13

July 5, 2016

1 NP-Completeness

How to prove a problem is hard

e.g. 3-coloring, min-length triangulation of a set of points, 0-1 knapsack, longest simple path, TSP, etc.

1.1 The Class P

(defining "easy/tractable" problems)

P = all decision problems solvable in worst-case polynomial time.

- "Decision problem" - output is "yes" or "no"
- "Polynomial" O(nˆd), where n is the input size (in # of bits)

e.g. given directed graph G, output "yes" iff there's a cycle. This is in P, by DFS.

"TSP-Decision" - input: given graph G and weight W, yes iff exists cycle C that vists each vertex exactly once & has total weight $\leq W$.

If TSP is solvable in P, then $TSPDECIS \in P$, and vice versa (by binary search in $O(log(nW_{max}))$). One could recover the original cycle by deleting edges and re-testing TSP, and see if the cycle changes (deleting edges is permanent).

1.1.1 Example: Prime

Input: n-bit number N.

Output: "yess" iff N is prime

Consider a brute force for algorithm. Runtime $O(\sqrt{N}n^2) = O(2^{n/2}n^2)$ (if we include division complexity as well).

Is PRIME in P? Yes, via some paper in number theory.

Justification for polynomial: - degree d is usually small - polynomials are closed under multiplication/composition/addition - model independent (RAM/Turing machine)

1.2 Class NP

Our theory concentrates on one common type of problems, "search problems".

NP = all decision problems that can be expressed in the form:

Input: Object x

Output: yes iff there exists object y s.t. property R(x,y) holds

Where:

1. object y has poly size
2. property R(x, y) can be checked in polynomial time