# CS3052 — Computational Complexity

## Practical 2 : PolyReduction

Steve Linton
sl4@st-andrews.ac.uk

Tom Kelsey
twk@st-andrews.ac.uk

***Due:*** 8 May 9pm, subject to any later changes. MMS is definitive.

***Key Points***

This is ***Practical 2***, in which we ask you to implement a number of polynomial reductions between NP Complete problems

***Due date***: 8 May 9pm – please watch MMS

***Credit***: 40% of the module.

***Submission***: Upload a zipfile with your report in PDF and any supplementary information (code, data, spreadsheets, etc.) to MMS.

## *Purpose*

This practical will help practice and develop your skills in:

- understanding polynomial-time reductions as used to show problems to be NP-Complete
- Programming with basic objects of discrete mathematics, such as graphs
- analysing algorithms.

## *Overview and Background*

A decision problem $P$ is called ***NP-Complete*** if:

- $P \in NP$: that is $P$ can be solved in non-deterministic polynomial time (or equivalently, a solution to an instance of $P$, accompanied by a suitable hint, can be checked in polynomial time) and
- any decision problem $Q \in NP$ can be polytime reduced to $P$

This means that if you had a polynomial time solution to $P$ you could have one for *every* problem in NP.
Three well known NP-Complete problems are:

- SAT – Boolean Satisfiabiloty – given a Boolean formula in Conjunctive Normal Form, is there an assignment of truth values to the variables which makes the formula as a whole evaluate to true
- 3SAT – Restricted Boolean Satisfiabiloty – SAT with the additional restriction that no clause may contain more than three variables.
- Graph Colouring – Given an undirected graph and a positive integer $k$, is it possible to assign a "colour" from 1 to $k$ to each vertex of the graph, such that any two vertices joined by an edge have different colours.

The main part of proving that a problem $P$ is NP-Complete is usually finding a reduction from a known NP-Complete problem to $P$. In this practical you will implement a cycle of reductions among these problems.

## *Basic Practical Specification*

### Polynomial Reductions

You are required to implement polynomial reductions as follows:

1. SAT to 3SAT
2. 3SAT to graph colouring
3. graph colouring to SAT

Each of these should be a separate program, ideally runnable from the command-line. The input and output formats should be documented and consistent so that output from one program can be piped into another (where appropriate) without human intervention.

Your reductions must convert positive instances of each problem into positive instances of the next and negative instances of each problem into negative instances of the next.

Your report should explain the reductions that you are using and give a clear argument why each is (a) correct and (b) polynomial time. Sources for the reduction algorithms should be cited (the implementations should be your own).

You should test your reductions for correctness and explain in your report what testing you did and why it shows correctness. You may want to implement (or download) a simple brute-force solver for one or all of these problems as part of your testing.

Marks of up to 11, 14 and 17 will be available for good quality implementations of 1, 2 or 3 reductions respectively, accompanied by a clear report.

### Extensions

For Marks up to 20 you should complete one of the following:

- Implement an additional reduction to a known or interesting problem which is more complex than these. For instance genetral graph colouring to graph 3-colouring or 3SAT to vertex cover.
- Extend your reductions to recover solutions to the non-decision versions of the problem, so that, for instance, you can recover the truth values of the boolean variables for a 3SAT instance from a colouring of the graph produced in the second reduction above.

### Additional Details

1. You may use any programming language, provided that I can find a specification of it so that I can read your programs
2. I would strongly prefer that the resulting programs run from the UNIX command line controlled by command-line parameters in the usual way. Interactive programs are strongly discouraged, as they are hard to test effectively. Under the circumstances, however, I am not insisting on this.
3. Similarly, I would prefer it if your programs could be compiled using `make` in the normal way, but I do not insist on it.
4. Whatever you do, please include clear instructions on how to build and run your programs. If you were not able to test your programs on the school systems, then please include details of what system (operating system, compiler, etc.) you did use.

## *Submission*

Your submission should include a report, ideally as a PDF, and code.

Make a ***zip archive*** of all of the above, and submit via MMS by the deadline.

## *Marking*

We remind you that it is not enough for your programs to be correct, you have to convince the markers that they are correct.

We are looking for:

- good, understandable code, tested and commented properly, with important design decisions explained in the report;
- good insight into complexity analysis.

The standard mark descriptors in the School Student Handbook will apply:
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

## *Lateness*

The standard penalty for late submission applies, except where waived, (Scheme B: 1 mark per 8 hour period, or part thereof):
`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

## *Good Academic Practice*

The University policy on Good Academic Practice applies:
`https://www.st-andrews.ac.uk/students/rules/academicpractice/`