Project 442-5
Finding Connected Components

Thomas Nowak
thomas.nowak@polytechnique.edu

Version for X2017

1 Introduction

The most basic way of partitioning a graph is splitting it up into its connected components. More advanced techniques are community detection or identification of the densest subgraph, which you will see later in this course. Still, the fundamental question of connected components often comes up, in particular when analyzing the basic computational capabilities of a network, as connectivity is a necessary condition for many tasks and services. The question gets even more interesting for *directed* graphs (digraphs) since many real-life networks such as social influence graphs and wireless communication networks are inherently directed. Also, one has to distinguish between weak and strong connectivity. From a computational point of view, oftentimes strong connectivity is the right generalization from undirected to directed graphs.

A digraph $G = (V, E)$ is strongly connected if it contains a path from u to v for all nodes $u, v \in V$. A sub-digraph of G is a *strongly connected component* of G if it is strongly connected and maximal with this property. Any digraph has a unique decomposition into strongly connected components, which partitions the set of nodes.

2 Data Sets

You'll use the following collections of real-life data to run your program on.

- SNAP: `http://snap.stanford.edu/data/index.html`
- CRAWDAD: `http://www.crawdad.org/`

The SNAP collection has large-scale graphs, mostly coming from social networks, many of which are directed (think the follows-relation on Twitter, which is not symmetric). The CRAWDAD collection has small to medium-scale graphs of snapshots of real-life wireless networks. You need to sign up on their website to get access to CRAWDAD, but sign-up is free and uncomplicated. In both collections there are some undirected data sets, which you can transform into directed data sets by randomly assigning a direction to every edge. Also, some data sets contain non-Boolean distance information (in terms of geometrical distance or signal strength), which enables some more refined analysis techniques.

3 Strongly Connected Components

Because two edges are enough to merge two strongly connected components, any algorithm needs to look at $\Omega(|E|)$ edges and thus takes sequential time $\Omega(|E|)$. Repeated application of Dijkstra's algorithm leads to an $O(|V|^3)$ algorithm, but actually calculates all distances and shortest paths as well, which we don't need (there is a path from node u to node v if and only if the distance from u to v is not $+\infty$). The more involved algorithm by Tarjan [2] is actually time-optimal in that it takes sequential time $O(|E|)$, which matches the above lower bound.

Question 1. Find and implement a sequential algorithm that identifies the strongly connected components of a digraph. Estimate its time complexity. Your algorithm doesn't need to be time-optimal.

To test your algorithm, you can use real data sets cited in Section 2. Another source of examples are *random* digraphs. For simplicity, fix a real parameter $p \in [0, 1]$ and include an edge (u, v) with probability p, independently of the other edges. We call them *Erdős-Rényi (ER) digraphs*.

Question 2. Write a program that generates ER digraphs for a given number of nodes n and a given parameter p. Test your program from Question 1 with these digraphs. Play around with different values of p to get multiple strongly connected components, but not only isolated nodes.

4 DBSCAN Algorithm

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm [1] allows to cluster the data points based on non-Boolean distances while eliminating statistical outliers. More specifically, the algorithm is parameterized by two parameters, $\varepsilon > 0$ and $M \in \mathbb{N}$. It outputs a set of *core points* that have at least M other data points at a distance of at most ε. It is particularly useful for data that can be embedded into some geometrical space in which distances have some immediate real-world meaning.

Question 3. Implement the DBSCAN algorithm. Test it with random placements in a 2-dimensional plane and real-life data sets. Specify the largest data sets your algorithm can handle and its performance on those. Compare the results of the DBSCAN algorithm with the strongly connected components (on data where this is possible). Try to come up with improvements to the algorithm for the data sets you studied, in particular with respect to heuristics for the choice of the parameters ε and M.

References

[1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Evangelos Simoudis, Jiawei Han, and Usama Fayyad (eds.), *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press, Menlo Park, 1996.

[2] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1(2):146–160, 1972.