Project 442-8
Finding Strongly Connected Components*

Thomas Nowak
thomas.nowak@polytechnique.edu

X2015

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.

3 Sequential Algorithm

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 [1] 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.

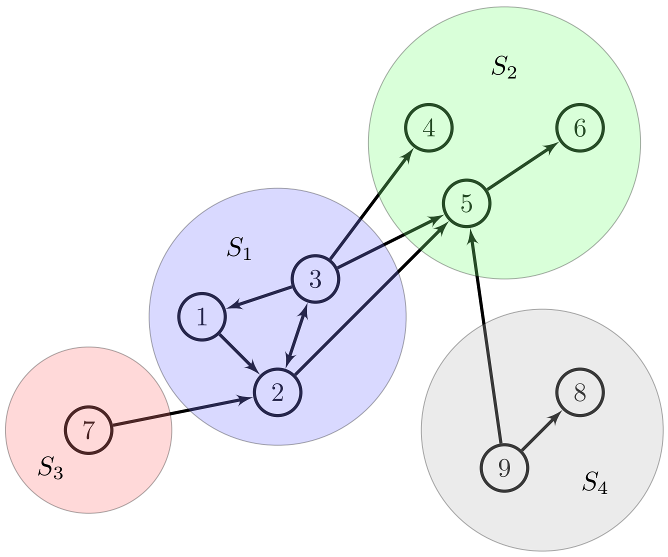

Figure 1: The sets S_1, S_2, S_3, and S_4 for pivot node $v = 1$

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 Parallel Algorithm

As a warm-up exercise for the parallel part of this project, we note that the independence of edges in ER digraphs lends itself to parallelization.

Question 3. Write a parallel program that generates ER digraphs.

We'll now describe a parallel algorithm to find the strongly connected components of a digraph [2, 3]. The basic idea is to pick any node v and then partition the nodes of the digraph G into four sets:

- $S_1 = \mathrm{Succ}_G(v) \cap \mathrm{Pred}_G(v)$
- $S_2 = \mathrm{Succ}_G(v) \setminus \mathrm{Pred}_G(v)$
- $S_3 = \mathrm{Pred}_G(v) \setminus \mathrm{Succ}_G(v)$
- $S_4 = V \setminus \big(\mathrm{Succ}_G(v) \cup \mathrm{Pred}_G(v)\big)$

Here, $\mathrm{Succ}_G(v)$ is the set of successor nodes of v, i.e., the nodes that are reachable from node v in G and $\mathrm{Pred}_G(v)$ is the set of predecessor nodes of v, i.e., the set of nodes from which node v is reachable in G. See Figure 1 for an example.

The set S_1 is equal to the set of nodes of the strongly connected component of G that includes node v. Any other strongly connected component has to be fully included in either of the sub-digraphs described by S_2, by S_3, or by S_4. We can thus recursively start the algorithm on these three digraphs and be sure that we only find strongly connected components of the original digraph.

Question 4. Implement the above parallel algorithm. Questions that were left open in the description are how to calculate the sets $\mathrm{Succ}_G(v)$ and $\mathrm{Pred}_G(v)$, and how to choose the pivot node v. A random choice is possible, but one can also try to aim for a node in a small strongly connected component. Try to come up with other improvement to the algorithm. Test it with ER digraphs and real-life data sets. Specify the largest data sets your algorithm can handle and its performance on those.

References

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

[2] Lisa K. Fleischer, Bruce Hendrickson, and Ali Pınar. On identifying strongly connected components in parallel. In: José Rolim et al. (eds.) *Proceedings of the 15 IPDPS 2000 Workshops*, Lecture Notes in Computer Science vol. 1800, Springer, Heidelberg, 2000, pp. 505–511.

[3] William McLendon III, Bruce Hendrickson, Steven J. Plimpton, and Lawrence Rauchwerger. Finding strongly connected components in distributed graphs. *Journal of Parallel and Distributed Computing* 65(8):901–910, 2005.