Title:	Methods and Tools for Software Engineering
Course ID:	ECE 650 Section 01
WWW:	`https://ece.uwaterloo.ca/~agurfink/ece650/`
LEARN:	`https://learn.uwaterloo.ca`
Campuswire:	`https://campuswire.com`
Instructor:	Prof. Arie Gurfinkel, `arie.gurfinkel@uwaterloo.ca`, EIT 4021
TA:	Aosen Xiong `@TAAosen` `a4xiong@uwaterloo.ca`

Office hours by appointment. Begin all email subjects with `[ECE650]`. Use **Campuswire** instead of email whenever possible!

Final Course Project — Due April 12, 2024

The skeleton for the course project is available at the main branch of `https://git.uwaterloo.ca/ece650-1241/skeleton` in directory `project`. Follow the instructions in Assignment 0 to correctly fetch and merge the files from the skeleton!

The project can be done in **groups of 2**. If you decide to do the project in a group, make sure that you request us to create a single group repository on GitLab.

This is the final course project. For the project you will need to:

- Augment your code from Assignment 4 in the way that is decribed below.
- Quantitatively analyze your software for various kinds of inputs.
- Write a brief report ($\approx$ 5 pages, 11 pt font, reasonable margins) with your analysis. Your report must be typeset in LaTeX, and must be in PDF. You can use Overleaf to collaborate on a LaTeXdocument.

You should augment your code from Assignment 4 in the following ways.

- Make it multithreaded. You should have at least 4 threads: one for I/O, and one each for the different approaches to solve the minimum vertex cover problem.
- Implement the following two additional ways to solve MIN-VERTEX-COVER, in addition to the REDUCTION-TO-CNF-SAT approach you had in Assignment 4. (We will call your approach from Assignment 4, CNF-SAT-VC.)

  1. Pick a vertex of highest degree (most incident edges). Add it to your vertex cover and throw away all edges incident on that vertex. Repeat till no edges remain. We will call this algorithm APPROX-VC-1.
  2. Pick an edge $\langle u, v\rangle$, and add both u and v to your vertex cover. Throw away all edges attached to u and v. Repeat till no edges remain. We will call this algorithm APPROX-VC-2.

Inputs

As input, use the output of `/home/agurfink/ece650/graphGen/graphGen` on `eceubuntu`. That program generates graphs with the same number of edges for a particular number of vertices, but not necessarily the same edges. Note that you can store its output in a file and use the file on other machines.

Output

Given a graph as input, your program should output the vertex cover computed by each approach in sorted order. That is, give the following input:

```
V 5
E {<3,2>,<3,1>,<3,4>,<2,5>,<5,4>}
```

The output from your program should be:

```
CNF-SAT-VC: 3,5
APPROX-VC-1: 3,5
APPROX-VC-2: 1,3,4,5
```

That is, the name of the algorithm, followed by a colon ':', a single space, and then the computed result as a sorted sequence of vertices, separated by commas.

Analysis

You should analyze how efficient each approach is, for various inputs. An input is characterized by the number of vertices. "Efficient" is characterized in one of two ways: (1) running time, and (2) approximation ratio. We characterize the approximation ratio as the ratio of the size of the computed vertex cover to the size of an optimal (minimum-sized) vertex cover.

For measuring the running time, use `pthread_getcpuclockid()`. For an example of how it is used, see `http://www.kernel.org/doc/man-pages/online/pages/man3/pthread_getcpuclockid.3.html`.

For measuring the approximation ratio, compare it to the output of CNF-SAT-VC, which is guaranteed to be optimal.

Your objective is to measure, for various values of $|V|$ (number of vertices), for the graphs generated by `graphGen`, the running time and approximation ratio. You should do this by generating graphs for $|V| \in [5, 50]$ using that program, in increments of 5. That is, graphs with $5, 10, 15, \ldots, 50$ vertices.

You should generate at least 10 graphs for each value for $|V|$, compute the time and approximation ratio for each such graph. You should measure the running time for at least 10 runs of each such graph. Then, you should compute the mean (average) and standard deviation across those 100 runs for each value of $|V|$. For the approximation ratio, if there is any random component (e.g., which edges you choose, for APPROX-VC-2), then you should measure that multiple times as well for each graph.

You might find the optimal approach (CNF-SAT-VC) difficult to scale to large number of vertices. For these instances, you can use a timeout to avoid waiting for its result and produce `CNF-SAT-VC: timeout` in the output.

CNF-SAT-VC can be scaled significantly by improving the encoding. Should you decide to improve the encoding, **bonus** points will be given for scaling to larger instances. Make sure to describe the imporvements you made to the encoding in the report. Points will be deducted for encodings that are scalable but are either incorrect or unexplained.

Report

The main part of your report are graphs (plots) corresponding to the data you generate as described in the "Analysis" section above. One way to show the output is to have two plots: one for running times and the other for approximation ratio. The horizontal axis is the number of vertices.

You should plot the mean for each value of $|V|$ for which you made measurements, and the standard deviation as an errorbar[1]. An example of a possible plot is shown in Figure 1.

