Procesamiento de Grafos

Daniela Vignau León

30 de octubre de 2019

Abstract

En el presente trabajo, se pretende mostrar al lector cómo realizar el procesamiento de grafos obtenido a partir de un *dataset* de la librería de SNAP.

1.Introduction

De manera general, un grafo está definido como *G = (V, E)* donde V representa a los vértices y E a las aristas. Existen muchos tipos:

- Dirigidos
- No dirigidos
- Con peso
- ... y muchos otros más.

Los grafos tienen aplicaciones en muchísimas áreas que permiten resolver diversos problemas; como modelar trayectos, administración de proyectos, encontrar los caminos más cortos u óptimos e inclusive para el estudio de las redes sociales.

A esto último se le conoce como *Análisis de redes sociales* donde se mapean las relaciones entre personas, grupos, organizaciones, URLs y otros tipos de información. Los vértices representan a las personas y los grupos, y las aristas muestran la relación entre los nodos.

Para el presente trabajo, se utilizó la libería de SNAP (Standford Network Analysis Platform) la cual permite la fácil manipulación y análisis de redes grandes en lenguajes de programación como C++ y Python. Es importante notar que ésta debe de estar previamente descargada e instalada. Asimismo se descargó un *dataset* proporcionado por SNAP el cual se puede descargar aquí.

2. Importación del grafo

Después de haber seleccionado el dataset con el que se quiere trabajar, se debe de descargar y extraer el archivo que tiene la extensión *.txt.gz* a la carpeta que contendrá el proyecto. En este caso se seleccionó la base de datos de **wiki-Vote**, si se desea descargar otro dataset, se tendrán que hacer unas modificaciones mínimas.

Dentro del proyecto, se deberá crear un gráfo para poder llamarlo en los diferentes métodos. Dentro del main, se deberá insertar la siguiente línea:

```
Dgraph graph =
    TSnap::LoadEdgeList<DGraph>
    ("wiki-Vote.txt", 0, 1)
```

3. Creación de métodos

El usuario deberá de crear los métodos para la exportación del grafo en los diferentes formatos que se requieran. De manera general, se deberá de abrir un archivo con la extensión requerida para escribir el grafo exportado. Se debe de tener mucho cuidado en cómo leen los formatos los grafos, pues es ahí donde radica la manera en la que se ve o no el grafo.

4. Resultados

Con el simple propósito de comparar uno u otro formato, se calculó el tiempo de ejecución en milisegundos de cada uno de los métodos. Los resultados fueron:

Formato	**Tiempo de Ejecución**
GraphML	108.674 ms
GEXF	116.105 ms
GDF	96.175 ms
JSON	104.142

Table 1: Tabla de tiempos de ejecución

En este caso se decidió graficar el formato GraphML para poder visualizar la gráfica en un software llamado *Gephi*. Se debe de descargar el software, inicializarlo y seleccionar qué archivo se quiere abrir. Una vez realizado esto, el grafo será creado y se verá de la siguiente manera:

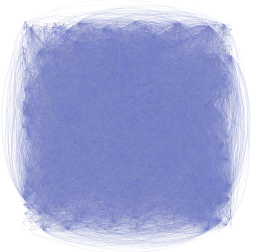

Figure 1: Grafo del dataset de wiki-Vote

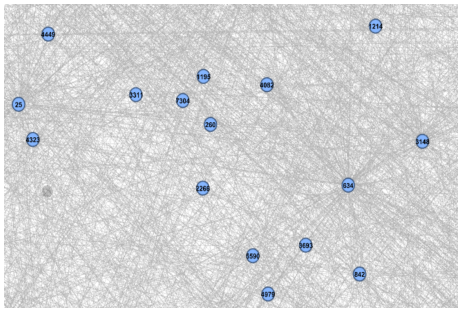

Figure 2: Acercamiento del mismo grafo

5. Interpretación de resutlados

Con respecto a los tiempos de ejecución, se puede apreciar que *GEXF* fue el formato más lento para exportar el grafo, y el más rápido fue *GDF*.

