Algorithm Complexity and NP-Completeness

Aim is to divide problems into "hard problems and "easy" problems.

Eg TSP:

Given an integer $m > 0$ and the distance between every pair of cities. A tour is a cycle that visits with each city exactly once. TSP is to find shortest length tour.

$$\#\text{ tours} = \frac{(m-1)!}{2}$$

In the worst case, any algo will require # operations that is of a similar size to the # tours.

MST:

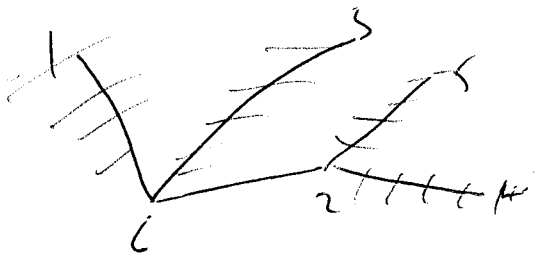
spanning trees is m^{m-2}

But we saw an algo (Prim's) that required # operations at most proportional to $m^2 \ll m^{m-2}$ for large m.

But how why:
A lower bound: $(m-1)$ Since connect vertex 2 to other vertices
Connect 3 to 1 or 2
Connect 4 to 1 or 2 or 3
etc.
Lower bound: forces tree to contain $(1,2)$

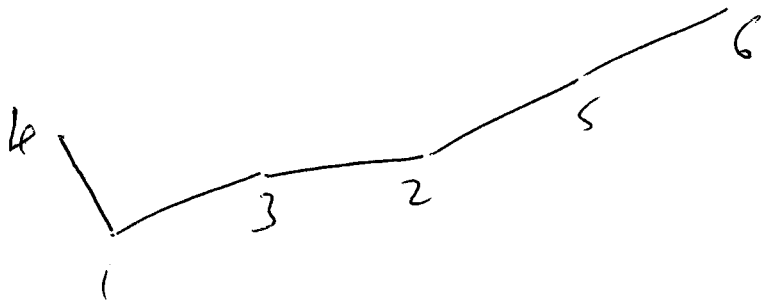
Prof: Find lowest valued leaf v, Leave a_i be the neighbour of v.
Delete v, repeat
Order a_i gives a sequence of $m-2$; is
Delete $m-2$ nodes to give anything. Last edge

MST $\longleftrightarrow$ sequences of any $n-2$ numbers between 1 and n.

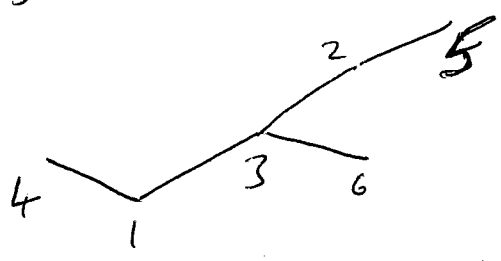


6 6 2 2 6

1 3 2 5 6



1 3 3 2 6



Size of a problem instance:

Measure complexity of an algorithm in terms of size of input

Roughly, how many binary characters are required to store the problem

Eg: i) input is an integer.

if $\exists x$ integer, $2^n \le x < 2^{n+1}$, then

$$x = \sum_{i=0}^{n} \delta_i 2^i \qquad \delta_i \text{ binary}, \ i = 0, \ldots, n$$

where δ_i are unique.

So x can be stored by storing the $(n+1)$ binary digits $\delta_0, \delta_1, \ldots, \delta_n$.

Note that $n \le \log_2 x < n+1$

Need additional digit to represent sign of x.

Rationals represented by two integers.

ii) Input is an LP (with integer data):

$$\begin{aligned} \min \quad & c^T x \\ & Ax = b \\ & x \ge 0 \end{aligned}$$

A is $m \times n$.

Size of problem instance is determined by $m, n, \sum \log(\text{entries of } A, b, c)$

iii) Graph:

Can store in many different ways:

i) Store each edge explicitly
ii) Adjacency matrix
iii) Store adjacency list for each vertex

Which one is deeper depends upon number of edges (sparsity of graph)

Size of number is ~~determined~~ by n, n.

~~Algorithm~~

The major classification of algorithm efficiency we will use is very insensitive to choice of data representation.

However, there are some restrictions:

i) Alphabet used to represent data must contain at least two symbols.

eg $17_{10} = 10001$ requires 2 digits in base 10

	5	2
	17	1
517	3	10
	10	2
	517	1

ii) What can we call "data"

Eg in TSP, ~~could give a problem instance by listing all tours.~~

natural representation is a list of edges, together with their costs

~~Not~~ permitted to store all $\frac{(n-1)!}{2}$ tours and their cost.

tours grows exponentially in ~~n~~ size of graph (n) and we would regard listing this information as part of the cost of an algo for the TSP

Computation time

Count time required for an elementary operation (addition, multiplication, comparison) as unit time. (Have to be a little careful if multiply very large numbers together.)

