Tarea No. 4. Procesamiento de Grafos

SEBASTIAN GONZALO VIVES FAUS, Tecnológico de Monterrey

El alumno aprenderá a trabajar con conjuntos de datos disponibles públicamente y a procesarlos utilizando alguna biblioteca ya existente para la programación de grafos así como visualizarlos con herramientas gráficas para descubrir información relevante. Adicionalmente, aprenderá a crear un artículo de investigación en formato de la ACM utilizando LaTEX.

ACM Reference Format:
Sebastian Gonzalo Vives Faus. 2019. Tarea No. 4. Procesamiento de Grafos. 1, 1 (October 2019), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCCIÓN

Esta investigación tiene como objetivo demostrar paso a paso como importar un dataset (dirigido), de la base de datos de SNAP [3], a un grafo, utilizando C++. Con el grafo obtenido, exportarlo con cuatro funciones, las cuales exportaran el grafo en los siguientes formatos: GraphML, GEXF, GDF y JSON Graph Format. Se obtendrá el tiempo de ejecución que le toma a cada función exportar el grafo en su determinado formato. Una vez exportado los grafos, se utiliza un programa llamado Gephi, para la visualización de uno de los grafos (en este caso, se elegio el formato GraphML). Se obtendrá una imagen, representando el grafo. Finalmente, se utilizará LaTEX (con el template ACM) para demostrar y explicar los resultados obtenidos.

2 PROCESAMIENTO EN C++

2.1 Importación

Lo primero que se hizo fue importar un data set de la base de datos de SNAP (en este caso, en la sección de *Large Network Dataset Collection*) [1]. El data-set obtenido para este ejemplo fue el de las votaciones de Wikipedia, desde que se creó hasta el 2008. Este data-set contiene un total de *7115 Nodos* y *103,689 Vértices*.

Una vez obtenido el data set (archivo .txt), vamos a nuestro código de C++. Utilizamos la función de importar el data set a un grafo (obtenido de la Documentación de SNAP) [4]:

PGraph TSnap::LoadEdgeList(const TStr InFNm, const int SRCColld, const int DstColld)

Author's address: Sebastian Gonzalo Vives Faus, Tecnológico de Monterrey.

XXXX-XXXX/2019/10-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
TSsParser Ss("Wiki-Vote.txt", ssfWhiteSep, true, true, true);
int SrcNId, DstNId, SrcColId = 0, DstColId = 1;

while (Ss.Next()) {
       if (! Ss.GetInt(SrcColId, SrcNId) || ! Ss.GetInt(DstColId, DstNId)) { continue; }
       if (! Graph->IsNode(SrcNId)) { Graph->AddNode(SrcNId); }
       if (! Graph->IsNode(DstNId)) { Graph->AddNode(DstNId); }
       Graph->AddEdge(SrcNId, DstNId);
      }
       Graph->Defrag();
```

Donde tenemos que proporcionarle el *archivo del data set*, al igual que las columnas donde se encuentran el *nodo origen* (SrcColId) y el *nodo destino* (DstColId). Una vez importado, podemos comprobar que la cantidad de nodos y vertices en el grafo coincida con lo indicado en el data set, utilizando la función de SNAP:

```
  void PrintGStats(const char s[], PNEGraph Graph) {
  printf(" %s nodos %d, edges %d, esta vacia? = %s\n",
      s, Graph->GetNodes(), Graph->GetEdges(),
      Graph->Empty() ? "Si" : "No");
  }
```

2.2 Exportación

Ahora que tenemos toda la información del grafo dentro de nuestro programa, podemos empezar a hacer el proceso de *exportación*. El grafo será exportado utilizando cuatro funciones [1], enseñadas a continuación:

- GraphML

```
 void GraphML(PNEGraph Graph) {
//Variables
ofstream file ("Wiki-Vote.graphml"); //Archivo de salida
int i = 1;
//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/xmlns http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd\">\n";
file << "<graph id=\"G\" edgedefault=\"directed\">\n";

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

for (PNEGraph::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();
}
}
```

- GEXF

```
  void GEXF(PNEGraph Graph) {
//Variables
ofstream file ("Wiki-Vote.gexf"); //Archivo de salida
int i = 1;
//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 (PNEGraph::TObj::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++){
file << "<node id=\"" << NI.GetId() << "\" />\n";
}

file << "</nodes>\n";
file << "<edges>\n";

for (PNEGraph::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();
}
}
```

- GDF

```
  void GDF(PNEGraph Graph) {
ofstream file ("Wiki-Vote.gdf"); //Archivo de salida
//GDF:
if (file.is_open()) {
file << "nodedef>id VARCHAR\n";
for (PNEGraph::TObj::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++){
```

