Assignment 5
COMP6741: Parameterized and Exact Computation
Serge Gaspers
19T3

Assignment 5 is a group assignment. The groups are the same as for Assignment 4.

For the solutions to this assignment, you may rely on all theorems, lemmas, and results from the lecture notes. If any other works (articles, Wikipedia entries, lecture notes from other courses, etc.) inspired your solutions, please cite them and give a list of references at the end of your report. You may use any result you find in the literature, without re-proving it. Existing implementations and libraries may also be used, as long as their licenses allow unrestricted academic use.

If you have questions about this assignment, please post them to the Forum.

Due date. This assignment is due on Tuesday, 12 Nov 2019, at 17:59 Sydney time. Submitting $x$ hours after the deadline, with $x > 0$, reduces the grade by $\lfloor x \rfloor \%$ per cent.

How to submit. Use the existing Bitbucket GIT repository that was assigned to your group. The Readme file in this repository describes where to put various files, including the report answering the questions below, by the submission deadline.

Assignment 4 already mentioned two parameters for **Feedback Vertex Set**, namely the vertex cover number and the degree-2 deletion number. It was also stated in Assignment 4 that

In Assignment 5, you will be asked to implement an algorithm for **Minimum Feedback Vertex Set** and analyse the performance of the implementation with respect to the parameters vertex cover number and degree-2 deletion number. In particular, we would like to find out whether your implementation is fast if the graph has small vertex cover number or small degree-2 deletion number. You will run your implementation on benchmark instances.

We will use **sagemath** for all implementations. See [http://www.sagemath.org/](http://www.sagemath.org/).

In addition, you have already implemented a method for generating instances for **Minimum Feedback Vertex Set**.

Exercise 1. This should be done in the beginning of your work on Assignment 5, but can evolve over time. Give a plan for the work done in Assignment 5, including milestones, a timeline with deadlines, assignment of tasks/subtasks to group members (some of these tasks may be to review the work of others). Be conscious that several tasks need to be performed after the implementation is completely finalized, including running it on benchmark instances (estimate how long this takes), evaluating the results, and finalizing your report. Ideally, you would also leave some additional time in case there are unforeseen issues. Did your timeline evolve over time? If any issues or setbacks arose, how did you handle them? Document all of this in your report. [10 points]

Exercise 2. Design an algorithm (optionally, re-use a known algorithm from the literature), which, given an instance for **Minimum Feedback Vertex Set**, outputs a smallest feedback vertex set of the instance. This algorithm is expected to be fast when the graph has small degree-2 deletion number of small vertex cover number. Describe this algorithm in the report and argue it is fast in these cases, for example by upper bounding the running time in terms of both the degree-2 deletion number and the vertex cover number of the input graph. Instead of a worst-case analysis of the running time in terms of these parameters, other arguments are very welcome as well. This algorithm may consist of sub-algorithms and a selection algorithm which decides which sub-algorithm to run. [20 points]
Exercise 3. Implement the algorithm of Exercise 2 or a variant of it (in case it is a variant, describe why the implementation differs from the algorithm described in Exercise 2). The implementation takes as argument a graph (of type `sage.graphs.graph.Graph`) and returns a smallest feedback vertex set (of type `list`). Valid reasons for the implementation differing from Exercise 2 include the simplicity and the practical efficiency of the implementation. Document your code (imagine next year’s class would like to re-use part of your code), implement some testing and validation procedures. Describe in your report how your algorithm can be tested on a suite of benchmark instances. [30 points]

Exercise 4. Describe your test environment (details on CPU, operating system, system memory, etc.). [5 points]

Exercise 5. Run the implementation of Exercise 3 on the final set of benchmark instances in your test environment, with a 5-minute time-out (if you have a good reason to select a different time-out, please state why). Also, run Sagemath’s standard implementation (`G.feedback_vertex_set()` for a graph G) on the same benchmark instances. Depict the results in a cactus plot. Interpret the results: when is your algorithm faster than the standard algorithm in Sagemath; do some profiling; in case you use sub-algorithms, which sub-algorithm performed best; among the instances that you generated, did your implementation have the expected running times? Are your results reproducible (i.e., is it easy for anyone with access to your report, your code, a similar test environment, and an Internet connection to reproduce your results)? [25 points]

Exercise 6. What can be improved in your implementations, and what are the obstacles that need to be overcome for these improvements? Is your implementation a viable candidate to be added to the graph algorithms of Sagemath? [5 points]

Exercise 7. As per Assignment 4, benchmark instances are provided as sage graph objects in saved files. Please provide at least 2 disadvantages of saving benchmark instances in this format and suggest better formats for saving graph benchmark instances. [5 points]

1 see, e.g., https://github.com/alexeyignatiev/mkplot