Complexity Theory

Jonathan Backer
backer@cs.ubc.ca

Department of Computer Science
University of British Columbia

July 19, 2007

Introduction

Reading:

- "NP-Completeness" 34 CLRS
- "NP-Completeness" 13 GT

Focus was on designing efficient algorithms.

- Upperbounded space and time requirements.

Complexity theory tells us

- only inefficient algorithms are possible, or
- no algorithm is possible.

Asymptotic hierarchy

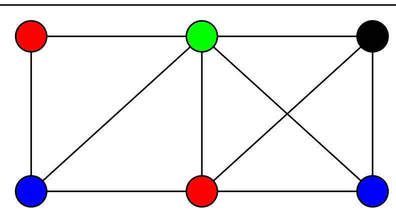
How "hard" is a problem?

- $\Theta(1)$: find the minimum element in a min-heap, hash-table look-up
- $\Theta(\log n)$: binary search, range tree computation, 2-3-4 tree operations
- $\Theta(n)$: order statistics, bucket sort
- $\Theta(n \log n)$: sorting, weighted activity selection, closest pair of points
- $\Theta(n^2)$: longest common subsequence
- $\Theta(n^2 \log n)$: Prim's and Dijkstra's algorithms
- $\Theta(n^3)$: Bellman-Ford algorithm

⋮

Complexity hierarchy

- NP-hard: No efficient (polynomially bounded time) algorithm exists.



Colour vertices with the *least* number of colours so that adjacent vertices have different colours.

⋮

- PSPACE-hard: Space efficient (polynomially bounded) solutions, but takes exponential time.

⋮

- Undecidable: No (complete) algorithm exists

  Will a given program eventually stop on a given input input?

Decision vs. optimization

- Decision problem: The answer is either YES or NO.
- Optimization problem: Find a maximal or minimal solution.

Typically, a poly-time solution to one variant gives a poly-time solution to the other variant.

Graph colouring

Given a graph $G = (V, E)$, a k-colouring of a G is a function $\chi : V \rightarrow \{1, \ldots, k\}$ such that $\{u, v\} \in E$ implies $\chi(u) \neq \chi(v)$.

Decision problem: Is a given graph k-colourable?

Optimization problem: Find the smallest k^* such that a given graph is k^*-colourable.

Poly-time

Definition

A problem is poly-time solvable if some algorithm A solves every instance of the problem in $O(n^k)$, where k is a constant and n is the # of bits used to represent the input.

Note: Depends on the input representation

Problem: Print the letter 'A' m times.

If m is represented in binary, then $n = \lg m$ and size of the output is $O(2^n)$.

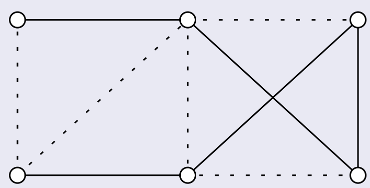
If m is written as a sequence of 1s, where the number of 1s is m (unary), then $n = m$ and the size of the output is $O(n)$.

Problem class: P vs NP

- P: all decision problems that are poly-time.
- NP: all decision problems where a YES can be poly-time verified given an appropriate certificate

Hamiltonian path problem

Given a graph, does some path visit every vertex exactly once?



Finding a such a path is hard. But verifying that a given path (certificate) is Hamiltonian is easy.

P vs. NP (cont'd)

- P $\subseteq$ NP because correct algorithms are certificates.
- Is NP $\subseteq$ P? $1,000,000 if you find out http://www.claymath.org/millennium/P_vs_NP

Satisfiability problem (SAT):

Given a set of boolean variables $x_1, \ldots, x_n$ and a set of clauses $C_1, \ldots, C_m$, where each clause is a disjunction of variables and their complements

e.g.

$$\begin{aligned} C_1 &= x_1 \vee x_2 \vee \overline{x_4} \\ C_2 &= \overline{x_1} \vee x_3 \vee x_4 \\ C_3 &= x_2 \vee \overline{x_3} \vee \overline{x_5} \vee x_7, \end{aligned}$$

can we assign `true`/`false` values to each variable so that every clause is satisfied?

