Title:	Methods and Tools for Software Engineering
Course ID:	ECE 650
WWW:	`https://ece.uwaterloo.ca/~wdietl/teaching/2014f/ece650/`
Lectures:	Thursday, 17:30 – 20:20
Instructor:	Dr. Werner Dietl, `wdietl@uwaterloo.ca`, DC 2522
TA:	Hua Fan, `h27fan@uwaterloo.ca`, DC 2628

Office hours by appointment. Begin all email subjects with `[ECE650]`.

Assignment 5 - Due Fri, Nov. 28, 12:00 noon

For this assignment, you 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 (≈ 5 pages, 11 pt font, reasonable margins) with your analysis. Your report must be typeset, and must be in PDF.

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

- Make it multithreaded. You should have at least 6 threads: one for I/O, and one each for the different approaches to solve the minimum vertex cover problem.
- Implement the following four 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.)
  - 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.
  - A refinement to the above algorithm. After you compute the set of vertices, go through them and throw away any vertex you do not need. That is, if C is the vertex cover you compute from running APPROX-VC-1, you can throw away vertex v from V if $C - \{v\}$ is a vertex cover. You can choose the vertices greedily. That is, simply go through the set of vertices and throw the unnecessary ones away one-by-one. We will call this algorithm REFINED-APPROX-VC-1.
  - Pick an edge <u, v>, 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.
  - A refinement to APPROX-VC-2, which we will call REFINED-APPROX-VC-2. In the refinement, you take the output vertex cover of APPROX-VC-2 and throw away any vertices that are not needed. As for APPROX-VC-1, you can choose the vertices greedily.

Inputs

As input, use the output of `/home/wdietl/graphGen/graphGen` on `ecelinux`. That program generates graphs with the same number of edges for a particular number of vertices, but not necessarily the same edges.

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 {<2,1>,<2,0>,<2,3>,<1,4>,<4,3>}
```

The output from your program should be:

```
CNF-SAT-VC: 2,4
APPROX-VC-1: 2,4
REFINED-APPROX-VC-1: 2,4
APPROX-VC-2: 0,2,3,4
REFINED-APPROX-VC-2: 2,4
```

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 `/home/wdietl/graphGen/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.

Report

The main part of your report are graphs (plots) corresponding to the data you generate as I describe 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 a yerrorbar. So, your plot should look something like what 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.

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 uncompressing/compile/make etc. fails: automatic 0
- Your program runs, awaits input and does not crash on input: + 20
- Correctly implemented 4 new algorithms: + 10 each, total + 40

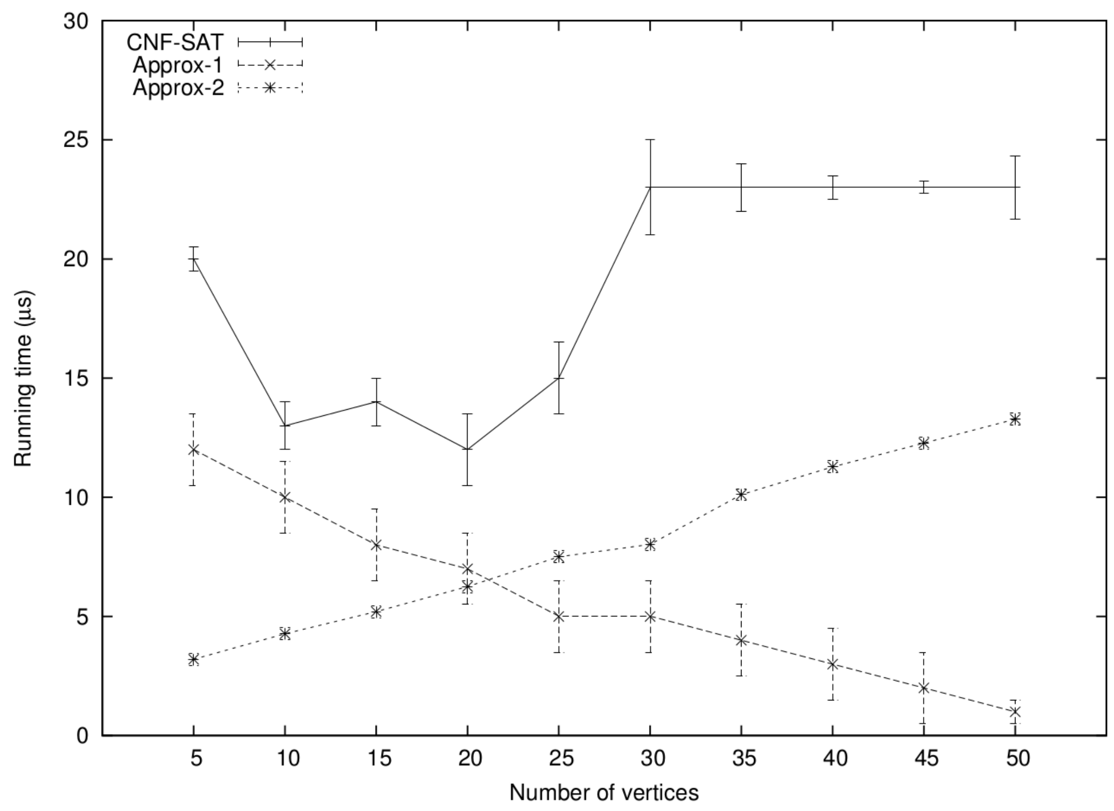

Figure 1: Example plot, generated using `gnuplot`. The error bars for Approx-2 are not visible because the standard deviation is small.

- Generated graphs: + 20
- Report: + 20

Submission Instructions

You should place all your files in a single folder (directory), `a5-ece650`. The folder should contain:

- A Makefile. It should contain a target "`all`" that makes the `a5-ece650` executable, and a target "`clean`" to clean all your intermediate (e.g., `.o`) files. If we do a "`make clean`" followed by a "`make all`", then everything should compile from scratch.
- All your C sources. We will supply `libsat.a` and `SAT.h` from `zchaff` in the current folder.
- A file named "`report.pdf`" with your report.

Your submission should not contain anything from `zchaff`.

You should `tar` and `gzip` the folder using the command: `tar czf a5-ece650.tar.gz a5-ece650`. (To `untar` and `gunzip`, we will issue `tar xzf a5-ece650.tar.gz`.)

You should upload `a5-ece650.tar.gz` to the dropbox on Learn for Assignment 5.