Project INF442-5

# Finding Connected Components

Contact: Martin (martin.krejca@polytechnique.edu)

Conceptual design by Thomas Nowak

## 1 Introduction

Graphs can be partitioned in a multitude of ways. More advanced techniques include community detection and the identification of densest subgraphs. However, such approaches are not always necessary. Sometimes, simply working with *connected components* suffices, in particular when analyzing the basic computational capabilities of a network, where connectivity is a necessary condition for many tasks. Further, since many real-world networks, such as social or wireless-communication networks, are inherently directed, determining connected components for *directed* graphs *(digraphs)* is especially interesting.

For digraphs, one typically distinguishes between *weak* and *strong* connectivity. In this project, we are concerned with the latter, as it is oftentimes the more desirable property. Formally, a digraph $G$ with vertices $V$ is strongly connected if and only if for all $u, v \in V$, there exists a (directed) path in $G$ from $u$ to $v$. In other words, each vertex of $G$ is reachable from any other vertex.

Since strong connectivity for an entire digraph $G$ is a rather strong requirement, we are rather interested in sub-digraphs of $G$ that are strongly connected. We call each *induced* sub-digraph of $G$ that is strongly connected and cannot be extended to a larger strongly connected sub-digraph a *strongly connected component* of $G$. Note that each digraph has a unique decomposition into strongly connected components, which partitions the set of vertices.

In this project, you are tasked with implementing algorithms that utilize strongly connected components (Sections 2 and 3). Each algorithm should be run on various data sets (Section 4).

## 2 Strongly Connected Components

There are various ways to determine the strongly connected components of a graph $G = (V, E)$ sequentially. For example, one computes for each vertex $v \in V$ the set of all vertices that $v$ is connected to via depth-first search. This yields all vertices for which there exists a path *from* $v$. Then, one computes the vertices that $v$ is connected to in the *transposed graph* of $G$, which yields all vertices that have a path *to* $v$. Last, one checks which vertices are in both sets, which overall returns the strongly connected component of $v$ in time $\mathrm{O}(|V| + |E|)$. Repeating this approach for vertices that are not in the strongly connected component of $v$ results in an algorithm for finding all strongly connected components of $G$.

A similar but more involved algorithm was proposed by Tarjan [Tar71], which manages to compute *all* strongly connected components in time $\Theta(|V| + |E|)$, that is, the time for a single depth-first search.

**Task 1.** Implement and test a sequential algorithm that identifies the strongly connected components of a given digraph. Estimate its time complexity. Your algorithm does not need to be as fast as Tarjan's algorithm.

Next to the data sets from Section 4, *random* digraphs are another interesting source for test cases. We consider the *Erdős–Rényi (ER) digraph* model, which is given a value $p \in [0, 1]$ as well as a vertex set $V$ and constructs a (random) digraph $(V, E)$ by adding each edge $(u, v) \in V^2$ to $E$ with probability $p$.

**Task 2.** Write a program that generates ER digraphs for a given number of vertices $n \in \mathbf{N}_{>0}$ and a given parameter $p \in [0, 1]$. Test your program from Task 1 with these digraphs. Play around with different values of $n$ and $p$ in order to get multiple strongly connected components but not only isolated vertices.

# 3 The DBSCAN Algorithm

As briefly touched on in Section 1, there are other interesting ways to learn more about the structure of a graph. We consider clustering via the DBSCAN algorithm [EKS+96]. This algorithm clusters its data based on (non-Boolean) distances while eliminating statistical outliers. More specifically, given a data set with distances as well as the input parameters $\varepsilon \in \mathbf{R}_{>0}$ and $M \in \mathbf{N}$, the algorithm returns a set of *core points* that have at least $M$ other data points at a distance of at most $\varepsilon$. This approach is particularly useful for data that can be embedded into some geometrical space in which distances have some immediate real-world meaning.

**Task 3.** Implement and test the DBSCAN algorithm with random placements of points in a 2-dimensional Euclidean plane as well as with real-world data sets (Section 4). Specify the largest data sets your algorithm can handle, and report its performance on those. Compare the results of the DBSCAN algorithm (on data where this is possible) with the strongly connected components of the data set. 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 $\varepsilon$ and $M$.

# 4 Data Sets

When testing your algorithms, please choose graphs from the following sources:

- 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-world wireless networks. You need to sign up on their website in order 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.

# References


[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Proceedings of KDD'96*. 1996, pp. 226–231. URL: https://dl.acm.org/doi/10.5555/3001460.3001507 (see page 2).

[Tar71] Robert Tarjan. "Depth-first search and linear graph algorithms." In: *Proceedings of SWAT'71*. 1971, pp. 114–121. DOI: 10.1109/SWAT.1971.10 (see page 1).