NP-Completeness

SAT is clearly in NP (what's the certificate?). But Cook (from the U of Toronto) showed the following:

Theorem

If SAT is poly-time, every problem in NP is poly-time (i.e. $\mathtt{NP} = \mathtt{P}$).

Definition

Problems with the property of Cook's theorem are called *NP-hard*. If a NP-hard problem is also in NP, it is called *NP-complete*.

We strongly suspect that a *NP*-hard problem has no polynomial time solution.

How do we prove that a problem is *NP*-hard?

Reductions

Definition

We reduce a problem A to another problem B by providing a transformation that

- takes any instance I_A of A and
- returns an instance I_B of problem B
- such that an answer to I_B gives an answer to I_A.

Example

Multiply every value by -1 to reduce finding the maximum to finding the minimum.

e.g. $\{1, 0, -3, 4, 2, \mathbf{6}\} \Rightarrow \{-1, 0, 3, -4, -2, -\mathbf{6}\}$

If the transformation is easy to compute, problem A is easier than problem B because solving B indirectly solves A.

Poly-time reductions

To prove that problem B is NP-hard:

- Find a known NP-hard problem A and
- reduce A to B with a poly-time reduction.

Let us be precise.

1. Pick a known NP-hard decision problem A.
2. Give a transformation T that makes any instance I_A of A an instance I_B of B in time polynomially bounded in the size of I_A
3. The transformation must be such that I_A is YES if and only if I_B is YES.

Then a poly-time solution to B gives a poly-time solution to A via T.

Why should you care?

Because you can

- stop looking for an efficient algorithm and graduate!
- justify heuristic methods that give good answers most of the time.
- justify approximation algorithms that give sub-optimal answers all of the time.
- prove that other problems are NP-hard more easily.

Graph colouring

Theorem

Graph 3-coloring is NP-complete.

Proof

Clearly 3-colouring is in NP because given a colouring we can verify that a colouring is valid (i.e. no two adjacent vertices have the same colour) and only uses three colours in poly-time.

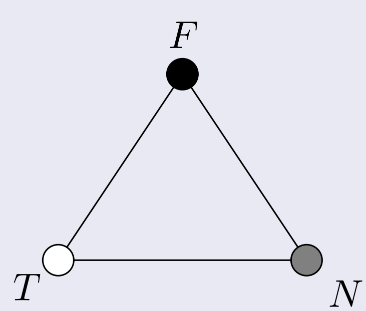
To prove that the problem is NP-hard, we reduce SAT to graph 3-colouring. The reduction uses colours to represent truth assignments and groups of vertices to represent clauses and variables.

We start with a triangle of labelled vertices T,F,N. The colour of T will represent true, the colour F will represent false, and the colour of N is neutral.

Truth assignments

Proof (cont'd)

We can swap colours of a valid colouring to get another valid colouring. So assume without loss of generality that T is white, F is black, and N is gray in every colouring.



Before we describe the construction in detail, let's analyse one subgraph (widget) that we use over and over again.

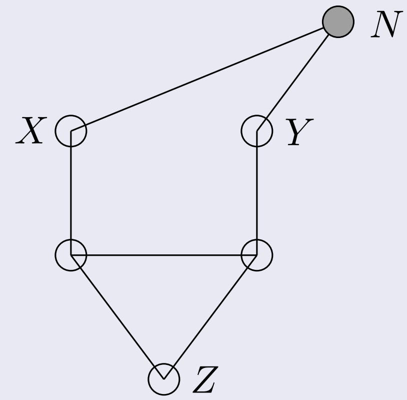
Widget colouring

Lemma

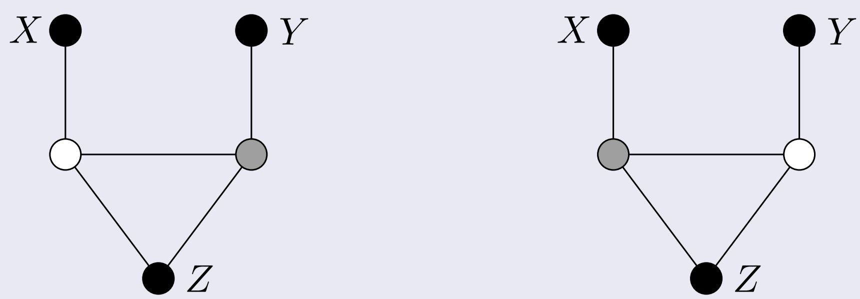
Note that X, Y are either black or white.

