COMP 355
Advanced Algorithms

NP-Completeness: Reductions
Chapter 8 (KT)

Rhodes College

1

Recap

Decision Problems/Language recognition: are problems for which the answer is either yes or no. These can also be thought of as language recognition problems, assuming that the input has been encoded as a string. For example:

$$\begin{aligned} \text{HC} &= \{G \mid G \text{ has a Hamiltonian cycle}\} \\ \text{MST} &= \{(G,c) \mid G \text{ has a MST of cost at most } c\}. \end{aligned}$$

P: is the class of all decision problems which can be solved in polynomial time. While MST $\in$ P, we do not know whether HC $\in$ P (but we suspect not).

Certificate: is a piece of evidence that allows us to *verify* in polynomial time that a string is in a given language. For example, the language HC above, a certificate could be a sequence of vertices along the cycle. (If the string is not in the language, the certificate can be anything.)

NP: is defined to be the class of all languages that can be *verified* in polynomial time. (Formally, it stands for *Nondeterministic Polynomial time.*) Clearly, P $\subseteq$ NP. It is widely believed that P $\neq$ NP.

Rhodes College

2

Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns. | Ex.
- Greed. | O(n log n) interval scheduling.
- Divide-and-conquer. | O(n log n) FFT.
- Dynamic programming. | $O(n^2)$ edit distance.
- Duality. | $O(n^3)$ bipartite matching.
- Reductions.
- Local search.
- Randomization.

Algorithm design anti-patterns.
- NP-completeness. | $O(n^k)$ algorithm unlikely.
- PSPACE-completeness. | $O(n^k)$ certification algorithm unlikely.
- Undecidability. | No algorithm possible.

3

Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?

A working definition. [Cobham 1964, Edmonds 1965, Rabin 1966] Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

4

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

up to cost of reduction

5

Polynomial-Time Reduction

Desiderata'. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

don't confuse with reduces from

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

computational model supplemented by special piece of hardware that solves instances of Y in a single step

Notation. $X \leq_P Y$.

Remarks.

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.
- Note: Cook reducibility.

in contrast to Karp reductions

6

Reductions

Suppose that there are two problems, H and U.

If we know that H is hard (cannot be solved in polynomial time), can we prove that U is also hard?

We effectively want to show that:

- (H !∈ P) ⇒ (U !∈ P).

To do this, we could prove the contrapositive,

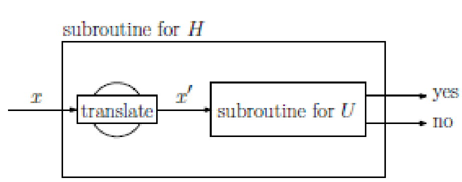
- (U ∈ P) ⇒ (H ∈ P).

To show that U is not solvable in polynomial time, we will suppose (towards a contradiction) that a polynomial time algorithm for U did exist, and then we will use this algorithm to solve H in polynomial time, thus yielding a contradiction.

7

Reductions

- Suppose we have a subroutine that can solve any instance of problem U in polynomial time.
- Given an input x for the problem H, translate it into an equivalent input x' for U. (where x ∈ H if and only if x' ∈ U)
- Run subroutine on x' and output whatever it outputs. If U is solvable in polynomial time, then so is H.
- We assume that the translation module runs in polynomial time. If so, we say we have a polynomial reduction of problem H to problem U, which is denoted $H \leq_P U$ (Karp reduction)

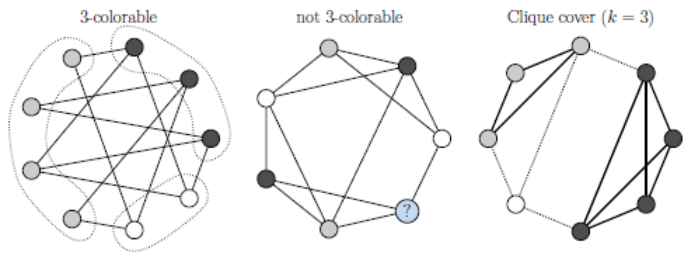


Reducing H to U

8

3-Colorability and Clique Cover

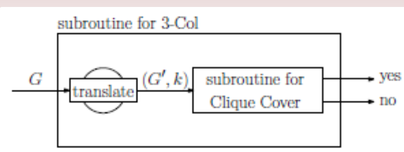
3-coloring (3Col): Given a graph G, can each of its vertices be labeled with one of three different "colors", such that no two adjacent vertices have the same label (see (a) and (b)).



