Concordia University
Dept. of Computer Science and Software Engineering

COMP 691 – Online Algorithms and Competitive Analysis
Winter 2019

Project
Submission deadline: Monday April 20th at 23:55

You can work in groups of up to 3 students.

For the project in this course, you can choose one of the following two options.

1 Option 1 – Propose Your Own Project

You may propose your own project. For that you need to submit a project proposal to me by email by **23:55 on Sunday, March 22nd**. The project proposal should be a pdf file containing the following components:

1. *High-level description* of the project. Mention whether you are working in a group (include names and student IDs of all students in the group) or alone.
2. *Literature review.* Describe textbooks or papers that your project will be based on.
3. *Main components of the project.* Describe what you plan to include in the final writeup component (mandatory), describe other components of the project, such as coding, libraries, inputs, etc.
4. *Evaluation criteria.* How do you suggest your project should be judged? What will indicate a successful project? What will indicate a failed project?
5. *Any additional information* that will help me judge the quality of the proposal.

Your project should be in the area of online algorithms and it should have comparable size and difficulty to the Option 2 described below.

2 Option 2 – Standard Project

In this option, you will implement **CBIP** and **FirstFit**[1] algorithms for the online coloring problem of bipartite graphs in the VAM-PH input model, evaluate and compare their performance on different inputs, **propose another algorithm** trying to improve performance, and report your findings.

[1]FirstFit is the simple greedy algorithm that colors a new vertex with the smallest color not used on any of its neighbors that have already been revealed in the input.

2.1 Parameters of the Problem

In this project, you will run online coloring algorithms only with bipartite graphs. The number of vertices in the input graph is going to be at most 1000.

2.2 Graph Format

In this project you will be reading/writing graphs from files. These graphs might not be necessarily bipartite, so you will convert them into bipartite graphs using **The Duplicating Method** and **The Random Balanced Partition Method** described in Section 2.4 below. Here we describe the format for reading/writing general graphs. You will be working with Network Repository (`http://networkrepository.com`) – one of the largest freely available online databases of graphs. Thus, you will use the Matrix Market Coordinate (MMC for short) Format to represent undirected unweighted graphs, since this format is most popular on Network Repository. Undirected graphs in this format are represented as follows.

```
% comment line 1
% comment line 2
n n m
u1 v1
u2 v2
...
um vm
```

More specifically, any line starting with % is a comment and should be ignored. The first line that does not start with % contains 3 integers n, n, m. The integer n is repeated twice and stands for the number of vertices $|V|$. The integer m stands for the number of edges $|E|$. The next m lines contain description of edges as pairs of vertices. For example, the following example shows how a graph with 4 nodes and 6 edges $\{1,2\}, \{1,3\}, \{2,3\}, \{1,4\}, \{2,4\}, \{3,4\}$ can be represented as follows:

```
%MatrixMarket matrix coordinate pattern symmetric
4 4 6
2 1
3 1
3 2
4 1
4 2
4 3
```

Note that vertex names are 1-based, i.e., $1 \le u_i \le n$ and $1 \le v_i \le n$. Also note that each edge appears only once as a pair (u_i, v_i). Since we are working with undirected graphs you should remember to add two pairs (u_i, v_i) and (v_i, u_i) to your datastructure, if that's how you are representing the graph.

More information on this format can be found at

`http://networkrepository.com/format-info.php`

Lastly, we note that not all files on Network Repository follow this format. If there is a graph from Network Repository that you want to use in this project but it's in the wrong format, then you have to either manually convert it or write a program to convert it to this format. Alternatively, you can simply find a graph that satisfies this format.

2.3 Implementation Details

You must use one of the following programming languages: C, C++, Java, Python. You are allowed to use only the libraries that come together with the standard distribution of a compiler/interpreter (for example, STL for C++ is allowed). Anything else has to be implemented by you.

You are free to structure your implementation the way you want, but you must have the following functions:

readGraph(*filename*). Reads graph information from the file *filename* in the MMC format and converts it into an internal representation of the graph (not necessarily bipartite) of your choice. You have a freedom of choosing an internal representation of the graph, just make sure that it is sufficiently efficient.

makeBipartite(G, method). Takes an internal representation of a general graph that might not be bipartite and converts it into an internal representation of a bipartite graph using either the Duplicating or the Balanced Random Partition method (see Section 2.4 for more details).

FirstFit(G, σ). The inputs for this function are an internal representation of a bipartite graph G and the order σ according to which G is presented to FirstFit in the VAM-PH input model. This function should return the *number of colors* used by FirstFit on this input graph.

CBIP(G, σ). The inputs for this function are an internal representation of a bipartite graph G and the order σ according to which G is presented to CBIP in the VAM-PH input model. This function should return the *number of colors* used by CBIP on this input graph.

MyAlgorithm(G, σ). The inputs for this function are an internal representation of a bipartite graph G and the order σ according to which G is presented to your coloring algorithm (should be different from FirstFit or CBIP) in the VAM-PH input model. This function should return the *number of colors* used by your algorithm on this input graph.

The signatures of the above functions can be different from the suggested above. If you would like to pass some auxiliary information around, you can add it as an additional parameter. If you would like to return more than one thing, you can also do that. In addition to the above functions you will probably find it useful to write many helper methods.

Source code should be well documented with proper comments and names of variables. The code should be easy to read. I will be checking the code for correctness and readability.