Now an optimization problem X (eg TSP, MST, IP)

consists of an infinite ~~sequence~~ number of instances $(d_1, d_2, \ldots)$

where data for d_i is given by a binary string of length $b_i = l(d_i)$.

Let A be an algo that solves every instance of X in finite time.

Running time of A on instance d_i is given by $g_A(d_i) \in \mathbb{R}_+$

Now two instances with the same length may well require different amounts of time.

So use <u>worst-case</u> performance as a measure of the performance of A.

So measure performance of A on all instances of X of size k by

$$f_A(k) = \max\{g_A(d_i) : l(d_i) = k\}.$$

Three advantages of this approach

i) Gives a guarantee of performance of algo

ii) Independent of probability distn of instances

iii) Appears to be easiest to analyze.

How does computational time change as problem size changes?

We say $f(k)$ is $O(g(k))$ if there exists a positive constant c and a positive integer k' such that $f(k) \leq c g(k)$ $\forall$ integers $k \geq k'$.

eg $\sum_{i=0}^{p} c_i k^i$ is $O(k^p)$.

Algorithm A is a <u>polynomial time algorithm</u> for problem X if $f_A(k)$ is $O(k^p)$ for some fixed p.

Let P be the class of problems that can be solved in polynomial time.

Problem X is in P iff $\exists$ there is a polynomial time algorithm for solving X.

(Main theme in computational complexity: inherent difference between problems known to be in P and others for which no polynomial time algorithm is known)

The function f is said to be <u>exponential</u> if for some constants $c_1, c_2 > 0$ and $d_1, d_2 > 1$ and a positive integer k', we have

$$c_1 d_1^k \leq f(k) \leq c_2 d_2^k \quad \forall \text{ integers } k \geq k'.$$

eg: Enumeration of the tours of a complete graph with k nodes.

[illegible]

Example of exponential blowing up:

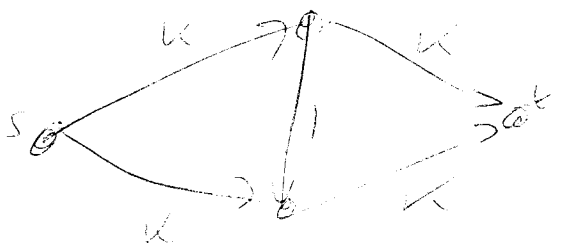
Consider $f_1(k) = 2^k$ and $f_2(k) = k^5$.

Assume basic calculations take a microsecond. (10^{-6} seconds)

If an algorithm requires 2^k calculations, it could not be completed in 300 centuries, whereas one that required k^5 calculations would be completed in < 15 minutes. $k = 60$.

Note that $2^{\log_2 k} = k$, so k is exponential in $\log k$.

($\log_2 k$ = storage $\Rightarrow$ $2^{1/4}$ ... $\leq k$)



Examples of problems in P:

i) MST

ii) Shortest path

iii) Matching

iv) Solving linear equations:

Solve $Ax = b$ A $n \times n$, nonsingular

Gaussian elimination requires n pivots, each of which requires n^2 operations.

How large can numbers get?

Size of numbers that occur is bounded by the largest magnitude of ~~the~~ determinant of any square submatrix of (A, b)

Now $\det A$ involves $n! < n^n$ terms, so largest det is $< (n\theta)^n$, where $\theta_A = \max_{ij} |a_{ij}|$, $\theta_b = \max_i |b_i|$, $\theta = \max(\theta_A, \theta_b)$

Then $\log$ (largest number) is $< n \log(n\theta) = n \log n + n \log \theta$

Thus Gaussian elimination is polynomial in size of input.

v) Linear programming:

Simplex is not polynomial

Ellipsoid is polynomial, ~~[illegible]~~

~~Office Hours.~~

Feasibility problems

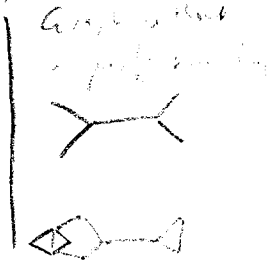
Give example first then definition

Defn A *feasibility problem* X is a pair (D, F) with $F \subseteq D$, where the elements of D are finite binary strings. D is the *set of instances* of X. F is the *set of feasible instances* of X. Given an instance $d \in D$, we want to determine whether $d \in F$. Given $d \in D$, the answer is either yes or no.

eg. 1) For ~~X = set of graphs~~ perfect matching problem "Does this graph contain a perfect matching":

D is set of graphs

F is set of graphs which contain perfect matchings.



