Finding connected components for massive graphs

Database Management

Data Science for Business – École Polytechnique

Ching-yu Lin

Fernando Pérez Fernández

Jiahao Wang

Table of Contents

[1. Introduction 3](#_Toc2026139)

[2. Finding Connected Component Problem 4](#_Toc2026140)

[2.1. Cracker Algorithm 4](#_Toc2026141)

[Seeds Identification 4](#_Toc2026142)

[Seeds Propagation 6](#_Toc2026143)

[2.2. Algorithm implementation in Python 6](#_Toc2026144)

[2.3. Algorithm implementation in Spark 6](#_Toc2026145)

[3. Other Algorithm implementation 7](#_Toc2026146)

[4. Experiment performed 7](#_Toc2026147)

[5. Conclusion 8](#_Toc2026148)

# Introduction

The task of our project is to implement an algorithm to find connected components in massive graphs. The algorithm implemented is described in a recent scientific paper: *Cracker: Crumbling Large Graphs Into Connected Components1.* It proposes a distributed algorithm for the discovery of connected components in large graphs.

In this report we will first briefly present the connected graph problem in massive graphs and describe the solution adopted by the previous paper. Second, we will implement two different solutions by using PySpark: one using RDD and the other one using DataFrames. Third, we will explained a different algorithm implemented by Google Research in 2014 on the paper: *Connected Components in MapReduce and Beyond2* and the default library in Spark GraphFrames*.*

Finally, we will perform different experiment using different size graphs coming from Blockchains for Bitcoin transacions: 2MB, 5MB, 10MB and 50MB in order to evaluate the performance and scalability of each implemented algorithm.

1GB, 420MB, 80MB, 40MB. 8MB (size of raw data)

As a conclusion, the results obtained with all the different implementations are analyzed and commented.

# Finding Connected Component Problem

The problem of finding connected components in a graph is common to several applications, such as image processing, Blockchain transactions or social network analysis. The size of graphs is increasing exponentially is the past years and it is crucial to define distributed algorithms for the discovery of connected components in large graphs.

## Cracker Algorithm

Let’s define an undirected graph such as where V is a set of vertices uniquely identified and E is the set of edges, having that Our task is to identify a connected component (CC) in G, defined as a maximal subgraph such that exist an undirected path in S for any two vertices .

Let’s explain the cracker algorithm with one small graph with just one Connected Component with ID = 0 (the smallest Vertex).



Figure 1: Initial Graph G1

The cracker algorithm uses two different phases to achieve the identification of the CCs: seed identification and seed propagation.

### Seeds Identification

The goal of this step is to have one single vertex for each connected component by progressively pruning vertex. As soon as one Connected Component is discovered, it is excluded for the computation (for instance one isolated component).

At the beginning of the algorithm, all vertices are marked as active. Two steps are presents in the stage:

1. **Min Selection Step**. The goal is to define for each node if it is a potential seed by one of the neighbors. From the initial graph G1, an intermediate graph H1 is created, linking all potential seeds. A node can be considered as a potential seed if it is a minimum in the neighborhood of any node. For instance, in the previous graph, the node 2 can be considered as a potential seed as it is the local minimum of the neighborhood of 5 (2 and 8).

In order to identify potential seeds, the local minimum of the neighborhood, defined as , is spread by each node to its neighborhood. From the graph Gt an intermediate graph Ht is created by adding new edges from all the nodes of the neighborhood of u and u itself to.

In the following figure, the graph H1 is generated from the graph G1 by adding the new edges as previously mentioned. For instance, the neighborhood of 5 are 2 and 8. In this neighborhood, is equal to 2. This is spread to all the nodes in the neighborhood (2, 5 and 8) and the edges 2→2, 5→2 and 8→2 are created. This is done for all the active nodes.



Figure 2: Creating of H from G

1. **Prunning step.**  In this step the aim is to create a new graph Gt+1 from Ht by excluding nodes that are not seeds and to add them to the propagation tree *T*. For all the nodes in Ht, is recomputed. Then, for each node in the neighborhood (except ) a new undirected edge is added to the graph Gt+1. For instance, in H1, the nodes *u* in the neighborhood of 0 are 1 and 2. In this neighborhood, is node 0. Therefore, an undirected link 1↔0 and 2↔0 is created in G2.

A node is not seed if it hasn’t self-link in Ht. Consequently, all the nodes that have no self-links in Ht are excluded. In other words, nodes that are not a local minimum by any other nodes is excluded. In this case, as we can observe in the following figure, nodes 4, 6, 7 and 8 are excluded in G2. The nodes excluded are inserted into the propagation tree T.



Figure 3

Finally, as we can observe in the Figure 4, a node is recognised as a seed if it is still active, but it is the only node in its neighbourhood (node 0 in this case). Then it is marked for deactivation to complete the *Seed Identification* phase.



Figure 4: Finalised Seed Identification Phase

### Seeds Propagation

During the Pruning Step described previously, a spanning tree *T* for each connected component is built by adding excludes nodes. When a vertex is excluded, a directed edge is added to *T*. After finishing the *Seed Identification* phase, for each Connected Component exist a tree rooted in its seed node, in the previous case the node 0. Such tree is used to propagate the seed identifier to all the nodes in the tree.

Before going the algorithm implementation, let’s understand better how this spanning tree is built. Looking again at the example above, as we go from graph H1 to G2, nodes 4, 6, 7 and 8 are excluded. Therefore, as we can see in the figure 5, a directed edge is created: 3 → 4, 3 → 6, 3 → 7 and 2 → 8. After, going from graph H2 to G3, nodes 3 and 5 are excluded, so the directed edges 0 → 3 and 0 → 5 are created. Finally, when the nodes 1 and 2 are excludes, the directed 0 → 1 and 0 → 2 are created. The seed of the Connected Component is 0.



Figure 5: Tree

* Dataset presentation:

|V|, |E|, #CC etc.

* Preprocessing : uuid 🡪integer
  + Example
  + Code

## Algorithm implementation in Python

## Algorithm implementation in Spark

# Other Algorithm implementation

# Experiment performed

# Conclusion