

Problems vs. Algorithms

Some problems are easier to check
than solve

Interested in the problem, rather than
particular algorithm

What's the best

In particular, problems with
polynomial-time algorithms

Polynomial Time

What does it mean for an algorithm A
to run in polynomial time

Let $T(n)$ be the worst-case time complexity
of A on any input of size n

There is a k such that

Informally, P is the set of all problems
that have polynomial-time algorithms

NP = Set of all problems where you can
a solution to a problem
instance in polynomial time.

P versus NP

$P \subseteq NP$, but there are probably some problems in

This question is probably among the 10 greatest open problems in mathematics

Hardest problems in NP are called NP-complete:

If Q is in NPC, it means that if any problem is in NP - P, then

Subtle Line between P and NP

Can have similar problems, one in P, one in NPC (hence)

1. Short path

Given nodes v, w in a labeled directed graph G , and a constant c , is there a simple path from v to w of length

In P?

2. Long Path

Given nodes v, w in a labeled directed graph G , and a constant c , is there a simple path from v to w of length

Not known to be in P, but can we check a solution in polynomial time?

Deciding versus Optimizing

Note that I phrased these as yes/no problems

Decision Problem: Is there a solution meeting a particular condition

Optimization Problem: What's the "best" () answer to the problem

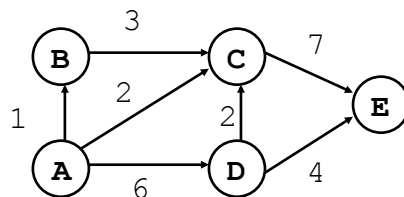
We will generally talk about decision problems.

Problem versus Problem Instance

Problem: Given a directed graph G with edge weights, nodes v, w in G and a constant c , is there a simple path from v to w of length

Problem instance

In the graph below, is there a simple path from A to E with length < 9 ?



Problem Reduction

Polynomial-time reduction from
problem Q to problem R

Transformation (function) f

- f can be computed in
- maps each instance I of Q to
- answer to I is yes if and only if

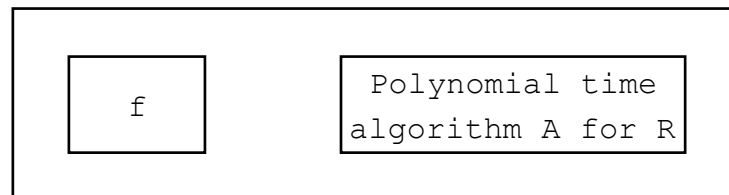
Why do we care?

Says that if R is in P, then Q is in P
Or that if Q not in P, then

Solving Q with R

How to construct poly-time algorithm
B for Q given a poly-time algorithm A
for R

B



Time Complexity of B

Suppose f is in $O(n^k)$
and A is in $O(n^m)$

What is the largest length($f(I)$) can
be?

How long will A take on $f(I)$?

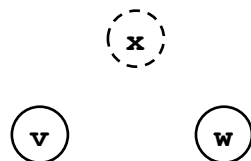
So B has time complexity in

Example Reduction

Long Cycle: Given a labeled directed
graph G , a node x and a constant c , is
there a simple cycle through x with
length

Reducing Long Path to Long Cycle

$\langle G, v, w, c \rangle$ is an instance of Long Path
Create G' that adds a new node x and edges
 (w, x) and (x, v) , each with length 1



Then reduction function f produces problem instance $\langle G', x, c+2 \rangle$ for the Long Cycle problem

Note: There is simple path from v to w in G with length $> c$ if and only if

Can we compute f in time polynomial in the length of $\langle G, v, w, c \rangle$?

Why Polynomial Time

Why not, say, just look at $O(n^4)$?

- Can compose poly-time algorithms and still be in P
- Robust to choice of computing model
 - Random-Access Machine
 - Turing Machine
 - Post Correspondence System
 - Lambda Calculus
 -
 -

Issue: Encoding Instances

How do we encode instances of a problem?

Could have a “bloated” encoding

Consider the weights in the Long Path problem. Suppose I encode an edge weight of j as a string of

Then $O(n)$ can be very large compared to the number of

Reasonable Encodings

Assume a “sensible” encoding.

- Numbers in binary (or another non-monadic base)
- Set, list has length proportional to the

Particular representation usually doesn't make a difference to whether problem is in P

Most sensible reps can be converted between in polynomial time

For example, Short Path is polynomial no matter if input graph is given as

Concrete Problem

Concrete problem = abstract problem +

Use $\langle I \rangle$ to stand for encoding of a problem instance I .

Look at problems as languages for problem Q

$L_Q = \{ \langle I \rangle \mid I \text{ is an instance of } Q \text{ with answer yes} \}$

$L_{\text{Long Path}} =$

$\{ \langle G, v, w, c \rangle \mid \text{There is a simple path from } v \text{ to } w \text{ longer than } c \text{ in } G \}$

What's Not in the Language?

What do strings look like that are not in $L_{\text{Long Path}}$?

$\langle G, v, w, c \rangle$ where there is **not** a simple path from v to w longer than c in G

What else?

Precise Definition of P

Set of all languages L such that there is a polynomial time algorithm A that decides L .

A decides L :

Polynomial-Time Verification

Certificate: Evidence that a solution exists for a problem instance

For example, certificate for $\langle G, v, w, c \rangle$ might be $\langle v_1, \dots, v_m \rangle$ where v_1, \dots, v_m is

Note, a problem instance can have

Verification algorithm A : $A(x,y) = 1$ if y is a certificate for x

Language of the algorithm

$L_A =$

Defining NP

NP is the set of all languages L with a polynomial time verification algorithm A and certificates of length $O(n^d)$ for some d

$L =$

NP-Hard and NP-Complete

NP-Hard problem R : For every problem Q in NP, L_Q has a polynomial-time reduction to

NP-Complete (NPC): R is NP-hard and

What are some NP-complete problems?