2) 0-1 feasibility

D is set of all integral matrices (A, b) where b contains one column and same number of rows as A

An instance is specified by integers n and n (dimensions of A) and numerical values for coefficients of A and b.

feasibility problem for $S = \{x \in \mathbb{Z}^n : Ax \leq b\}$.

Hence $F = \{(A, b) : \{x \in \mathbb{Z}^n : Ax \leq b\} \neq \phi\}$

"Yes" answer is established by exhibiting a feasible x.

("No" answer is established by exhibiting a $y \geq 0$ and $A^T y = 0, b^T y < 0$ in the non-0-1 case

3) 0-1 lower bound feasibility.

$D = \{(A, b, c, z)\}$

$F = \{(A, b, c, z) : \{x \in B^n : Ax \leq b, c^T x \geq z\} \neq \phi\}$

Note that if 0-1 lower bound feasibility can be solved then so can optimization: use binary search. Number of steps is $\leq \log_2$ (upper bound on optimal value).

Proposition If 0-1 lower bound feasibility problem can be solved in polynomial time then the 0-1 optimization problem can be solved in polynomial time.

Proposition has an obvious generalization to other optimization problems.

Certificate of feasibility

Given an instance $d \in D$, let Q_d denote a certificate of feasibility. This is information that can be used to check feasibility in polynomial time.

The length of Q_d must be polynomial in the length of the data.

Eg: for perfect matching problem, Q_d will be a list of edges. So to check feasibility, we check that the edges listed in Q_d do in fact comprise a perfect matching.

Given d, string Q, want to check that Q is a certificate of feasibility.

We say that a feasibility problem (D.P.) is in NP if there exists a certificate of feasibility for $d \in F$ that can be checked in polynomial time.

Restrict $\mathcal{P}$ to feasibility problems.

Eg.) the perfect matching feasibility problem is in NP:

If $d \in F$, let Q be a list of edges of a perfect matching.

Then we can check that the edges do not share any endvertices and that every vertex is incident to one of the edges.

ii) 0-1 feasibility:

If $d \in F$, let Q be a feasible binary point x.

We can check in polynomial time that $Ax \leq b$, and x is binary.

iii) ~~Feasibility type problem corresponding to the TSP, i.e. the~~

Hamiltonian circuit problem. Does $\exists$ tour through the n cities?

If $d \in F$, let Q be a list of edges in a tour.

iv) TSP feasibility problem. Does $\exists$ tour with value at most z?

If $d \in F$, let Q be a list of edges in a tour with value no more than z.

Clearly, $P \subseteq NP$, since if $X \in P$ the algorithm itself gives a polynomial time verification.

is $P = NP$??

Maybe the major question of computational complexity.

It is generally accepted that $P \neq NP$, but not proven.

There is a class of problems (NP-complete problems) such that if a polynomial time algo existed for any of them, then that would imply $P = NP$.

Eg, the TSP is NP-complete, knapsack is NP-complete.

NP-complete problems are hardest problems in NP.

$co\text{-}NP$. $X = (D, f) \in co\text{-}NP \iff \bar{X} = (D, \bar{F}) \in NP$

Eg: Max flow — $\bar{X}$ is: does a cut ~~be~~ ?

~~$\bar{X} = (D, F)$~~ ~~Eg~~ ~~D = all graph~~

~~F = all graph w/o perfect matching.~~

X: Does flow $>$?

Conjecture: $P = NP \cap co\text{-}NP$.

"easy" problem - P
"hard" problem - NP-complete
relationship "easier than" or "not more difficult than"

~~If~~ Let X, Y are two problems:
(a) If X is ~~easy~~ and Y is easier than X then X is easy
(b) If X is hard and X is easier than Y then Y is hard

Papadimitriou & Steiglitz p. 351:

"Why is NP an interesting class of problems?

...

the recognition versions of <u>all reasonable</u> combinatorial optimization problems are in NP, ... (because) ... C.O. problems aim for the optimal design of objects.

It is reasonable to expect that, once found, the optimal solution can be written down concisely, and thus serve as a certificate for the recognition version".

Something not in NP:

Solution of quadratic inequalities:

Given $Q_1, \ldots, Q_m$ symmetric $n \times n$ matrices, and scalars $b_1, \ldots, b_m$, does there exist n-vector x satisfying $x^T Q_i x \leq b_i$ for $i = 1, \ldots, m$?

Eg: with $n=1$, $m=2$: $x^2 \leq 2$

$-x^2 \leq -2$.

Only feasible solution is $\sqrt{2}$, which cannot be written down in polynomial time.

Can ask for a RATIONAL x: then problem is in NP.