COSC 341
Theory of Computing
Lecture 17
Polynomial reductions, and $\mathbf{NP}$-hardness

Stephen Cranefield

stephen.cranefield@otago.ac.nz

Lecture slides (mostly) by Michael Albert

Keywords: Polynomial reduction, graph theory, $\mathbf{NP}$-hard, $\mathbf{NP}$-complete

Time complexity revision

- The time complexity of a Turing machine is the maximum amount of time (in 'ticks': transitions made) that it will run for on input of size n.
- We assume we are working with TMs that always halt.
- A decision problem is in complexity class $\mathbf{P}$ if there is *some* deterministic TM that decides the problem and has time-complexity bounded by a polynomial in n.
- (New information) Basic arithmetic operations are in $\mathbf{P}$ (if represented in base 2 or higher).
- A decision problem is in $\mathbf{NP}$ if *some* non-deterministic TM can decide it and has time-complexity bounded by a polynomial in n.
- Example $\mathbf{NP}$ problem: FAIR-DIVISION
  - Given a sequence $a_1, a_2, \ldots, a_n$ of positive integers, does there exist a partition of it into two sequences of equal sum.
  - A genie can write a sequence of binary hints $h_1, h_2 \ldots h_n$ indicating which partition each a_i is in. This can be verified in polynomial time.

Reductions revisited

- A reduction from one decision problem, say A, to another, B is a procedure that converts instances of A to instances of B that preserves their status.
- Strictly speaking this is called a many-one reduction or Karp-reduction.
- The idea is that the reduction should be "easy". If so, we can infer:
  - If solving B is easy, then so is solving A.
  - If solving A is hard, then so is solving B.
- We're usually interested in proving that problems are hard so we need:
  - A source problem that's known to be hard.
  - A reduction from it to our problem.

Polynomial-time reductions

A polynomial-time reduction of A to B is a deterministic algorithm, i.e. TM, for converting instances of A to instances of B such that:

- the time required for the conversion is bounded by a polynomial (in the input size), and
- all positive instances of A are converted to positive instances of B, and
- all negative instances of A are converted to negative instances of B.

Theorem

If B is in $\mathbf{P}$ (respectively, $\mathbf{NP}$) and A has a polynomial-time reduction to B then A is in $\mathbf{P}$ (respectively, $\mathbf{NP}$).

There's a little more to this than meets the eye, which is one of the reasons why $\mathbf{P}$ is considered a robust definition of problems that have efficient solutions.

Graph theory overview and example problems

- See Notes 15 and the lecture recording.
- We don't need any more knowledge of graphs than is covered in COSC201.

$\mathbf{NP}$-hard

- Suppose we could find some problem ReallyHard which had the property that for every problem, Prob in $\mathbf{NP}$ there was a polynomial-time reduction from Prob to ReallyHard.
- That says ReallyHard is a really hard problem, since solving it would allow us to solve any problem in $\mathbf{NP}$ with additional polyomial overhead.
- Such problems are called $\mathbf{NP}$-hard.
- If, in addition, they happen to belong to $\mathbf{NP}$, then they're $\mathbf{NP}$-complete.

Question: Do such problems exist?

Answer: **YES**! And much more – *many natural problems in optimisation and search are* $\mathbf{NP}$*-complete*. For example, Fair-Division, Hamilton-cycle, and 3-Colouring are such problems.

Example reduction: INDEPENDENT-SET to CLIQUE

Given an instance (G, k) of INDEPENDENT-SET:

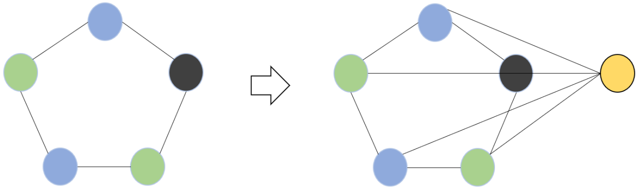
1. Construct a new graph (the complement of G), G^c where v and w are adjacent in G^c if and only if they were not adjacent in G.
2. Any independent set in G is a clique in G^c and vice versa, so the status of (G, k) for INDEPENDENT-SET is the same as that of (G^c, k) for CLIQUE.
3. Algorithm:

```
// copy nodes ...
// print edges
for v from 1 to n
  for w from v+1 to n
    if {v,w} is an edge of G continue
    print(v + w + '#')
```

Example reduction: 4-COLOURING to 5-COLOURING

Given an instance G of 4-COLOURING:

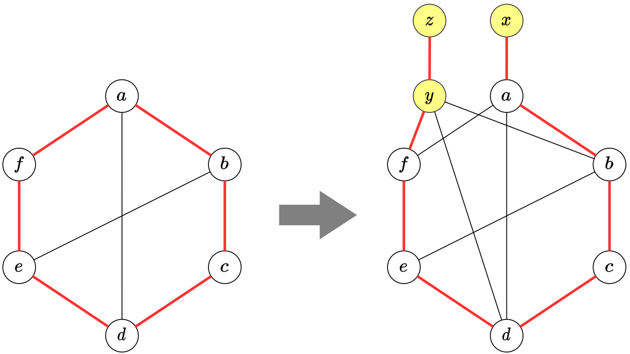
- Construct a new graph G^{+1} which has one new vertex that is adjacent to all the vertices of G.



- If G is positive for 4-COLOURING then G^{+1} is positive for 5-COLOURING (4-colour G and use the 5th colour for the new vertex).
- If G is negative for 4-COLOURING then G^{+1} is negative for 5-COLOURING (because a 5-colouring of G^{+1} induces a 4-colouring of G).

Example reduction: HAMILTON-CYCLE to HAMILTON-PATH

See Notes 15. Discussed in Tutorial 17.



- Choose any node (a in the diagram) as the starting node for the path.
- Add the yellow nodes (x, y and z). x is adjacent to a, y is adjacent to the neighbours of a, and z is adjacent to y.
- A Hamilton path exists in the new graph if and only if there is a Hamilton cycle in the original graph.