Advanced Graph Algorithms

Jan Dreier, Philipp Kuinke,
Peter Rossmanith

Lehr- und Forschungsgebiet Theoretische Informatik

Overview

Organisation

Libraries

Algorithms

The Topic

Graph algorithms are an important tool to solve problems and Graph Libraries provide a framework for them. Open source gave rise to a lot of tools and it is time to give something back.

The Topic

The goals of this practical course are as follows:

- Improve on your teamwork,
- improve your programming skills,
- learn new and advanced graph algorithms, and
- familiarize yourself with contributing to active open source projects

And hopefully: Extend existing graph libraries with new algorithms.

What we expect

Since this is a Masters level course we expect you already have the following

- strong foundation in theoretical computer science
- knowledge of basic graph algorithms (dfs, shortest-path, spanning tree, flows, etc.)
- solid programming skills
- familiarity with git
- strong independence

Timeline

- Today you will form two teams.
- In three weeks you have selected a library and familiarized yourself with it. You should know exactly which algorithms are implemented and which are not (this includes checking pull requests!)
- Present (and explain!) the library and the first algorithm(s) you want to implement.
- In regular presentations you will tell us and the others about your progress and receive feedback.

Workflow

In each team has to

- understand a complicated algorithm
- implement it properly
- adhere to the contribution guidelines of your library
- write extensive tests and debug your code
- make your code as efficient as possible (profiling and optimization using callgrind, gprof, etc)
- write helpful documentation
- communicate with maintainer
- present your progression to the other teams
- submit a pull request to your library
- react to changes requested by maintainer
- start over with another algorithm

Meetings

- Tell us and the other team
  - what you have done
  - what you are working on
  - what you plan to do
  - what the difficulties are
  - what your long-term goals are
- Everybody needs to present
- Tuesdays 14:15-15:45

Source Control

We recommend using Github or Gitlab

- Github: fork library directly and develop in the open
- Gitlab: unlimited private repos
- give us read access

Libraries

The choice of the library is down to personal preference.

1. Boost (C++)
2. igraph (C)
3. JgraphT (Java)
4. Networkx (Python)
5. Something else?

Boost

- Language: C++
- Website: `http://www.boost.org/doc/libs/1_66_0/libs/graph/doc/`
- Content: `www.boost.org/doc/libs/1_66_0/libs/graph/doc/table_of_contents.html`
- Repo: `https://github.com/boostorg/graph`

Notes:

- most popular open source graph library for C++
- Mature, fast and flexible
- Template based

Notable Missing algorithms:

- treewidth decompositions
- centrality measures
- planar separators
- ...

igraph

- Language: C
- Website: `http://igraph.org/`
- Manual: `http://igraph.org/c/doc/`
- Repo: `https://github.com/igraph/igraph`

Notes:

- Collection of network analysis tools with focus on efficiency and portability
- less general and more focused than boost
- Interface for python and R
- maintainer seems busy
- in C (memory allocation, pointers,. . .)

JgraphT

- Language: Java
- Website: `http://jgrapht.org/`
- Repo: `https://github.com/jgrapht/jgrapht`

Notes:

- Has a wiki: `https://github.com/jgrapht/jgrapht/wiki`
- Clear contribution guidelines
- Good documentation
- Very object oriented (every Algorithm is a class)

Notable Missing algorithms:

- Planarity algorithms

Networkx

- Language: Python
- Website: `https://networkx.github.io/`
- Repo: `https://github.com/networkx/networkx`

Notes:

- Excellent documentation
- Very active community
- Python is slow compared to C++ and Java
- Code is easily readable

Notable Missing algorithms:

- Exact algorithms for NP-hard problems

Something Else

You can choose another graph library but it has to be well-maintained and in use! Talk to us if you want to do that.

Algorithms

(some ideas)

Simple FO Model-Checking

- given graph G and FO-formula φ, decide whether $G \models \varphi$
- in time $n^{|\varphi|}$ via branching
- possible to add some straightforward pruning rules
- an easier project to get started

Simple Heuristics for Treewidth Decomposition

- computing an optimal treewidth decomposition is hard, but a greedy strategy often leads to good results.
- for algorithms see slide *Possible Heuristics on Upper Bounds* `http://web.eecs.utk.edu/~cphillip/cs594_spring2015_projects/treewidth.pdf`
- an easier project to get started

Centrality Measures

- there are various centrality measures. They all try to identify the most important vertices within a graph.
- boost only has edge betweenness centrality
- but there are many more
  `https://en.wikipedia.org/wiki/Centrality`

Two Vertex-Disjoint Paths

- A linear-time algorithm that does not need a planar embedding is presented for the problem of computing two vertex-disjoint paths, each with prescribed endpoints, in an undirected 3-connected planar graph.
- paper: Hagerup, A Very Practical Algorithm for the Two-Paths Problem in 3-Connected Planar Graphs `https://link.springer.com/chapter/10.1007/978-3-540-74839-7_14` (Behind Springer wall. Ask us if you cannot access it)

Faster Maximum Flow Algorithms

- many libraries implement Edmonds-Karp, which runs in time $O(VE^2)$.
- By making a case distinction $O(VE)$ is possible, see: `https://en.wikipedia.org/wiki/Maximum_flow_problem#Solutions`

Weighted Flow Algorithms

- many libraries have flow algorithms, but few have algorithms for weighted flow problems, e.g., min cost max flow.
- some are listed at `https://en.wikipedia.org/wiki/Circulation_problem#Related_problems`
- these problems can often be solved easily using LPs. Can we achieve competitive performance with a more lightweight approach?

Planarity

- Planarity testing
- Planar embedding
- Planar Separator
- Planar graph generator

FPT

- Consider algorithms which run in time $f(k)n^{O(1)}$ for some parameter k
- Example: Find vertex cover of size k in time $2^k n$
- many algorithms only feasible for very small k
- More fine-grained than P and NP
- missing in almost every library
- We have many resouces for you if you are interested in this topic (e.g., Parameterized Algorithms by Cygan et al.)

Random Graphs and Complex Networks

- Community Graph Generators with ground truth (e.g. LFR benchmark)
- Kleinberg-, Chung-Lu-, Configuration-Model