CSC 373: Algorithm Design and Analysis
Lecture 14

Allan Borodin

February 11, 2013

Some materials are from Stephen Cook's IIT Mumbai (Bombay) slides.

Announcements and Outline

Announcements

- I have posted the first three questions for Problem Set 2.
  - First two questions relate to flow networks.
  - Third question concerns the poly time reduction of the graph colouring search problem to the graph colouring decision problem.
  - Please start on the problem set and make sure the questions make sense to you.

Today's outline

- We continue complexity theory and NP completeness
- I am going to continue to use some slides from a recent talk by Stephen Cook.

What have we been doing so far in this course

- So far, we have almost entirely been presenting polynomial time algorithms.
- As one exception, we did consider the DP for the knapsack problem.
- Many times we didn't care if the operands (say in interval scheduling) were real numbers or integers.
- We just assumed that we could do basic arithmetic operations and comparisons in one step.
- This was not a problem because we didn't use algorithms that would build up large integers. (Note that if we only use addition, then repeated doubling can only produce a number as large as 2^n in n steps).

And now we take a detour from efficient algorithms

- We have often refered to a problems NP-hardness when we considered computing optimal solutions for a number of optimization problems.
- We alluded to the widely held belief that optimal solutions for such NP-hard problems cannot be computed efficiently (for all inputs).
- This basis for this belief is expressed as the conjecture (sometimes called Cook's Hypothesis) that $P \neq NP$.
- Hence we embarked on approximation algorithms for such problems.
- We now want to define the meaning of this conjecture and the evidence we have for believing in this conjecture.
  - Briefly stated, the main evidence is the extensive number of problems which can be shown to be "equivalent" in the sense that if any one of them can be computed efficiently (i.e. in P) then they all are.
  - And, of course, these are problems that have been studied for many years (decades and in certain cases centuries) without anyone being able to find polynomial time algorithms.

What is NP?

- The class NP (Nondeterministic Polynomial Time)
- NP consists of all search problems for which a solution can be efficiently (i.e. in polynomial time) verified.
- More specifically, a set (or language) L is in the class NP if there is a polynomial time computable $R(x, y)$ such that for all $x \in L$, there is a certificate y such that $|y| \leq poly(|x|)$ and $R(x, y)$.

Examples in NP (besides everything in P)

- Prime Factorization
- Cracking cryptographic protocols.
- Scheduling delivery trucks, airlines, hockey matches, exams, . . .

Claim

All optimization problems considered thus far have a corresponding NP decision problem.

P **versus** NP

- P: Problems for which solutions can be efficiently (i.e. in polynomial time) found.
- NP: Problems for which solutions can be efficiently verified. Alternatively, problems which can be solved in non-deterministic polynomial time and hence the meaning of the N in NP.
- Formally, P and NP refer to decision problems.
- Why polynomial and not some other class of time bounds?

Conjecture

$$\mathrm{P} \neq \mathrm{NP}$$

- Most (or really almost all) computer scientists believe this conjecture. But it seems to be incredibly hard to prove this conjecture. How do we rule out **all** polynomial time algorithms?

Why is proving $\mathrm{P} \neq \mathrm{NP}$ difficult?

- Another reason is that some search problems in NP (such as finding a solution to a set of linear inequalities) turn out to be easy.
- Here is another easy example (generalizing the maximum bipartite matching problem introduced in Lecture 11).

The matching problem for undirected graphs

Given a large group of people, we want to pair them up to work on projects. We know which pairs of people are compatible, and (if possible) we want to put them all in compatible pairs.

- If there are 50 or more people, a brute force approach of trying all possible pairings would take billions of years.
- However in 1965 Jack Edmonds found an efficient algorithm. So this problem is in P.
- How can we identify the hard NP problems?

NP-Complete Problems

- These are the hardest NP problems.
- A problem A is p-reducible to a problem B if an "oracle" (i.e. a subroutine) for B can be used to efficiently solve A.
- If A is p-reducible to B, then any efficient procedure for solving B can be turned into an efficient procedure for A.
- If A is p-reducible to B and B is in P, then A is in P.

Definition

A problem B is NP-complete if B is in NP and every problem A in NP is p-reducible to B.

Theorem

If A is NP*-complete and A is in* P*, then* P $=$ NP.

To show P $=$ NP you just need to find a fast (polynomial-time) algorithm for any one NP-complete problem!!!

- A great many (literally thousands) of problems have been shown to be NP-complete.
- Most scheduling problems (delivery trucks, exams etc) are NP-complete. For example, the JISP and Knapsack problems (when expressed as decision problems are NP-complete.
- The following simple exam scheduling problem is NP-complete:

Example

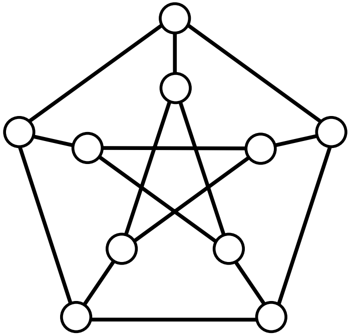
- We need to schedule N examinations, and only three time slots are available.
- We are given a list of exam conflicts: A conflict is a pair of exams that cannot be offered at the same time, because some student needs to take both of them.
- **Problem:** Is there a way of assigning each exam to one of the time slots T_1, T_2, T_3, so that no two conflicting exams are assigned to the same time slot.
- This problem is also known as graph 3-colorability.