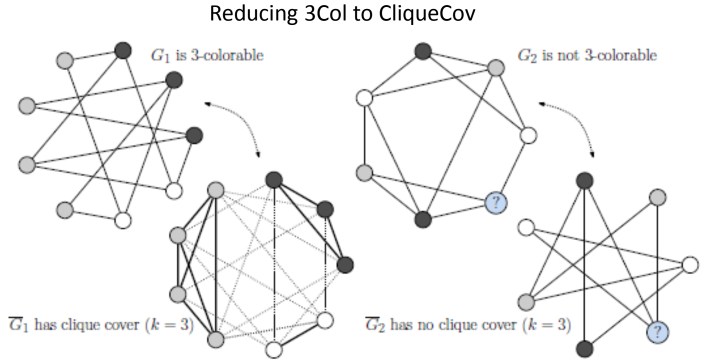
(a) (b) (c)

Clique Cover (CCov): Given a graph G = (V,E) and an integer k, can we partition the vertex set into k subsets of vertices $V_1, \ldots, V_k$ such that each V_i is a clique of G

3-Colorability and Clique Cover



Reducing 3Col to CliqueCov



(a) (b)

Proof of 3Col -> Clique Cover

Claim: A graph G = (V,E) is 3-colorable if and only if its complement G = (V,E) has a clique-cover of size 3. In other words, G ∈ 3Col ⇐⇒ (G, 3) ∈ CCov.

Proof:
(⇒) If G 3-colorable, then let V_1, V_2, V_3 be the three color classes. We claim that this is a clique cover of size 3 for G, since if u and v are distinct vertices in V_i, then {u, v} /∈ E (since adjacent vertices cannot have the same color) which implies that {u, v} ∈ E. Thus every pair of distinct vertices in V_i are adjacent in G.

(⇐) Suppose G has a clique cover of size 3, denoted V_1, V_2, V_3. For i ∈ {1, 2, 3} give the vertices of Vi color i. We assert that this is a legal coloring for G, since if distinct vertices u and v are both in V_i, then {u, v} ∈ E (since they are in a common clique), implying that {u, v} /∈ E. Hence, two vertices with the same color are not adjacent.

Polynomial-time reduction

Definition: We say that a language (i.e. decision problem) L_1 is polynomial-time reducible to language L_2 (written $L_1 \leq_P L_2$) if there is a polynomial time computable function f, such that for all x, x ∈ L_1 if and only if f(x) ∈ L_2.

Lemma: If $L_1 \leq_P L_2$ and L_2 ∈ P then L_1 ∈ P.

Lemma: If $L_1 \leq_P L_2$ and L_1 !∈ P then L_2 !∈ P.

Because the composition of two polynomials is a polynomial, we can chain reductions together.

Lemma: If $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$ then $L_1 \leq_P L_3$.

NP-completeness

Definition: A language L is NP-hard if L′ $\leq_P$ L, for all L′ ∈ NP. (Note that L does not need to be in NP.)

Definition: A language L is NP-complete if:

1. L ∈ NP (that is, it can be verified in polynomial time), and
2. L is NP-hard (that is, every problem in NP is polynomially reducible to it).

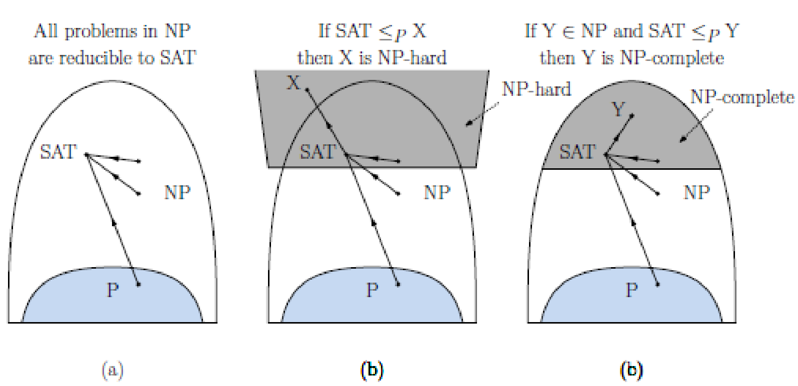
Lemma: L is NP-complete if

1. L ∈ NP and
2. L′ $\leq_P$ L for some known NP-complete language L′.

Rhodes College

13

Structure of NPC and reductions



Rhodes College

14