CSE6140 Fall2015 Project

October 29, 2015

1 Overview

The Minimum Vertex cover (MVC) problem is a well known NP-complete problem with numerous applications in computational biology, operations research, the routing and management of resources. In this project, we will treat the problem using different algorithms, evaluating their theoretical and experimental complexities on real world datasets.

2 Objective

- Get hands-on experience in solving an intractable problem that is of practical importance
- Implement an exact branch-and-bound algorithm for Minimum Vertex Cover
- Implement approximate algorithms for Minimum Vertex Cover, that run in reasonable time and provide high-quality solutions
- Implement heuristic algorithms for Minimum Vertex Cover (without approximation guarantees)
- Develop your ability to conduct empirical analysis of algorithm performance on datasets, and to understand the trade-offs between accuracy, speed, etc., across different algorithms
- Develop teamwork skills while working with other students

3 Groups

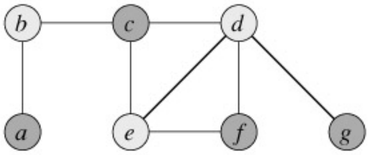
Your groups have been posted to T-Square under:
Resources → Project → groups.pdf
If there are concerns with programming language compatibility, please email us with your team name and the concern.

4 Background

Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, a vertex cover is a subset $C \subseteq V$ such that $\forall (u, v) \in E : u \in C \vee v \in C$. The Minimum Vertex Cover problem is therefore simply the problem of finding minimizing $|C|$.

For more details on the MVC problem, refer to Karp's original 21 NP-Complete problems `http://www.cs.berkeley.edu/~luca/cs172/karp.pdf`.

The following diagram shows an example of minimal vertex cover in light shade



5 Algorithms

You will implement algorithms that fall into three categories:

1. exact, computing an optimal solution to the problem
2. construction heuristics some of which have approximation guarantees
3. local search with no guarantees but usually much closer to optimal than construction heuristics.

In what follows, we present the high-level idea behind the algorithms you will implement -

1. **Exact algorithm using Branch-and-Bound**. Implement the Branch-and-Bound algorithm for Vertex Cover as seen in class. You may design any lower bound of your choice, but consider the following strategies as a guidance:

   - Let C' be a partial vertex cover of G, and let $G'(V', E')$ be the subgraph of G not covered by C'. $V' = V - C' - \{v \in V | \forall (v, u) \in E, u \in C'\}$ (note we prune nodes that cannot help us cover any uncovered edges) and $E' = \{(u, v) \in E | v, u \in V'\}$. Then, a lower bound can be calculated as $LB = |C'|$+(lower bound on the VC for G'). [Note that one lower bound we know how to obtain is from our approximation algorithms (the approx solution quality A is $\leq 2OPT$, hence $LB = \frac{1}{2}A \leq OPT$). The approx algorithm in class finds a maximal matching given a graph, one way to find an even tighter bound is to compute the *maximum* matching of G' (this a more complicated problem but still has a polynomial time algorithm). Alternatively, for those of you who want to try this, given G' you can construct the corresponding LP relaxation for finding minimum VC, use an available LP solver in your programming language of choice, and obtain the LP optimal solution as a lower bound to use in your branch and bound implementation (note you are not allowed to use a solver that directly solved the ILP using branch-and-bound).]

Since this algorithm is still of worst-case exponential complexity, your algorithm must be cut off after running for T minutes, and the current best solution must be returned if one has been found. Clearly, for small datasets, this algorithm will most likely return an optimal solution, whereas it will fail to do so for larger datasets.

2. **Construction Heuristics with approximation guarantees**. Please implement the constructive heuristic for MVC with approximation guarantees as seen in lecture 19 on the 29^{th} of October. The algorithm in class chooses any remaining edge in E'. You may decide to augment the algorithm with some ways of prioritizing the vertices in E'. Optionally, you may implement one of the approximation algorithms with guarantees described in [1].

