Lecture 24: NP-completeness

Lecture Overview

- Decision vs Search vs Optimization Problems
- P and NP
- Reductions between problems
- NP-Complete Problems

Readings

CLRS 34

Decision Problems vs Search Problems vs Optimization Problems

A *decision problem* asks us to *check if something is true*. E.g. Is there a simple path of length at most some given bound B from a given node s of a graph G to another node t? Possible answers: {'yes', 'no'}.

A *search problem* asks us to *find a solution with certain properties if such a solution exists*. E.g. *Find* a path from a given node s of G to another node t with length at most some given bound B if such a path exists. Possible answers: {paths of length at most B}$\cup${'no path of length at most B exists between s and t'}.

A *optimization problem* asks us to *find among all solutions the one with the best performance in some metric*. E.g. *Find* the shortest path from a given node s of G to another node t if any path exists. Possible answers: {shortest paths from s to t}$\cup${'no path between s and t exists'}.

Clearly, the decision version of the shortest paths problem defined above is not harder than the search version, since an algorithm solving the latter problem can solve the former. The opposite direction is not always clear, since certifying that a solution exists may not always give us that solution. But this is a philosophical discussion beyond this course...

And **what about the optimization versus the search version of the problem?** These are essentially equivalent since we can do binary search on the possible path lengths B to reduce the optimization version to the search version. On the other hand, reducing the search version to the optimization one is straightforward.

We focus on search problems for the remaining of this lecture.

Search problems, a bit more formally

What makes up a search problem? A *search problem* $\mathcal{P}$ is defined by:

- a set $\mathcal{I}_\mathcal{P}$ of valid instances, or inputs to the problem;
- an algorithm $\mathcal{A}_\mathcal{P}$, which, given $x \in \mathcal{I}_\mathcal{P}$ and a proposed solution y to x, checks if y is a valid solution to x, i.e. a solution satisfying the desired properties.

E.g., for the shortest path problem (SPP)

- $\mathcal{I}_{\text{SPP}} := \{$ weighted graph G + pair of vertices s and t + bound $B\}$;
- $\mathcal{A}_{\text{SPP}}$ checks if y is a path on G from s to t whose length does not exceed B.

The Class NP

NP is the class, i.e. the set, of all search problems $\mathcal{P}$ satisfying the following properties

1. for all (valid) instances $x \in \mathcal{I}_\mathcal{P}$, there exists a solution y to x whose description has size polynomial in x, if a solution exists at all;
2. algorithm $\mathcal{A}_\mathcal{P}$ runs in time polynomial in $|x|$ and $|y|$.

NP stands for "non-deterministic polynomial time". The use of non-determinism in the name of NP conveys the meaning that a non-deterministic algorithm (i.e. an algorithm that is allowed to toss random coins in its execution) could guess a random sting y and invoke $\mathcal{A}_\mathcal{P}$ to check if the guessed string y is a valid solution to x. From property 2 above, if there is a solution to x there has to be one of size at most polynomial in x. Hence, the probability of success of the algorithm would be at least inverse exponential. Alas, in general it may be no larger than inverse exponential, which is far from being practical...

The Class P

P is the class of all search problems in NP that can be solved in polynomial time.

The P versus NP Question

Clearly, P $\subseteq$ NP (by definition). But the answer to whether P contains NP is not known yet. Many researchers believe that P $\neq$ NP, i.e. $P \subset NP$, but showing this seems to be beyond present mathematics. In fact, the 'P versus NP' question is in the list of the seven Millennium Prize Problems of the Clay Mathematical Institute. Presently six of the initial seven problems (given in 2000) remain unsolved with a prize of $\$1,000,000$ accompanying each.

The following table discusses a few examples of problems in P and NP that we have encountered in this class.

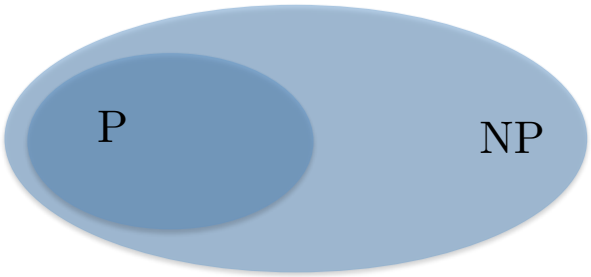

Figure 1: P $\subseteq$ NP

Problem:	in NP?	in P?
short path (i.e. find a path of length at most B between s and t)	yes	yes
long path (i.e. find a path of length at least B between s and t)	yes	unknown
common Subsequence (i.e. find a common subsequence of two strings of length at least ℓ)	yes	yes
vertex cover (i.e. find a set of vertices of size at most k such that every edge is adjacent to the set)	yes	yes in trees, unknown in general graphs
dominating set (i.e. find a set of vertices of size at most k such that every vertex is in or adjacent to the set)	yes	yes in trees, unknown in general graphs

Reductions Between Problems

A *reduction from problem A to problem B* is a construction showing that an algorithm for problem B is sufficient for solving problem A. It consists of two functions f and g. f maps instances of A to instances of B, while g maps solutions of B to solutions of A. These functions need to satisfy the following properties.

If x is an instance of problem A, then $f(x)$ is an instance of problem B such that

- if y is a solution to $f(x)$, then $g(y)$ is a solution to x;
- if there is no solution to $f(x)$, then there is no solution to x either.

See Figure 2 for how to use functions f and g to produce an algorithm for problem A given an algorithm for problem B.