The remainder of your report should be reasoning about your plots. That is, you should explain why your plots look the way they do. For example, if there is a "spike" in the approximation ratio for some value of $|V|$ for one of the approaches, you should explain why there is such a spike. You should also explain apparent trends. For example, if, for one of the approaches, the running time seems to increase linearly with $|V|$, you should reason about why that is happening.

[1] You can use yerrorbar in gnuplot to draw an error bar: `http://gnuplot.sourceforge.net/docs_4.2/node262.html`

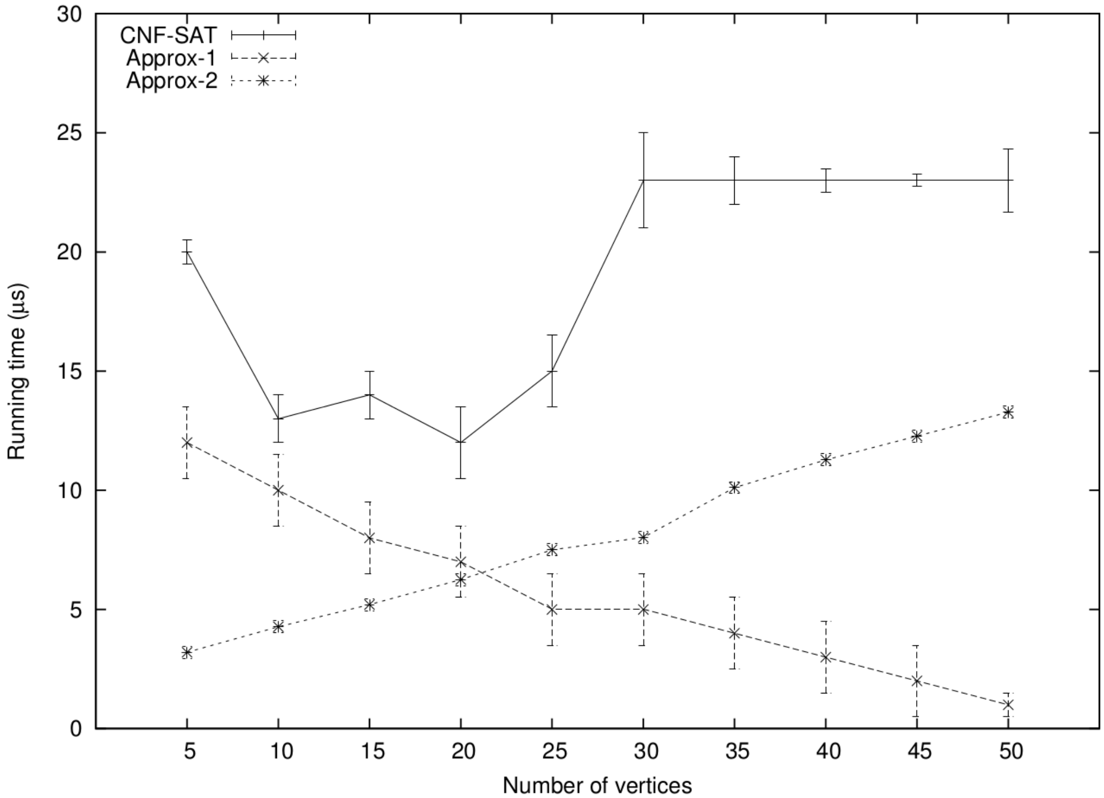

Figure 1: Example plot, generated using `gnuplot`. The error bars for Approx-2 are not visible because the standard deviation is small.

Marking

We will mark by: (1) Trying some inputs and checking your output, (2) inspecting your code to make sure that you are using pthreads correctly, and, (3) reading your report.

- Marking script for compile/make etc. fails: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Correctly implemented 2 new algorithms: + 20 each, total + 40
- Generated plots: + 20
- Report: + 20
- Bonus: + 10

CMake

As discussed below under "Submission Instructions", you should use a `CMakeLists.txt` file to build your project. We will build your project using the following sequence:

```
cd project && mkdir build && cd build && cmake ../
```

You can assume that MiniSat will be placed in directory `project/minisat`. Your submission should only include your own code. If your code is not compiled from scratch (i.e., from the C++ sources), you get an automatic 0. Unlike for the assignments, you must create the `CMakeLists.txt` file on your own. You can use a `CMakeLists.txt` file from previous projects or from the course examples provided on GitLab.

Submission Instructions

You should place all your files in the `project` directory in your GitLab repository. The directory should contain:

- All your C++ source-code files.
- A `CMakeLists.txt`, that builds your C++ executable `ece650-prj`.
- A file `user.yml` that includes your name, WatIAM, and student number of all the team members. Note that *WatIAM* is the user name for your Quest account, e.g. `agurfink`, and a *student number* is an 8-digit number, e.g. `20397238`. If you have done the assignment as a group, the information for both members of the group should be included in the `user.yml`.
- A file named "`report.pdf`" with your report.

See `README.md` for any additional information.

The submitted files should go to `project` directory in your repository.