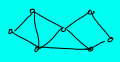
Theory of NP-completeness

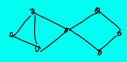
Garey & Johnson:
Computers & Intractability:
A Guide to the Thy of NP-completeness

Showing that a problem is NP-hard gives you a license to give up looking for an efficient algorithm for the problem.

Def: A decision problem is of the form:
Instance: (A description of an instance of the problem, i.e., a finite input)
Question: (A yes/no question about the instance.)

Ex: HAM-CIRCUIT:
Instance: A graph G
Question: Is there a cycle in G that runs through each vertex of G exactly once?
("Hamiltonian circuit")


yes


no

A yes-instance is one where the answer is "yes"
A no-instance - - - - - - "no"

PATH:
Instance: A weighted directed graph G, two vertices s & t, and a integer k.
Question: Is there a directed path in G from s to t of total weight $\leq k$?

We will restrict ourselves (mostly) to decision problems.

The language of a decision problem is the set of all yes-instances, suitably encoded as binary strings.

Decision problems are "essentially" langnages. That means all decision problems can be cast as problems of membership in a langnage.

Langnage L:
Instance: a binary string x
Question: Is $x \in L$?

Two complexity classes:
P and NP.

Def: P is the class of all langnages that are decidable in polynomial time.
That means — a langnage L is in P iff there exist an algorithm A that takes a binary string x as input,
- outputs "yes" if $x \in L$
- outputs "no" if $x \notin L$
- A runs in worst-case time $Poly(n)$, where $n = |x|$.

$Poly(n)$ means $O(n^c)$ for some constant c.

A is "fast" if A runs in time $Poly(n)$ where n is the length of the input.

P is the class of decision problems that are decidable in polynomial time.
("easy" problems)

Ex: PATH $\in$ P [use Dijkstra or Bellman-Ford]
HAM-CIRCUIT $\overset{?}{\in}$ P (open question)

NP (nondeterministic polynomial time)

is the class of all languages (decision problems) whose members (yes-instances) are verifiable in polynomial time.

Def: A language L is in NP iff there exists an algorithm A (the verifier) that takes two input strings x, y, such that
- A runs in worst-case time $Poly(|x|)$
- A outputs "yes" or "no"
- If $x \in L$, there exists a y such that $A(x,y)$ outputs "yes" (y is a correct proof that $x \in L$)
- If $x \notin L$, $A(x,y)$ outputs "no" for all y. ("y is not a correct proof")

Ex: HAM-CIRCUIT $\in$ NP:

A correct proof could be: A list $\langle v_1, \dots, v_n \rangle$ of vertices.

A checks that:
- each vertex occurs on the list exactly once
- (v_i, v_{i+1}) is an edge for each $1 \le i < n$
- (v_n, v_1) is an edge.

Note: If $A(x,y)$ outputs yes for some y, then there is a y' such $|y'| = Poly(|x|)$ such that $A(x,y')$ outputs yes. Because A runs in time $Poly(|x|)$, A cannot read more than the first $Poly(|x|)$ many bits of y (could just let y' be the prefix of y of length = the run time of A on $(x, -)$).

∴ Can assume wlog that $|y| = Poly(|x|)$ always.

VERTEX-COVER (VC):

Instance: A graph G and an integer $K \le |G.V|$

Question: Is there a set of vertices C such that each edge has at least one endpoint in C (the edge is "covered" by C) and $|C| \le k$?

VC $\in$ NP:

Given $\langle G, k \rangle$ and a set $C \subseteq G$ as 2nd input, check that
- $|C| \le k$
- each edge of G has at least one endpoint in C

Verifier outputting "no" can either mean
- no valid proof exists, a no-instance b/c input is
- input is yes-instance, but the proof is invalid

SATISFIABILITY (SAT)

Instance: A Boolean formula φ over Boolean variables $x_1, \dots, x_n$

Question: Is φ satisfiable, i.e. is there a setting of the variables $x_1, \dots, x_n$ (satisfying assignment) that makes φ true?

Ex: ($\bar{v}$ means NOT(v))

$\varphi = (x \vee y) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee y)$

SAT $\in$ NP:

Given a truth (0^{-1}) assignment of the variables, the verifier