3. **Local Search**. There are many variants of local search and you are free to select which one you want to implement. Please implement 2 types/variants of local search. They can be in different families of LS such as SA vs Genetic Algorithms vs Hill Climbing, or they can be in the same general family but can differ by the neighborhood they are using, or the perturbation strategy, etc. They need to be different enough to observe qualitative differences in behavior. Optionally, you may use one of the algorithms described in [1] to compute initial solutions for your variants of local search.

6 Data

You will run the algorithms you implement on some real and random datasets extracted from the 10th DIMACS challenge, which can be downloaded from T-Square: Resources → Project → DATA.zip

Each graph in the data set is represented in the popular Metis I/O format. An unweighted graph $G(V, E)$ is represented as follows:

- A first line with three integer numbers representing the size of the graph and a zero(indicating the graph is unweighted): $|V|$ $|E|$ 0
- $|V|$ lines follow. Where line i specifies the list the adjacencies of vertex i.

See section 4 of `http://www.cc.gatech.edu/dimacs10/data/manual.ps` for further details.

Below is a table describing each of the real-world instances:

Table 1: Real world instances

Problem name	**Description**	$\|V\|$	$\|E\|$
jazz.graph	Jazz musicians network . List of edges of the network of Jazz musicians.	198	2742
karate.graph	Zachary's karate club: social network of friendships between 34 members of a karate club at a US university in the 1970s.	34	78
football.graph	American College football: network of American football games between Division IA colleges during regular season	115	613
as-22july06.graph	Graph of the whole Internet: a symmetrized snapshot of the structure of the Internet at the level of autonomous systems, reconstructed from BGP tables	22963	48436

7 Code

All your code files should include a top comment that explains what the given file does. Your algorithms should be well-commented and self-explanatory. Use a README file to explain the overall structure of your code, as well as how to run your executable.

Your executable should take as input i) the filename of a dataset and ii) the cut-off time (in seconds) iii) the method to use, iv) a random seed. If it is run with the same 4 input parameters, your code should produce the same output.

Any run of your executable with any four inputs (filename, cut-off time, method and seed) must produce two types of output files:

1. Solution files:
    - File name: $< inputFilename > \_ < method > \_ < cutoff > \_ < randSeed > *.sol$, e.g. $jazz\_BnB\_600.sol$, $jazz\_LS1\_600\_4.sol$ Note that as in the example above, randSeed is only applicable when the method of choice is randomized (e.g. local search). When the method is deterministic (e.g. branch-and-bound), randSeed is omitted from the solution file's name.
    - File format:
        (a) line 1: quality of best solution found (integer)
        (b) line 2: list of vertex IDs of the vertex cover (comma-separated): $v_1, v_2, v_2, ..., v_n$
2. Solution trace files:

- File name: $< inputFilename > \_ < method > \_ < cutoff > \_ < randSeed > *.trace$, e.g. $jazz\_BnB\_600.trace$, $jazz\_LS1\_600\_4.trace$. Note that randSeed is used as in the solution files.
- File format: each line has two values (comma-separated):
  (a) A timestamp in seconds (double)
  (b) Quality of the best found solution at that point in time (integer). Note that to produce these lines, you should record every time a new improved solution is found.
  Example:
  3.45, 102
  7.94, 95

8 Output

You should run all the algorithms you have implemented on all the instances we provide, and submit the output files generated by your executable, as explained in the Code section.

P.S.: DO NOT SUBMIT THE INPUT DATA FILES.

9 Evaluation

We now describe how you will use the outputs produced by your code in order to evaluate the performance of the algorithms.

1. Comprehensive Table: Include a table with columns for each of your MVC algorithms as seen below. For all algorithms report the relative error with respect to the optimum solution quality provided (soon) to you in the MVC instance files. Relative error ($RelErr$) is computed as $(Alg - OPT)/OPT)$. Round time and $RelErr$ to two significant digits beyond the decimal. For local search algorithms, your results for each cell should be the average of some number (at least 10) of runs with different random seeds for that dataset. You will fill in average time (seconds) and average vertex cover size. You can also report in any other information you feel is interesting.