In addition to the above methods, there must be a way to run your code so that it generates *ALL* the results that will appear in your writeup. You can do this by writing a small wrapper around the four methods mentioned above. I should be able to receive your submission, read the README file, understand how to compile your code (if there is a need for it), then run a single command, e.g., */.a.out*, and it would reproduce all figures and data used in your writeup. If your code doesn't compile, or doesn't reproduce similar results, you may lose a significant portion of the points.

2.4 Evaluation Details

Your evaluation will consists of two parts.

First part is evaluation of algorithms with respect to *random* graphs. The random bipartite Erdős-Rényi graph is denoted as $G(n, n, p)$ and is generated as follows. The two blocks U and V of vertices in the bipartite graph are each fixed to be $\{1, 2, \ldots, n\}$ and the set of edges E between U and V is generated randomly. Each edge $\{i, j\}$ is included in E with probability p and excluded from E with probability $1 - p$. In particular, $G(n, n, p)$ graphs have n^2p edges on average.

NOTE: Input graphs are bipartite and we are dealing with them in the VAM-PH input model, not the BVAM model which we studied in the context of online matching.

You should run your implementation on $G(n, n, p)$ with different values of $n \leq 500$ (so that the total number of vertices is $2n \leq 1000$), different values of p, and with different presentation orders of vertices $U \cup V$. You should consider different ways of ordering input items. Some suggestions: (a) generate the input sequence in random order; (b) generate the input sequence by alternatively presenting one node from U and one node from V until all nodes are presented; (c) generate the input sequence by alternatively presenting 5 nodes from U and 5 nodes from V; (d) anything else you might want to try. For each value of p, n, and input order presentation, you might need to run your implementation multiple times until the sample average of the results starts to stabilize. Observe if there is any relationship between the performance of the algorithms FirstFit, CBIP, and Your Algorithm and p, or n, or order of presentation. Summarize your results in a figure as time series or in a table. There is a lot of freedom in how you can choose your experiments and how you can run them.

Second part is evaluation of algorithms with respect to benchmarks from Network Repository (`http://networkrepository.com`). Find 5 graphs on the Network Repository with at most 1000 and no less than 50 nodes. Perform similar experiments as for the Erdős-Rényi model, but with respect to the 5 graphs you found on the network repository. Report your findings in a figure or a table.

Note that if you have trouble finding graphs that are bipartite, you can instead start with a non-bipartite graph $G = (V, E)$ and create a bipartite graph out of it using the following methods:

The Duplicating Method. Duplicate the vertex set V. Let L be the first copy of V and R be the second copy of V. Put an edge between $\ell \in L$ and $r \in R$ if and only if $\{\ell, r\} \in E$.

The Random Balanced Partition Method. Partition V into two blocks L and R randomly such that $|L| = \lceil |V|/2 \rceil$ and $|R| = \lfloor |V|/2 \rfloor$, keep only those edges from the original G that have one endpoint in L and another endpoint in V.

2.5 Writeup Details

Write a report of at least 5 and no more than 7 pages (not including title and bibliography) explaining what you have done, what difficulties you had during implementation, results that you found. You can also suggest further possible improvements and future progress. Your writeup must have the following sections:

Title, Author(s) Name(s), Date. Self-explanatory.

Introduction. Brief description of the problem, its significance, the main message of the paper, and briefly mention the key results.

Background. Contains some history about the online coloring problem. Historical notes and references at the end of Chapter 5 can be helpful for this, but you should go beyond what's just written there.

New Algorithm. This is where you describe your algorithm that you propose that is different from FirstFit and CBIP. You will evaluate this algorithm alongside FirstFit and CBIP.

Implementation details. This is where you describe important implementation details. You don't have to explain every aspect of your code. You only have to describe the structure of your code and what were the most challenging aspects of the implementation and how you overcame them. You should also explain how you tested your implementation and why you are sure there are no bugs.

Experimental Setup. This is where you explain details of your experimental setup. What architecture did you use? What versions of compilers/interpreters? How fast was the CPU? How many cores? How much RAM? What graphs did you choose for the evaluation? Why?

Results. This is where you present your findings, time-series plots, tables, and comment on what you see. You need to analyze how the performance of CBIP compares to FirstFit and Your Algorithm, any patterns that you found, any general observations. The deeper and the more interesting observations the better your grade will be.

Future directions. This is where you explain in which direction you can extend this project. Mention interesting open problems, ideas you came across, etc. If you had more time, what things would you have liked to try?

Bibliography. Cite anything that you have used, including survey papers and Network Repository.

The writeup must be typed up and must be in the PDF format. You are strongly encouraged to use LaTeX for that, although MS Word or other word processing software can also be used for this purpose.

Just as a guide, you could allocate page limits to the sections as follows: Intro and Background together occupy 1 page, New Algorithm could be 0.5-1 page, Implementation Details and Experimental Setup together could occupy 1 page, Results section occupies 2.5 pages, and Future Directions occupies 0.5 pages.

2.6 What to Hand In

You are allowed to work in groups of up to 3 students. If you are working in a group, **exactly one person** from the group must submit the assignment. DO NOT MAKE 2 or 3 SUBMISSIONS FOR THE SAME GROUP.

You should create a single .zip archive and upload it to Moodle before the deadline. The file must contain the following items:

1. Source code – could be a single file or multiple files if you decided to split your implementation into several modules.
2. Files from the network repository. These are the graphs that you used as benchmarks for your evaluation.
3. PDF of the writeup.
4. README file – this should be a text file explaining the structure of your submitted .zip archive and instructions on how I can reproduce your results.

Submissions significantly deviating from the above standard could lose points.