Reductions

How do we show that a problem is easy?

Reduce each instance to one (or more) instances of a known easy problem.

$I_1 \in P_1 \qquad R(I_1) = I_2 \in P_2$

To solve I_1 in P_1:

1. Apply R to turn I_1 into $I_2 \in P_2$
2. Apply algorithm for P_2 to I_2.

Cost = cost of applying R plus cost of applying P_2 algorithm.

Examples

Reduction to some form of matrix multiplication:

Transitive closure

Context-free language recognition

Reduction to linear programming

Reduction to network flow

etc.

Cost of reduction?

linear time, quadratic time, ...

Positive use of reduction:

? $\Rightarrow$ easy

Reduce a problem of unknown complexity

to an easy problem

"Negative" use of reduction:

How do we show that a problem of

unknown complexity is hard?

Reduce a hard problem to it

hard $\Rightarrow$?

If the questionable problem were easy, so would be

the hard problem XX

Reductions in both directions show computational equivalence (up to the cost of the reduction)

$$P_1 \Leftrightarrow P_2$$

both are easy or both are hard

Transitivity of reductions

$$P_1 \underset{R_1}{\Rightarrow} P_2 \underset{R_2}{\Rightarrow} P_3$$

Gives a reduction from P_1 to P_3

Cost is cost of $R_2(R_1())$

both p-time, overall p-time

linear linear

R_1 R_2

n	an	bn
	an	$(ab)n$
n	an^2	bn^2
	an^2	$b(an^2)^2 = ba^2n^4$
	an^k	bn^l

$$b(an^k)^l = ba^l n^{kl}$$

Satisfiability: Is a Boolean (logical) function true for some choice of variable assignments?

$(x \vee y) \wedge (\bar{x} \vee \bar{y})$ sat: $x=1,\ y=0$

$\wedge$ and

$\vee$ or

$-$ not

x variable

$x, \bar{x}$ literal

$(x \vee \bar{y} \vee z)$ clause: disjuntion ("or") of literals

$(x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$ conjunctive normal form: conjunct ("and") of clauses

$x \vee \bar{x}$ tautology: true for all choices of variables

F is sat iff $\bar{F}$ is not a tautology: can be falsified

Reduction of CNF sat to 3-CNF sat

(at most 3 literals/clause)

$$(x \vee y \vee z \vee w \vee u \vee v) \Rightarrow$$

$$(x \vee y \vee a) \wedge (\bar{a} \vee z \vee b) \wedge (\bar{b} \vee w \vee c)$$

$$\wedge\ (\bar{c} \vee u \vee v)$$

Needs $k-3$ extra vars per clause of length k.

Graph coloring reducible to sat,

and vice-versa (p-time reductions)

Must phrase graph coloring as a yes-no question: can graph G be colored with k colors?

G: n vertices, m edges

F: nk variables x_{ij}, one per vertex per color

x_{ij} true iff vertex i colored color j

Clauses:

Each vertex colored

$(x_{i1} \vee x_{i2} \vee \cdots \vee x_{ik})$ $i \in V$ $\quad n$

[No vertex colored twice

$(\bar{x}_{ij} \vee \bar{x}_{il})$ $i \in V$, $j \neq l$ colors $\quad n\binom{k}{2}$]

No adjacent vertices the same color

$(\bar{x}_{il} \vee \bar{x}_{jl})$ $(i,j) \in E$, l a color $\quad mk$

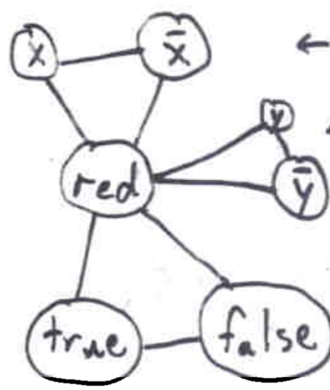
literals $= nk + 2n\binom{k}{2} + 2mk$

Vice-versa: (3-sat)

Reduction to 3-coloring

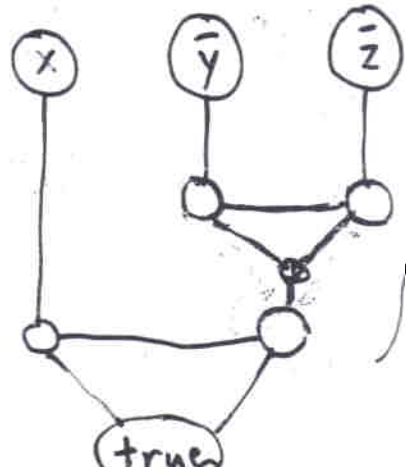
Vertices: x, $\bar{x}$, true, false, red, 5 per clause

x, $\bar{x}$ colored true, false



← every variable

Clause $x \vee \bar{y} \vee \bar{z}$



} diff. "gadget" for each clause

Colorable iff formula satisfiable

Clique $\Rightarrow$ Sat

x_{ij} $\quad i \in V, \; 1 \le j \le k$

x_{ij} true = vertex i is j^{th} in clique

$i, l \in V$ not adj.

$(\bar{x}_{ij} \vee \bar{x}_{lj'}) \quad \forall j, j'$

$(\bar{x}_{ij} \vee \bar{x}_{i'j}) \quad i \neq i' \quad \forall j$

$(x_{1j} \vee x_{2j} \vee \cdots x_{nj}) \quad \forall j$

Sat $\Rightarrow$ Clique

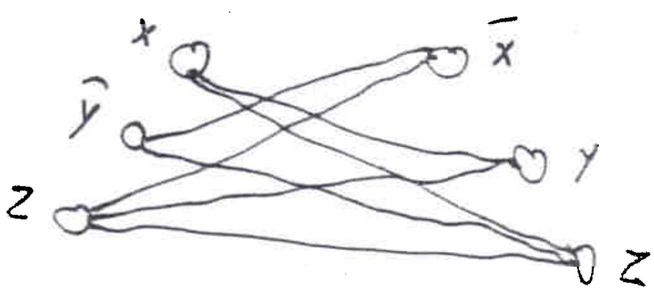
Given a graph, are there k pairwise adjacent vertices?

One vertex per literal occurrence

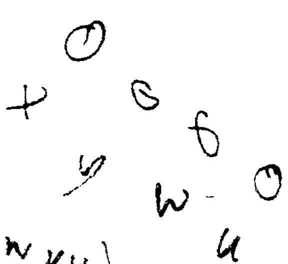
two vertices in different clauses joined by an edge if compatible (not $x, \bar{x}$)

$k =$ # clauses

$(x \vee \bar{y} \vee z)$



$(\bar{x} \vee y \vee z)$



$(x \vee y \vee w \vee u)$

Clique $\Leftrightarrow$ Independent set

Are there k pairwise nonadjacent vertices?

Complement graph

Clique $\Leftrightarrow$ Vertex cover

Are there k vertices "covering" all edges

S a vertex cover in G iff

$V-S$ is an independent set in G iff

$V-S$ is a clique in $\bar{G}$

P = problems solvable in p-time

NP = yes-no problems s.t. if answer is "yes", can be verified in p-time given a (p-length) "proof" (hint).

p-time on a Turing machine or random-access machine