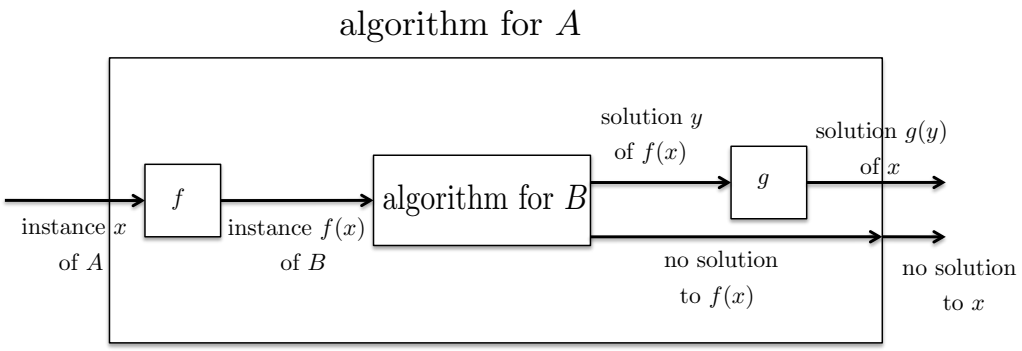

Figure 2: Reduction *from* search problem A *to* search problem B

e.g. Vertex Cover $\longrightarrow$ Dominating Set.

How do we turn an algorithm for DS into an algorithm for VC?

Description of f: Given a graph G, convert every edge of G into a triangle by introducing a dummy vertex per edge, as in Figure 3. Let $G^* := f(G)$.

Description of g: Given a DS y for G^* of size at most k obtain a VC $g(y)$ of G of size at most k as follows: if a dummy vertex is used in y, then use either of its neighboring vertices in $g(y)$; if a non-dummy vertex is used in y, use that vertex also in $g(y)$.

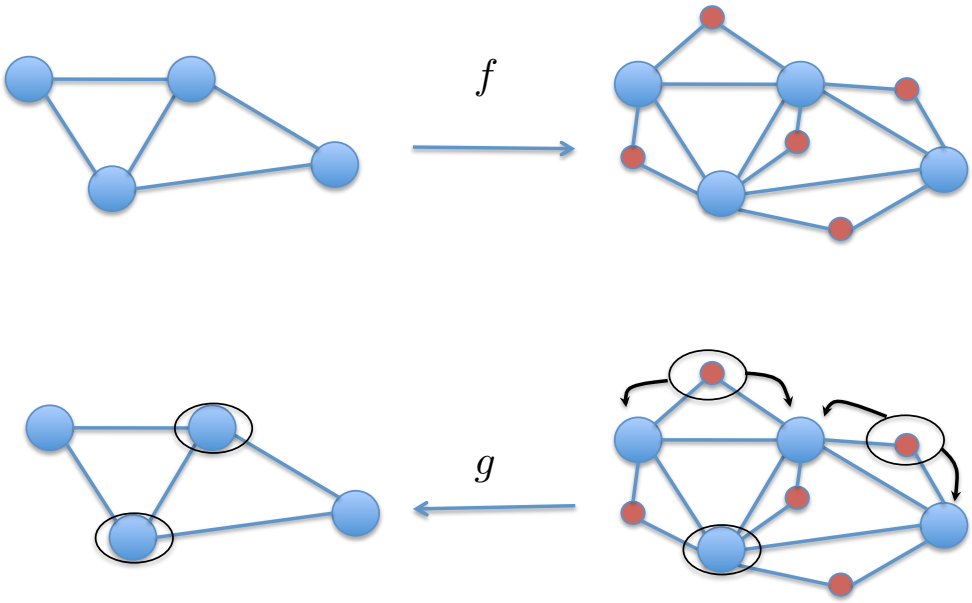

Figure 3: Reduction *from* Vertex Cover *to* Dominating Set

The Class of NP-Complete Problems

A search problem $\mathcal{P}$ is an NP-*complete problem* if

1. $\mathcal{P}$ belongs to NP;
2. there is a reduction from every problem in NP to the problem $\mathcal{P}$.

One of the most important theorems in the development of computer science is the following theorem due to Cook, Karp and Levin.

Theorem[Cook-Karp-Levin]: Vertex Cover is NP-complete.

As a corollary of this theorem it follows that dominating set is also NP-complete (since VC reduces to it), and so are many other problems, e.g., the Knapsack problem, the longest path problem, etc.

The class of NP-complete problems captures the hardest problems inside NP. For many of these problems researchers have been trying unsuccessfully to develop efficient algorithms for decades. This suggests that NP-complete problems are probably not solvable in polynomial time, and most computer scientist share this view (see Figure 4). However, while the P vs NP question remains unanswered, there is no formal proof that NP-complete problems outside of P really exist.

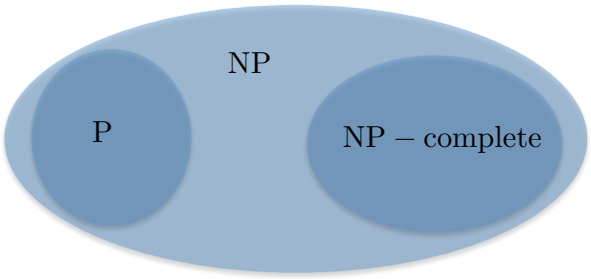

Figure 4: The complexity world, as most computer scientists expect it to be.