COMP26120
Academic Session: 2019-20

Lab Exercise 8: Exploring NP-Completeness with Graph Colouring

Duration: 1 lab session.

This lab requires relatively little coding and quite a lot of reading and thinking.
It is also a *formative* lab, this means that you can submit it for marking and receive feedback but the mark is not used in your final grade. Note that, as for all labs, the concepts in the lab are examinable.

Learning Objectives

By the end of this lab you should be able to:

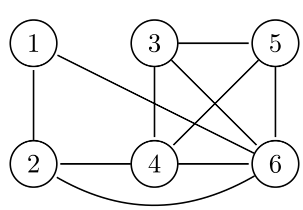
- Describe the graph colouring problem
- Explain how we know that graph colouring is NP-complete
- Define what a clique is and how it is related to graph colouring/NP-completeness

Graph Colouring

Graph colouring is the problem of trying to *colour* an undirected graph with a given set of colours such that two adjacent nodes do not have the same colour.

Given an undirected graph $G = \langle V, E \rangle$ and a set of colours C, a graph colouring is a function $\rho : V \to C$ that assigns a colour to each vertex of G. A graph $G = \langle V, E \rangle$ is *well-coloured* by ρ if for ever edge $(b_1, b_2) \in E$ we have $\rho(b_1) \neq \rho(b_2)$. Let C_i be a set of colours of size i e.g. $C_2 = \{c_1, c_2\}$. A graph $G = \langle V, E \rangle$ is k-colourable if there exists a $\rho : V \to C_k$ such that G is well-coloured by ρ.

For example, the following graph is not 3-colourable but is 4-colourable. Can you convince yourself of this?



Why do we care about graph colourability? The most well-known example of a graph colourability problem is register allocation in compilation. During compilation it is necessary to assign program variables (which take the role of vertices) to a finite set of machine registers (which take the role of colours) Two program variables cannot be stored in the same register if they are *live* in the program at the same time. There is a program analysis called live variable analysis which can identify which variables are live concurrently. If we define an edge between each pair of program variables (vertices) that are live at the same time and the resulting graph is k-colourable (for k registers) then we can fit those variables into the registers; furthermore, the graph colouring tells us exactly how to do so.

In this lab you will demonstrate that graph colouring is NP-complete by first reducing the problem to a known problem in NP (boolean formula satisfiability) and then writing a polynomial-time algorithm for showing that a graph is well-coloured. You will also briefly explore the graph clique problem.

Minisat

In this lab you will make use of a SAT solver called Minisat. A SAT solver is a tool that solves the boolean satisfiability problem. If you took COMP21111 (Logic and Modelling) then you should be very familiar with these ideas. If you did not then don't worry - you met the notion of boolean logic and satisfiability in COMP11120 (Mathematical Techniques for Computer Science) and the concepts are straightforward.

Briefly, Minisat works on problems in Conjunctive Normal Form. A literal is a boolean variable or its negation. A clause is of the form $l_1 \vee \ldots \vee l_n$ where l_i are literals and $n > 0$. A problem is a set of clauses interpreted as a conjunction. The SAT solver will attempt to build a *satisfying assignment* that sets each of the boolean variables to 0 or 1 to satisfy the problem. A literal b is satisfied if b is assigned to 1 and a literal $\neg b$ is satisfied if b is assigned to 0. A clause is satisfied if *at least one* of its literals is. A problem if satisfied if *all* of its disjuncts are. For example, given the boolean variables b_1, b_2, b_3, b_4, the following problem

$$\begin{array}{c} b_1 \vee b_2 \\ b_3 \vee b_4 \\ \neg b_1 \vee \neg b_2 \\ \neg b_3 \vee \neg b_4 \\ \neg b_1 \vee \neg b_3 \\ \neg b_2 \vee \neg b_4 \end{array}$$

has a satisfying assignment $b_1 = 1, b_2 = 0, b_3 = 0, b_4 = 1$. In fact, you will see later that the above could be an encoding for the 2-colourability of the graph $\langle \{1, 2\}, \{(1, 2)\} \rangle$.

If you didn't understand this then don't worry too much. We have provided two functions that you can use in your code:

- `atLeastOneOfThese` takes an array of variables and introduces the constraint that at least one of those variables needs to be true e.g. $b_1 \vee \ldots \vee b_n$
- `notBothOfThese` takes two variables and introduces the constraint that both variables are not allowed to be true e.g. $\neg(b_1 \wedge b_2)$ or $\neg b_1 \vee \neg b_2$.

Your job will be to translate the graph colourability problem into constraints of this form (that's all you need).

We have provided two pre-compiled libraries `libminisat.so` and `libminisat-c.so` that contain Minisat and its C bindings respectively. We have also provided the header file `minisat.h` and the appropriate command in the `makefile` to link these all together. However, if they don't work on your machine (they should work on the School machine, let me know if they do not) then you can rebuild them by checking out the repositories at https://github.com/agurfinkel/minisat and https://github.com/niklasso/minisat-c-bindings respectively and following the instructions.

The file `graph_colouring.c` uses these libraries to create a `minisat_solver` object and add clauses to it. You can hack this to create different kinds of constraints if you want but the TAs will not be able to support you in this.

As a final note. We know that the boolean satisfiability problem is NP-complete (it was in your lectures). However, it has been shown that for many real-world cases of the problem we can achieve much better complexity. This is an interesting observation; just because something is NP-complete in general it does not mean that the specific problem you are trying to solve has this worst-case form (but it might).