- *If Z is white, at least one of X, Y is white.*
- *If both X, Y are black, then Z is black.*

These statements are equivalent.



Proof.

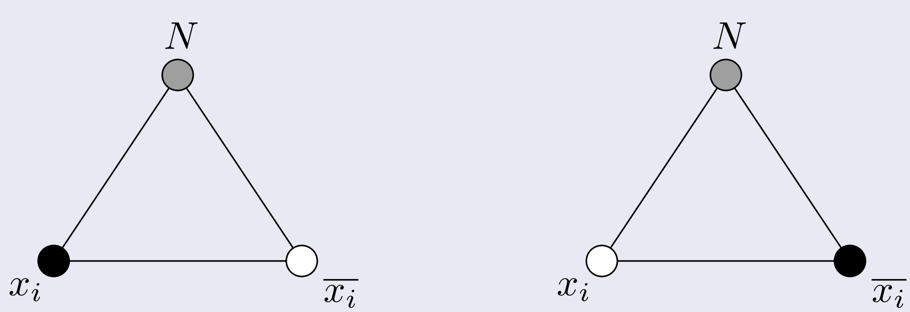


□

Variable triangles

Proof of NP-hardness (cont'd)

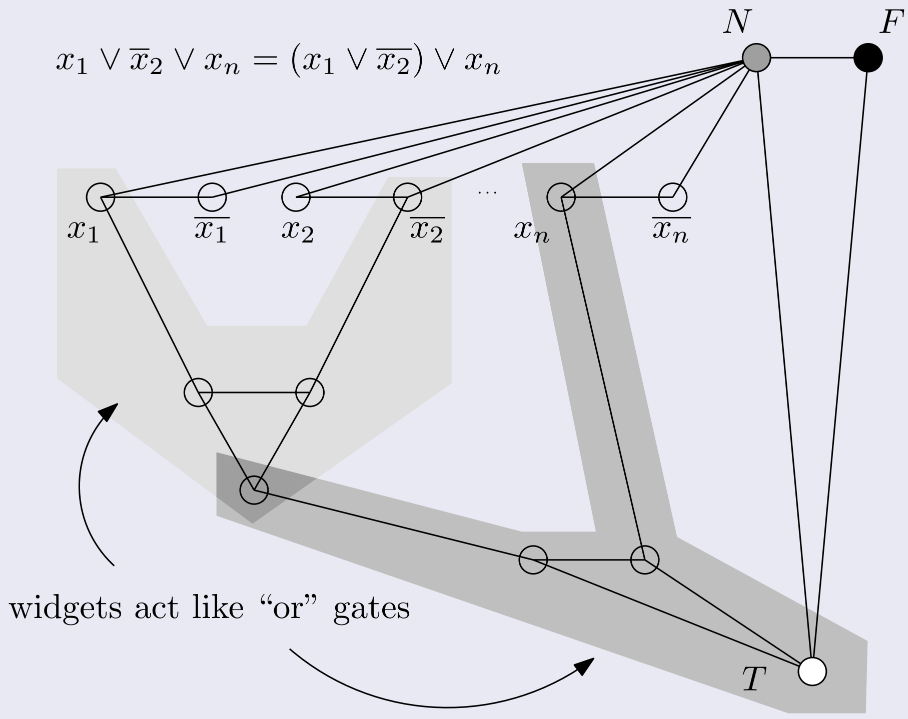
For each variable x_i we introduce two vertices labelled x_i and $\overline{x_i}$ that form a triangle with N. Colouring x_i white corresponds to assigning the variable x_i the value of true; colouring it black corresponds to assigning the variable the value of false.



The triangle implies that x_i is coloured white if and only if $\overline{x_i}$ is coloured black.

Clause construction

Proof (cont'd)



Analysis

- The number of vertices and edges is linear in # of variables + # of clauses.
- The graph structure closely follows SAT instance, so it's a polynomial time construction.
- A satisfying truth-value-assignment to the SAT instance gives a colouring.
- That a colouring gives a truth-value-assignment is best seen with an example...