CS 161: Fundamentals of Artificial Intelligence

Spring 2025 – Assignment 4 – Due 11:59pm, Tuesday, April 29

In this assignment you will write a Python program that converts graph coloring problems into SAT problems and use a SAT solver to solve them. The graph coloring problem concerns if there exists an assignment of colors to nodes of a graph such that no two adjacent nodes are of the same color. Your task consists of two steps:

1. (70 pts) Convert a graph coloring instance into a Boolean formula Δ in *Conjunctive Normal Form* (CNF).
2. (30 pts) Decide if Δ is satisfiable by invoking the RSat solver (http://reasoning.cs.ucla.edu/rsat/).

We have broken the task into multiple functions and provided you with some basic code for file I/O and for gluing all the functions together (see **hw4_skeleton.py**).

1 CNF

A Boolean formula Δ is in CNF if it is a conjunction of *clauses*, where a clause is a disjunction of literals (a literal is a variable or its negation).

Example 1 *The following formula Δ is a CNF defined over three Boolean variables X, Y, Z.*

$$\Delta = (X \vee \neg Y \vee Z) \wedge \neg X \wedge (\neg Y \vee \neg Z)$$

It has three clauses: $X \vee \neg Y \vee Z, \neg X, \neg Y \vee \neg Z$ and five literals: $X, \neg X, \neg Y, Z, \neg Z$.

In this assignment, each Boolean variable is represented by a positive integer (variable index). As a result, a positive integer is used for a positive literal and a negative integer is used for a negative literal. A clause is simply a list containing the literals of the clause. A CNF formula is then simply a list of clauses. For example, if variable X has index 1, variable Y has index 2, and variable Z has index 3, then the clause $X \vee \neg Y \vee Z$ is represented as a list $[1, -2, 3]$, and the above CNF Δ is then represented as a list of list $[[1, -2, 3], [-1], [-2, -3]]$. Of course, the order of clauses and literals in each clause does not matter.

2 Convert Graph Coloring to SAT

Consider a graph G and k possible colors (whose nodes are labeled with their node indices as below). We next show the procedures to construct a CNF from a graph coloring problem.

1. For each node n and for each color c, create a Boolean variable to represent whether node n is assigned color c. You will write a function `node2var(n, c, k)`. This function should return the index of the Boolean variable that represents the condition that node n is assigned color c (with k colors being considered). Use the following conversion convention:

$$\text{variable index} = (n-1) \cdot k + c.$$

2. For each node n, add a clause to represent the constraint that a node is assigned *at least* one color whose color index comes from the set $\{1, 2, \ldots, k\}$. You will write a function `at_least_one_color(n, k)` that takes a node index and returns this clause.

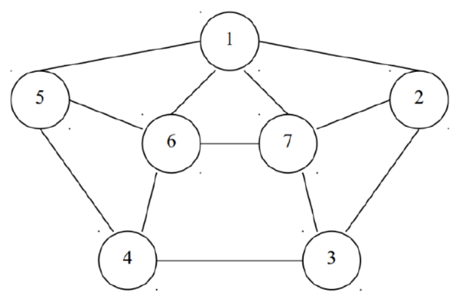

Figure 1: Example graph in graph1.txt

3. For each node n, add a list of clauses to represent the constraint that a node is assigned *at most* one color whose color index comes from the set $\{1, 2, \ldots, k\}$. You will write a function `at_most_one_color(n, k)` that takes a node index and returns this list of clauses. You will also write a function `generate_node_clauses(n, k)`. This function should return a list of clauses that constrain node n to be colored with *exactly* one color from the set $\{1, 2, \ldots, k\}$.

4. For each edge (m, n) in the graph, add a list of clauses to represent the constraint that prohibits two adjacent nodes m and n to be assigned the same color from the set $\{1, 2, \ldots, k\}$. You will write a function `generate_edge_clauses(e, k)`. An (undirected) edge e is simply a tuple of two node indices m, n. This function takes an edge `e` and returns this list of clauses.