	Branch and Bound			Etc. (other algorithms)		
Dataset	Time (s)	VC Value	RelErr	Time (s)	VC Value	RelErr
x.graph	3.26	3400	0.0021			

Additional evaluation plots applicable to local search algorithms only will be further required and specified in the future. All the information you need to produce these plots is in your solution trace files.

1. Qualified Runtime for various solution qualities (QRTDs): A plot similar to that of slide 24 of [cse6140-fall2015-lecture15-Empirical.pd]. The x-axis is the run-time in seconds, and the y-axis is the fraction of your algorithm runs that have 'solved' the problem. Note that 'solve' here is w.r.t. to some relative solution quality q^*. For instance, for $q^* = 0.8\%$, a point on this plot with x value 5 seconds and y value 0.6 means that in 60% of your runs of this algorithm, you were able to obtain a tour of cost at most the optimal cost plus 0.8% of that. When you vary q^* for a few values, you obtain the points similar to those in slide 24.

2. Solution Quality Distributions for various run-times (SQDs): Instead of fixing the relative solution quality and varying the time, you now fix the time and vary the solution quality (slide 28 of lecture PDF). The details are analogous to those of QRTDs.

3. Box plots for running times: Since your local search algorithms are randomized, there will be some variation in their running times. You will use box plots, as described in the 'Theory' section of this blog post: `http://informationandvisualization.de/blog/box-plot`. Read the blog post carefully and understand the purpose of this type of plots. You can use online box plot generators such as `http://boxplot.tyerslab.com/` this one to produce the plot automatically from your data.

10 Report

1. Formatting

   You will use the format of the Association for Computing Machinery (ACM) Proceedings to write your report.

   For users of:

   - Word: download the template from `http://www.acm.org/sigs/publications/pubform.doc`.
   - LaTeX: download the 'Option 2: LaTex2e - Tighter Alternate style' Style File V2.5 (.CLS file) and template Sample File V2.0 (.TEX file) from `http://www.acm.org/sigs/publications/proceedings-templates#aL2`.

2. Content

   Your report should be written as if it were a research paper in submission to a conference or journal. A sample report outline looks like this:

   - Introduction: a short summary of the problem, the approach and the results you have obtained.
   - Problem definition: a formal definition of the problem.

- Related work: a short survey of existing work on the same problem, and important results in theory and practice.
- Algorithms: a detailed description of each algorithm you have implemented, with pseudo-code, approximation guarantee (if any), time and space complexities, etc. What are the potential strengths and weaknesses of each type of approach? Did you use any kind of automated tuning or configuration for your local search? Why and how you chose your local search approaches and their components? Please cite any sources of information that you used to inform your algorithm design.
- Empirical evaluation: a detailed description of your platform (CPU, RAM, language, compiler, etc.), experimental procedure, evaluation criteria and obtained results (plots, tables, etc.). What is the lower bound on the optimal solution quality that you can drive from the results of your approximation algorithm and how far is it from the true optimum? How about from your branch-and-bound?
- Discussion: a comparative analysis of how different algorithms perform with respect to your evaluation criteria, or expected time complexity, etc.
- Conclusion

11 Deliverables

Failure to abide by the file naming and folder structure as detailed here will result in penalties.

1) Each group must submit a Report: a PDF file of the report, titled

$$group\_ < group\_id > - < gtusername > -Report.pdf$$

following the guidelines in section Report.

2) Each group member should submit the same ZIP or RAR file, titled:

$$group\_ < group\_id > - < gtusername > .zip$$

Example: $group\_F - smith3.zip$

The file must have the following files/folders:

2.1 Code: a folder named ‘code‘ that contains all your code, the executable and a $README - group\_ < group\_id > .txt$ file, as explained in section Code.

2.2 Output: a folder named ‘output‘ that contains all output files, as explained in section Code.

References

[1] François Delbot and Christian Laforest. Analytical and experimental comparison of six algorithms for the vertex cover problem. *Journal of Experimental Algorithmics (JEA)*, 15:1–4, 2010.