Description

This lab is divided into three small parts. Each part should be relatively quick to complete.

Part 1: Translation to SAT

Your first task is to complete the translation parts of the program `graph_colouring.c`. You should complete three functions:

- `getVar` should translate a vertex index and a colour into an integer. Your function needs to be a bijection and you need the maximum number to correspond to `getVar(graph->MaxSize,colours)`. In your code you will use `getVar(n,c)` to get the variable that is true when vertex n has colour c and false otherwise. You will need to enforce some constraints over these variables so that the interpretation makes sense (see below).
- `addAdjacencyConstraints` should add constraints indicating that two adjacent vertices cannot have the same colours. For example, if b_{12} is true if vertex 1 has colour 2 and b_{22} is true if vertex 2 has colour 2 then we want to ensure that $\neg(b_{12} \wedge b_{22})$ e.g. they are different.
- `addColouringConstraint` should add constraints indicating that a vertex should have exactly one colour. This comes in two parts. Firstly, at least one of the variables giving a colour to a vertex should be true e.g. if there are three colours then $b_{11} \vee b_{12} \vee b_{13}$ would mean that vertex 1 is one of those colours. Secondly, we want to make sure a vertex does not have more than one colour e.g. b_{11} and b_{12} should not both be true.

We suggest reading through the rest of `main` to see what it is doing. You have been provided with code for manipulating graphs in `graph.h` and `graph_functions.c`.

You can compile this by running `make part1` - note that it gives you the instruction to run the command

```
export LD_LIBRARY_PATH=D_LIBRARY_PATH:lib
```

this is necessary to make the dynamic library for minisat visible when you run it.

The file `example.gx` contains the graph given in the introduction to this lab manual, check that

```
./graph_colouring example.gx 3
```

does not find a colouring whilst

```
./graph_colouring example.gx 4
```

does. Now write some extra test graphs to check that your code works. Create graph `five.gx` that is 5-colourable but not 4-colourable.

Part 2: Polynomial-time Certificate

You should now complete the function `checkColouring` that checks that the graph is *well-coloured* by the given colouring. This should be polynomial-time (in the number of edges). Find a graph that demonstrates the difference in running time between finding the colouring and checking that it is a colouring. Save this graph in *demo.gx* (the marking system will run with a time limit of 10s).

Part 3: A lower-bound with Cliques

Consider the more general setting of trying to find the smallest k such that a graph is k-colourable. In this part you should write your answer in a file called `cliques.txt`.

Find out what a clique is and write down the definition. Explain how knowing that there is a clique of size m could help with the task of finding the smallest k such that a graph is k-colourable.

Find out what the *maximum clique problem* is and write down a definition. What is its complexity? How is this problem related to the problem of graph colourability? Write down your answer.

Submission and Marking Scheme

You should submit *graph.h*, *graph_functions.c*, *graph_ colouring.c*, *five.gx*, *demo.gx*, and *clique.txt*.

Part 1. Is the translation from graph colouring to SAT problem implemented completely and correctly?	
All three functions are correctly implemented. A quick test is that the example.gx graph should not be 3-colourable but should be 4-colourable. The student should also have some other tests to show that their code works correctly.	(2)
As above but there is a small mistake	(1.8)
A good attempt has been made but the solution does not work in all cases e.g. a necessary constraint is wrong or has been missed.	(1)
An attempt has been made but it is far from a correct solution.	(0.5)
No attempt has been made	(0)

Part 1. Does the student understand how the translation works and what the implications of the translation are? This includes finding a graph five.gx that is 5-colourable but not 4-colourable.	
The student can clearly explain how their code works and how the translation works more generally. The student can explain why their submitted five.gx graph is 5-colourable but not 4-colourable. The student can explain what the existence of the translation means given that the SAT problem is NP-hard and they can explain what property the translation must have for the associated argument to work.	(3)
As above but some of the explanations contained small gaps in understanding that needed filling in.	(2.5)
As above but some of the explanations contained large gaps in understanding that needed filling in.	(2)
The code can be explained but the student cannot attempt to explain what the implications of the translation are i.e. they cannot relate their solution to the complexity of the graph colouring problem.	(1)
No attempt has been made	(0)

Part 2. Has a function for checking that the graph colouring is correct been implemented comletely and correctly?	
The function has been implemented completely and correctly - it gives the expected result for some tests on example.gx, five.gx and others provided by the student.	(1)
An attempt has been made but it is not completely correct.	(0.5)
No attempt has been made	(0)

Part 2. Does the student understand the implications of the checking function with respect to the complexity of the graph colouring problem?	
The student can explain how their function works and what its complexity is. They can then explain what this means with respect to the complexity of the graph colouring problem. They have provided a demo.gx graph that demonstrates the difference in complexity between solving the graph colouring problem and checking the solution.	(3)
As above but some of the explanations contained small gaps in understanding that needed filling in.	(2.5)
As above but some of the explanations contained large gaps in understanding that needed filling in.	(2)
They can explain their checking funtion but cannot discuss the implications for the complexity of the graph colouring problem.	(1)
No attempt has been made	(0)

Part 3. Has the student written a clear definition of what a clique is and a good discussion of their relevance to the graph colouring problem?	
The student provides a clear definition of what a clique is and what the maximum clique problem is. They can discuss how the notion of a clique is related to graph colourability and if this has any implications to the complexity of that problem.	(1)
There are reasonable definitions for what a clique is but no further dicussion or that discussion is not clear enough.	(0.5)
No attempt has been made	(0)