest is size$\longrightarrow \infty$) $\surd$
- is insensitive to equivalent problem encodings $\surd$

Question

Why care about polynomial time algorithms anyhow?

Answer 1

ask him: Jack Edmonds (Photo from 2009)

Answer 2: Computation Times on 2.2 GHz

Say, we can do $2.2 \cdot 10^9$ operations per second (2.2 GHz)

	algorithm A's computation time[a], for $t_A(n) =$				
n	$\log n$	$n \log n$	n^2	n^3	2^n
16	≈ 0	≈ 0	≈ 0	0.002 ms	0.03 ms
64	≈ 0	≈ 0	0.002 ms	0.12 ms	266 y
256	≈ 0	0.001 ms	0.06 ms	7.6 ms	$1.6 \cdot 10^{60}$ y
4.096	≈ 0	0.02 ms	7.6 ms	31 s	???
65.536	≈ 0	0.47 ms	2 s	78 h	???
16.7 Mio	≈ 0	0.2 s	35 h	68066 y	???

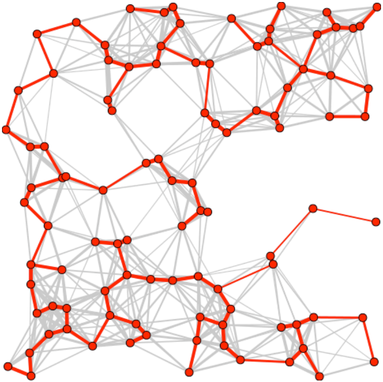
[a] s=second, ms=1/1000 s, h=hour, y=year

Outline

1. Introduction
2. Analysis of Algorithms
3. Minimum Spanning Trees

Spanning Trees

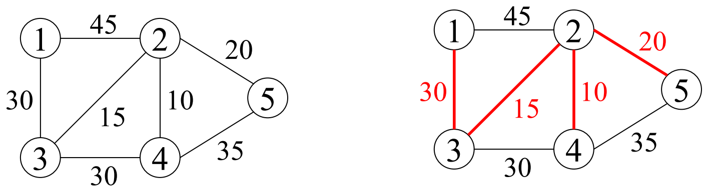
$T \subseteq E$ is a spanning tree for graph $G = (V, E)$ if graph (V, T) is connected and acyclic.
$\Leftrightarrow$ graph (V, T) is connected/acyclic and $|T| = |V| - 1$
$\Leftrightarrow$ in (V, T) there is a unique path between any two $v, w \in V$



Minimum Spanning Trees (MST)

MST problem

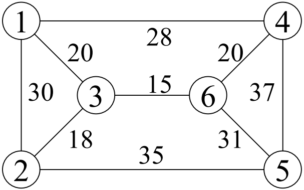
Given an edge-weighted, connected, undirected graph $G = (V, E, c)$, with $|V| = n$ and $|E| = m$, find a minimum weight spanning tree (MST).
(cheapest connected acyclic subgraph)

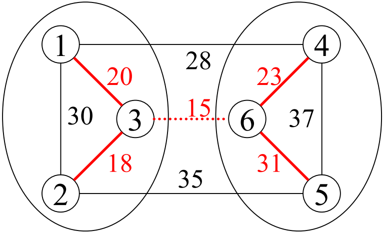


Spanning Trees and Cuts

Definitions

- For subset of nodes $W \subseteq V$, $\delta(W)$ denotes the cut induced by W: all edges "leaving" W $[\delta(W) = \delta(V \setminus W)]$
- Given a spanning tree T of G, let $C(e)$ be the cut induced by deleting edge e from T





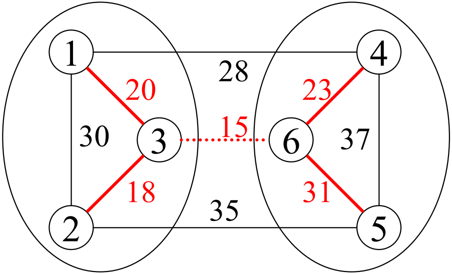
The Cut Condition

Theorem

Given a graph $G = (V, E)$ with edge costs c_e, $e \in E$, then a spanning tree $T \subseteq E$ is an MST if and only if

$$c_e \leq c_f \text{ for all edges } e \in T \text{ and all edges } f \in C(e)\,.$$

(i.e., T has the cheapest connection for any cut)



The Cut Condition

Proof

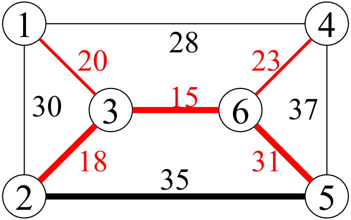
The Path Condition

In a tree T, denote by $P_T(v,w)$ the (unique) path from v to w

Theorem (Exercise)

Given a graph $G=(V,E)$ with edge costs c_e, $e \in E$, then a spanning tree $T \subseteq E$ is an MST if and only if

$c_e \leq c_f$ for all edges $f=\{v,w\} \in E \setminus T$ and all edges $e \in P_T(v,w)$



This does not mean that $P_T(v,w)$ is a shortest (v,w)-path!

Kruskal's Algorithm (1956)

Algorithm 1: `Kruskal`

input : $G = (V, E, c)$
output: $T \subseteq E$, minimum spanning tree of G

sort edges such that $c_{e_1} \leq \cdots \leq c_{e_m}$;
$T = \emptyset$;
for $(i = 1, \ldots, m)$ **do**
  if $(T \cup e_i)$ *is acyclic* **then**
    $T = T \cup e_i$;

Theorem

Kruskals algorithm solves MST problem in time $\mathrm{O}(\, m \log m + n^2\,)$.

Kruskal's Algorithm: Correctness

Proof that T is a tree

- resulting T is acyclic by definition
- for any $W \subseteq V$, T contains the cheapest edge from $\delta(W)$ (by the sorting), therefore T is connected $\quad (T \cap \delta(W) \neq \emptyset)$

$\square$

Proof that T is MST (using the path condition)

- any edge $f = \{v, w\}$ not added to T creates cycle, namely $\{v, w\} \cup P_T(v, w)$,
  in particular, all edges $e \in P_T(v, w)$ are already in T
- by the sorting of the edges $\Rightarrow c_f \geq c_e \; \forall e \in P_T(v, w)$ $\square$

Kruskal's Algorithm: Computation Time

To start with

- $O(m \log m)$ for sorting the c_e values (MergeSort)
- need to do m times: Is $T \cup e$ acyclic?, and if so, add e to T

We need a clever data structure!

Store & update to which component any node belongs:

- Initialize $x(v) = v \; \forall \; v \in V$ [n components] $O(n)$

m times we do for an edge $\{v, w\}$:

- Check ($T \cup \{v, w\}$ acyclic) $\Leftrightarrow x(v) \neq x(w)$ $O(1)$
- If yes, add $\{v, w\}$ to T [merge 2 components, $n - 1$ times]:
  For all $i \in V$: If $x(i) == x(w)$ let $x(i) := x(v)$ $O(n)$

$$O(m \log m) + O(n) + mO(1) + (n-1)O(n) \in O(m \log m + n^2)$$