CS5015 Practical: Searching for Maximum Cliques

Ruth Hoffmann <cs5015.staff@st-andrews.ac.uk>

Due date: 17th November, 21:00
60% of the coursework component of the module (36% of the overall module assessment)
MMS shows practical weighting and definitive date and time for deadlines

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturers regarding any queries well in advance of the deadline.

Goal

Be able to identify the impact of an heuristic on an algorithm.

Background

The maximum clique problem asks, when given a graph $G = (V, E)$ with V vertices and E edges, then what is the largest subset of V that is a clique in G? A clique is a set of vertices which are all adjacent to each other, i.e. each distinct pair of vertices in a clique are adjacent. For example, Figure 1 shows a clique of size 4 (highlighted in red) inside a larger graph. There could be more of the same size, but there are none with more vertices.

This practical will explore how to improve a search algorithm which looks for a maximum clique in a graph.

The Assignment

You are provided code and graph instances. The code consists of a parser (`parser.py`) which reads in a file in the `DIMACS` folder and returns a graph in a usable format.

The `DIMACS` folder contains graph instances which are provided for this practical. The source of the files is the DIMACS benchmark set found on `https://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`. I have provided you with a subset of this benchmark set. See Hints for more information.

The code in `search.py` takes the graph from `parser.py` and then finds the maximum clique using some search algorithm with an inefficient heuristic.

The code provided to you has been generated using Claude Sonnet 4.5. The main (slightly redacted) prompt was

```
Write an inefficient <search> algorithm for the maximum clique problem.  The input
                        graph is created by parser.py.
```

It was then followed with some additional queries to fix how `parser.py` returns the graph and how `search.py` handles it. The comments were also generated by Claude and have not been checked for correctness.

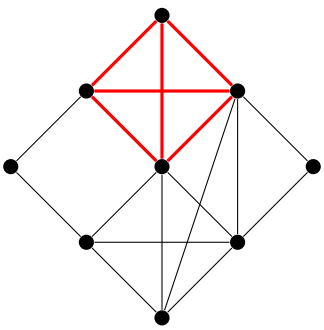

Figure 1: Graph with a maximum clique highlighted.

The search algorithm has been redacted as it is part of your assignment to identify which algorithm is implemented. Note that the provided code is purposefully inefficient and thus will take a long time for some of the provided benchmark instances.

Below are the steps that you have to implement.

1. First identify which type of search algorithm is provided in `search.py`. Explain your answer.
2. Extend/augment the code to be more efficient. Copy the code in `search.py` to a new file and extend it. Do *not* change the underlying search algorithm structure (i.e. change the search type). You *can* improve parts of the algorithm. Make it clear in the code where and what your changes are.
3. Explain your changes to the search code.
4. Run an analysis of the original algorithm against your improved version. Use the provided instances in the `DIMACS` folder as your test bed. You can create graphs using any tools of your choosing (Excel is perfectly reasonable). You may extend the set of instances with the rest of the benchmark set if you want to, but beware that some of them are difficult to solve.
5. Show the results of the original code vs your code. Explain by how much and how the improvements are being exhibited.
6. Provide README or script to run your analysis. Note that the instructions have to be clear to be able for anyone to re-compute your results.
7. Write a report, see the next section for more information.

Report

In your report you need to clearly and succinctly explain all of the above steps. The report should address the following:

- What is the algorithm that has been provided?
- What is the the improvement that you added to the algorithm?
- How are you expecting the improvement to impact the algorithm?
- An analysis with evidence of how the improvement impacted the algorithm. Including explanations of any outliers.
- Any testing which was done to confirm that the improved algorithm still works correctly.

Please note that the aim of the report is to be concise and clear. The report has an *advisory word limit of 1800 words*. You *do not* have to reach this word limit.

Requirements

You should submit two elements to MMS (compressed as a zip file):

- All code: the provided code, your code with the benchmark instances, and your README.
- A compact report (as a PDF file) presenting work as described above.

Marking

Your mark will depend on the overall quality and completeness of your submission, including the depth of your analysis, and the extent to which you evidence your findings. The submission will be marked on the University's common reporting scale, according to the mark descriptors given in the CS Student Handbook (at `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`).

Lateness

The standard penalty for late submission applies (Scheme A: 1 mark per 24 hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

I would remind you to ensure you are following the relevant guidelines on good academic practice as outlined at

https://www.st-andrews.ac.uk/education/handbook/good-academic-practice/

Hints for completing the practical

1. The Maximum Clique Problem has had many different approaches, you might want to look into academic literature for some inspiration for how to make the search more efficient. Some are listed here `https://www.csplib.org/Problems/prob074/references/`. Note that some of these papers might only be accessible through the University Library.
2. The benchmark set has been shortened to contain instances that should be solvable with an efficient algorithm within 5mins (each). Some will be unsolvable with the provided algorithm (because it is that inefficient).
3. Use the information on the benchmark set webpage to check if your algorithm is still correct.
4. An FAQ document will be released on StudRes and updated with questions from your peers. Keep an eye on it.
5. I do not recommend attempting to parallelise the search. Note that that also entails changing the search type which you are not supposed to do.