Se pueden obtener datos interesantes con respecto al grafo utilizando el software anteriormente mencionado. Algunos de ellos son: el camino promedio entre los nodos es de 3.341, el coeficiente promedio de agrupamiento de nodos es de 0.081, la densidad del grafo es de 0.002 y el grafo tiene una modularidad de 0.43. En la figura 2, se muestra un acercamiento del grafo para que el lector pueda, de cierta manera, apreciar cómo es que se ven los vértices conectados así como

las etiquetas de cada uno de los nodos.

6. Ventajas y desventajas

Sobre los formatos

GraphML: *Ventajas:* Permite añadir atributos a vértices y aristas, grafos jerárquicos. *Desventajas:* No soporta sub-grafos ni *hiper bordes.*

GEXF: *Ventajas:* Permite especificar atributos dados por el usuario (como peso de los nodos y direcciones de las aristas). *Desventajas:* Es un archivo pesado.

GDF: *Ventajas:* Permite añadir atributos a vértices y aristas, el documento está dividido entre esos dos. Fácil de leer y de convertir a CSV. *Desventajas:* Puede llegar a ser un poco pesado,

JSON: *Ventajas:*:El esquema es muy flexible, fácil de leer y dado que es map-based facilita el envío de datos en aplicaciones Web, por ejemplo. *Desventajas:* No es muy práctico utilizarlo para aplicaciones Web ya que puede duplicar información.

La complejidad temporal de los cuatro formatos es de O(V + E), donde V se refiere a los vértices (nodos) y E a las aristas y todos cuentan con una complejidad espacial de S(1).

Sobre la visualización de grafos

Quizás una de las ventajas más grandes de poder visualizar un cierto grafo, en Gephi por ejemplo, es poder darse una idea sobre la distribución de los datos e inclusive se podría llegar a ciertas conclusiones con esto. Otra de las grandes ventajas, es que es realmente fácil encontrar datos como los que se mencionaron en la sección 5: camino promedio entre nodos, coeficiente de agrupamiento, etc.

Una gran desventaja es que es difícil manipular cientos de miles de datos y que a menos que el usuario tenga una buena idea sobre qué quiere saber del grafo, podrá hacer uso de todas las herramientas. Puede llegar a suceder que se deben de descargar más softwares para visualizar el grafo, porqe al menos Gephi, no permite el formato JSON.