Graph 3-Colorability

- A graph is a collection of nodes, with certain pairs of nodes connected by an edge.

Problem

Given a graph, determine whether each node can be colored red, blue, or green, so that the endpoints of each edge have different colours.

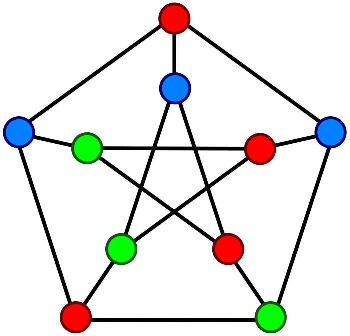


Graph 3-Colorability

- A graph is a collection of nodes, with certain pairs of nodes connected by an edge.

Problem

Given a graph, determine whether each node can be colored red, blue, or green, so that the endpoints of each edge have different colours.



Some more remarks on graph coloring

- The natural graph coloring optimization problem is to color a graph with the fewest number of colors.
- We can phrase it as a search or decision problem by saying that the input is a pair (G, k) and then
  1. The search problem is to find a k-coloring of the graph G if one exists.
  2. The decision problem is to determine whether or not G has a k coloring.
  3. Clearly, solving the optimization problem solves the search problem which in turn solves the decision problem.
  4. Conversely, if we can efficiently solve the decision problem then we can efficiently solve the search and optimization problems.
- Formally it is the graph coloring decision problem which is NP-complete. More precisely, the decision problem for any fixed $k \geq 3$ is NP-complete. However, 2-Colorability is in P.
- But we will often abuse terminology and speak of the search problem or the optimization problem as being NP-complete.

Reducing Graph 3-Colourability to 3SAT

- We are given a graph G with nodes, say $V = \{v_1, v_2, \ldots, v_n\}$
- We are given a list of edges, say $(v_3, v_5), (v_2, v_6), (v_3, v_6), \ldots$
- We need to find a 3CNF formula F which is satisfiable if and only if G can be colored with 3 colors (say red, blue, green).
- We use Boolean VARIABLES
  $r_1, r_2, \ldots, r_n$ (r_i means node i is colored red)
  $b_1, b_2, \ldots, b_n$ (b_i means node i is colored blue)
  $g_1, g_2, \ldots, g_n$ (g_i means node i is colored green)

- Here are the CLAUSES for the formula F:
  - We need one clause for each node:
    $(r_1 \vee b_1 \vee g_1)$ (node 1 gets at least one color)
    $(r_2 \vee b_2 \vee g_2)$ (node 2 gets at least one color)
    ...
    $(r_n \vee b_n \vee g_n)$ (node n gets at least one color)
  - We need 3 clauses for each edge: For the edge (v_3, v_5) we need
    $(\overline{r_3} \vee \overline{r_5})$ (v_3 and v_5 not both red)
    $(\overline{b_3} \vee \overline{b_5})$ (v_3 and v_5 not both blue)
    $(\overline{g_3} \vee \overline{g_5})$ (v_3 and v_5 not both green)

- The size of the formula F is comparable to the size of the graph G.

- **Check:** G is 3-colorable if and only if F is satisfiable.

On the nature of this polynomial time reduction

- If we consider the previous reduction of 3-coloring to 3-SAT, it can be seen as a very simple type of reduction.
- Namely, given an input w to the 3-coloring problem, it is transformed (in polynomial time) to say $h(w)$ such that

$$w \in \{G \mid G \text{ can be 3-colored}\} \text{ iff}$$
$$h(w) \in \{F \mid F \text{ is a satisfiable 3CNF formula}\}.$$

- This was the same kind of polynomial time reduction that showed that bipartite matching is reducible to maximum flows.

Polynomial time transformations

- We say that a language L_1 is polynomial time transformable to L_2 if there exists a polynomial time function h such that

$$w \in L_1 \text{ iff } h(w) \in L_2.$$

- The function h is called a polynomial time transformation.

Polynomial time reductions and transformations

- In practice, when we are reducing one NP problem to another NP problem, it will be a polynomial time transformation.
- We will use the same notation $\leq_p$ to denote a polynomial time reduction and polynomial time transformation.
- As we have observed before if $L_1 \leq_p L_2$ and $L_2 \in$ P, then $L_1 \in$ P.
- The contrapositive says that if $L_1 \leq_p L_2$ and $L_1 \notin$ P, then $L_2 \notin$ P.

$\leq_p$ is transitive

- An important fact that we will need to prove NP completeness of problems is that polynomial time reductions are transitive.
- That is $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ implies $L_1 \leq L_3$.

- One fact that holds for polynomial time transformation is but is believed not to hold for polynomial time reductions is the following:

NP closed under polynomial time transformation

If $L_1 \leq_p L_2$ and $L_2 \in$ NP then $L_1 \in$ NP.