After finishing all the above parts, you should be able to convert a graph coloring problem into a SAT problem. To do so, call

```
graph_coloring_to_sat(graph_fl, sat_fl, k)
```

Here, `graph_fl` is the filename of the input graph file (e.g. "graph1.txt"). `sat_fl` is the name of the output file you want the program to write to (e.g. "graph1.cnf"). k is the number of possible colors being considered in the problem. We provide two example graphs in *graph1.txt* and *graph2.txt.* FYI, the graph file has the following format:

- The first line contains 2 numbers. The first one is the number of vertices in the graph; the second one is the number of edges.
- Each subsequent line describes an edge. An edge is represented by two numbers—the node indices of the two nodes linked by the edge.

Now that you have the conversion program working, you can convert the graph coloring problem of the graph provided in graph1.txt with $k = 3$ and $k = 4$ colors into CNFs respectively. Save the generated CNFs with filenames **graph1_3.cnf** and **graph1_4.cnf** respectively and submit them together with your code.

3 Solve SAT with RSat

We show previouly how to convert a graph coloring problem into a SAT instance. The generated CNFs are saved in .cnf files. We can now solve the SAT instance with RSat solver.

Download the RSat SAT solver from (http://reasoning.cs.ucla.edu/rsat/). Use version **2.01**. Read the manual carefully. If you cannot run Rsat on your machine (especially for MAC users), you can run the Linux executable file of Rsat on SEASnet. It is your responsibility to get Rsat running before the deadline. Please check if you can run Rsat on your machine, and apply for a SEASnet account if not as soon as possible. More information about SEASnet and its Linux servers can be found here: `https://www.seasnet.ucla.edu/lnxsrv/` and `https://www.seasnet.ucla.edu/student-account-info/`.

Use RSat to solve the SAT instance obtained above (in .cnf files) and answer the following questions:

1. Consider the CNF generated from graph 1 with $k = 3$ colors (graph1_3.cnf). Is this SAT instance satisfiable?
2. Consider the CNF generated from graph 1 with $k = 4$ colors (graph1_4.cnf). Is this SAT instance satisfiable?
3. What do the answers of these two SAT instances tell you about the graph coloring problem of the above graph? Can you give a solution (a coloring assignment) to the graph coloring problem of graph 1 based on the results of RSat? If you can, please specify the color assignment.
4. Use a similar approach to solve the graph coloring problem of graph 2 in graph2.txt. What is the minimum number of colors required to properly color this graph? You don't need to specify the color assignment for graph 2.

Write your answers to the above questions in **report.txt**. The answer should be clear and concise.

Submission

- Submit all solution files independently (not as a **zip** file and not in a **folder**) **on Gradescope**. The naming of files must follow the instructions strictly. Failure to follow the instructions could result in a penalty to your grade.
  - Submit your Python program in a file **named hw4.py**.
  - Submit two files **graph1_3.cnf** and **graph1_4.cnf** that contains the CNF generated from graph1.txt with $k = 3$ and $k = 4$ colors respectively.
  - Submit a report named **report.txt** that contains your solution to the four questions in Section 3.
- Your programs should be written in good style. In Python, a comment is any character following a hash character (#) on a line. Provide an overall comment explaining your solutions. Furthermore, every function should have a header comment explaining precisely what its arguments are, and what value it returns in terms of its arguments. In addition, you should use meaningful variable names.
- The physical layout of the code on the page is very important for making python programs readable. Make sure that you use blank lines between functions and indent properly. Programming style will be a consideration in grading the assignment.
- You are restricted to using the python built-in functions. All other packages made by `import` (like `numpy`, `queue`, and `collections`) are forbidden.
- You may assume that all input to your functions is legal; i.e. you do not need to validate inputs.
- Do not write any additional helper functions for your code unless this is explicitly allowed. Test functions are OK.
- Your function declarations should look **exactly as specified** in this assignment. Make sure the functions are spelled correctly, take the correct number of arguments, and those arguments are in the correct order.

- Even if you are not able to implement working versions of these functions, please **include a correct skeleton** of each. Some of these assignments are auto graded and having missing functions is problematic.
- By submitting this homework, you agree to the honor code stated in HW1.