7. Código

```
// Exportando a GraphML
void GraphML(DGraph graph) {
  ofstream file ("wiki_vote.graphml");
  if (file.is_open()) {
      file << "<?xml version=\"1.0\"
          encoding=\"UTF-8\"?>\n";
      file << "<graphml
          xmlns=\"http://graphml.
      graphdrawing.org/xmlns\"
          xmlns:xsi=
      \"http://www.w3.org/2001/
      XMLSchema-instance\"xsi:schemaLocation
      =\"http://graphml.graphdrawing.org/
      xmlnshttp://graphml.graphdrawing.org
      /xmlns/1.0/graphml.xsd\">\n";
      file << "<graph id=\"G\"
          edgedefault=\"directed\">\n";

      for (DGraph::TObj::TNodeI NI =
          graph->BegNI(); NI <
          graph->EndNI(); NI++)
         file << "<node id=\"" <<
            NI.GetId() << "\"/>\n";

      int i = 1;
```

```
      for (DGraph::TObj::TEdgeI EI =
          graph->BegEI(); EI <
          graph->EndEI(); EI++, ++i)
      file << "<edge id=\"e" << i <<
          "\" source=\"" <<
          EI.GetSrcNId() << "\"
          target=\"" << EI.GetDstNId()
          << "\"/>\n";
      file << "</graph>\n";
      file << "</graphml>\n";
      file.close();
   }
}

//Exportando a GEXF
void GEXF(DGraph graph) {
   ofstream file ("wiki_vote.gexf");
   if (file.is_open()) {
      file << "<?xml version=\"1.0\"
          encoding=\"UTF-8\"?>\n";
      file << "<gexf
          xmlns=\"http://www.gexf.net/1.2draft\"
          version=\"1.2\">\n";
      file << "<graph mode=\"static\"
          defaultedgetype=\"directed\">\n";

      file << "<nodes>\n";
      for (DGraph::TObj::TNodeI NI =
          graph->BegNI(); NI <
          graph->EndNI(); NI++)
         file << "<node id=\"" <<
            NI.GetId() << "\" />\n";
      file << "</nodes>\n";

      file << "<edges>\n";
      int i = 1;
      for (DGraph::TObj::TEdgeI EI =
          graph->BegEI(); EI <
          graph->EndEI(); EI++, ++i)
         file << "<edge id=\"" << i <<
             "\" source=\"" <<
             EI.GetSrcNId() << "\"
             target=\"" <<
             EI.GetDstNId() << "\" />\n";
      file << "</edges>\n";

      file << "</graph>\n";
      file << "</gexf>\n";
      file.close();
   }
}

//Exportando a GDF
void GDF(DGraph graph) {
   ofstream file ("wiki_vote.gdf");
   if (file.is_open()) {
      file << "nodedef>id VARCHAR\n";
      for (DGraph::TObj::TNodeI NI =
          graph->BegNI(); NI <
          graph->EndNI(); NI++)
         file << NI.GetId() << "\n";

      file << "edgedef>source VARCHAR,
          destination VARCHAR\n";
      for (DGraph::TObj::TEdgeI EI =
          graph->BegEI(); EI <
          graph->EndEI(); EI++)
         file << EI.GetSrcNId() << ", "
            << EI.GetDstNId() << "\n";

      file.close();
   }
}

//Exportando a JSON
void JSON(DGraph graph) {
  ofstream file ("wiki_vote.json");
   if (file.is_open()) {
      file << "{ \"graph\": {\n";
      file << "\"nodes\": [\n";
```

```
    for (DGraph::TObj::TNodeI NI =
        graph->BegNI(); NI <
        graph->EndNI();;) {
       file << "{ \"id\": \"" <<
           NI.GetId() << "\" }";
       if (NI++ == graph->EndNI())
          file << " ],\n";
       else
          file << ",\n";
    }

    file << "\"edges\": [\n";
    for (DGraph::TObj::TEdgeI EI =
        graph->BegEI(); EI <
        graph->EndEI();;) {
       file << "{ \"source\": \"" <<
           EI.GetSrcNId() << "\",
           \"target\": \"" <<
           EI.GetDstNId() << "\" }";
       if (EI++ == g->EndEI())
          file << " ]\n";
       else
          file << ",\n";
    }
    file << "} }";
    file.close();
  }
}
```

El código para realizar este proyecto puede ser encontrado dando click aquí.

References

[1] *Jbmusso. 2016. GraphSON Reader and Writer Library. Github. Retrieved October 25, 2019 from https://github.com/tinkerpop/blueprints/wiki/GraphSON-Reader-and-Writer-Library*

[2] *Gephi. GraphML Format. Retrieved October 25, 2019 from https://gephi.org/users/supported-graph-formats/gdf-format/*

[3] *FileInfo. GEFX file Extension. Retrieved October 25, 2019 from https://fileinfo.com/extension/gexf .*

[4] *FALCOR. JSON Graph. Retrieved October 25, 2019 from https://netflix.github.io/falcor/documentation/jsongraph.html*

[5] *Gephi. GDF Format. Retrieved October 25, 2019 from https://gephi.org/users/supported-graph-formats/gdf-format/*

[6] *Tim Sheard. 2009. Graphs in Computer Science. Retrieved October 25, 2019 from http://web.cecs.pdx.edu/ sheard/-course/Cs163/Doc/Graphs.html/*

[7] *Orgnet. Social Network Analysis: An Introduction. Retrieved October 25, 2019 from http://www.orgnet.com/sna.html*