Algorithm Selection for Maximum Common Subgraph

Paulius Dilkas - 2146879

2nd December 2017

1 Status report

1.1 Proposal

1.1.1 Motivation

There are several algorithms for the maximum common subgraph problem, each capable of outperforming the others for some problem instances. The purpose of this project is to create a machine learning (ML) algorithm portfolio that can select the best algorithm for each pair of graphs. Furthermore, the insights extracted from the ML model could be used to provide human-understandable information about which algorithm favours what kind of graphs and allow for meta-algorithms that can switch what algorithm they are using in the middle of execution.

1.1.2 Aims

This project will generate data by running all (currently 4) algorithms on close to 100000 problem instances for 3 different types of graph labelling: no labels, vertex labels, and both vertex and edge labels. For the last two cases, the number of vertices/edges assigned to each label will also be varied. Then an ML model will be trained for each of the 3 cases. Their prediction quality will be evaluated together with comparing the portfolios with the original algorithms. Finally, smaller ML models will be considered to provide human-understandable high-level view of how the algorithms compare.

1.2 Progress

- A Makefile created to support running multiple experiments in parallel.
- Regular expressions (with `sed`) created to parse each algorithm's output into a row of a CSV file.
- Graph feature extractor modified to support a different file format.
- Selected features to be used in ML, generated data about them, plotted their distributions.
- One of the algorithms ($k \downarrow$) extended to support vertex labels.
- R scripts developed to clean and check the data, run the ML model, and plot/output various results.

1.3 Problems and risks

1.3.1 Problems

- One of the algorithms (the clique encoding) turned out to use too much memory on some instances. Upper bounds on graph sizes were added to the Makefile and `ulimit` was used to limit memory usage.
- One of the FATA nodes keeps crashing, and there is no easy way to continue halfway-done experiments without additional development.

1.3.2 Risks

- There might be too much data to train an ML model (a model for unlabelled instances reached over 300 GB RAM). **Mitigation:** take samples from the data, reduce the number of trees in a random forest, limit the size of a tree.
- Generating data might take too long. Worst case estimate is the end of January. **Mitigation:** train a model on a subset of data. Plots and numbers can be updated later.

1.4 Plan

Semester 2

- Week 1–2: descriptions of how the algorithms perform in each of the 3 cases.
  - **Deliverable:** cumulative plots, heatmaps, tables.
- Week 3–5: statistical analysis of the quality of ML models.
  - **Deliverable:** convergence plots, variable importance/usage measures, margin plots and histograms of margins for each algorithm.
- Week 6–7: compare the portfolios with individual algorithms.
  - **Deliverable:** cumulative plots, PAR10 scores, other statistics.
- Week 8: experiment with small ML models (perhaps a single decision tree).
  - **Deliverable:** important features and their critical values identified, suggestions for future work.
- Week 9–10: writing.
  - **Deliverable:** conclusion, expanded introduction, old plots updated with latest data.