```
file << NI.GetId() << "\n";
}

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

file.close();
}
}
```

- JSON

```
  void JSON(PNEGraph Graph) {
ofstream file ("Wiki-Vote.json"); //Archivo de salida
if (file.is_open()) {
file << "{ \"graph\": {\n";
file << "\"nodes\": [\n";
for (PNEGraph::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 (PNEGraph::TObj::TEdgeI EI = Graph->BegEI(); EI < Graph->EndEI(); ){
file << "{ \"source\": \"" << EI.GetSrcNId() << "\", \"target\": \"" << EI.GetDstNId() << "\" }";
if (EI++ == Graph->EndEI())
file << " ]\n";
else
file << ",\n";
}
file << "} }";

file.close();
}
}
```

Cada función genera su propio archivo de salida (*.graphml, .gexf, .gdf, .json*), los cualaes pueden ser utilizados en otros programas de graficación de grafos (en nuestro caso, este será *Gephi*).

Table 1. Resultados de lo Tiempos de Ejecución de las Funciones de Exportación

N. de Intento	T. de Importación (*ms*)	GraphML (*ms*)	GEXF (*ms*)	GDF (*ms*)	JSON (*ms*)
1	49	138	138	33	54
2	67	182	152	33	56
3	44	152	158	46	66
4	54	167	150	38	79
5	43	144	186	53	88
Promedio	*51.4*	*156.6*	*156.8*	*40.6*	*68.6*

2.3 Tiempo de Ejecución

Como parte de esta investigación, es requerido calcular el *tiempo de ejecución* de cada función de exportación, con el fin de comparar cada tiempo y determinar cuál de las cuatro funciones es la más rápida. Para encontrar el tiempo de ejecución de cada función (en *milisegundos*), se utilizaron las siguientes funciones:

```
//Variables de tiempo;
high_resolution_clock::time_point start;
high_resolution_clock::time_point finish;

//Empieza el tiempo
start = high_resolution_clock::now();

//Termina el tiempo
finish = high_resolution_clock::now();

//Obten la duración restando el tiempo final - tiempo inicial
auto duration = std::chrono::duration_cast<std::chrono::milliseconds>( finish - start ).count();
```

Los resultados (probando el código cinco veces y obteniendo el tiempo promedio de cada uno) de las funciones se pueden ver reflejados en la **Tabla** 1.

3 COMPLEJIDAD TEMPORAL

Los cuatro algoritmos comparten comparten la misma complejidad temporal en el mejor y caso promedio (no hay peor caso, ya que sería el mismo que el promedio). En el mejor caso (donde no entra el *if*, lo que significaría que el archivo de salida no estuvisese abierto por alguna razón externa) la complejidad del algoritmo es *O(1)*. En el caso promedio (hay dos *for*, pero ninguno dentro del otro) la complejidad del algoritomo es $\Omega(n)$.

4 ANÁLISIS DEL GRAFO EN GEPHI

Se analizó el grafo exportado por nuestro programa en C++ específicamente la extensión *.gexf*, ya que es el más compatible con Gephi (solo requerimos analizar solo uno de los cuatro métodos de exportación). Dentro del programa de Gephi, seguimos las siguientes instrucciones para importar el grafo dentro del programa:

(1) Abrir el programa de *Gephi* (versión actual: 0.9.2).
(2) En la ventana de inicio (*Welcome to Gephi*), ir a la opción: *Open graph file...* en la sección de *New Project*.

(3) Seleccionar el archivo exportado con extensión *.gexf*.
(4) Aparecerá una ventana de *Importación*, donde nos muestra si el grafo es dirigido o no, al igual que su número de Nodos y Vértices que contiene. Si los datos son correctos, darle a la opción de *OK*.
(5) A continuación, nos aparecerá una visualización de nuestro grafo, donde cada nodo esta conectado respectivo destino (las líneas son los vértices). Dependiendo la cantidad de nodos y vértices (en este caso son 7,115 nodos y 103,689 vértices) será la complejidad de la visualización (entre mas conexiones, más difícil se vuelve de navegar y observar las conexiones.
(6) Se puede ver la representación del grafo utilizado (*Wiki-Vote.gexf*) en la siguiente imágen 1.
(7) Gephi permite al usuario jugar con la gráfica con sus herramientas predeterminadas. Mencionando algunas:
- Puedes mover, seleccionar, incrementar y disminuir el tamaño de la visualización de la gráfica.
- **Painter:** Te permite cambiar el color de las nodos, seleccionandolos con el *mouse izquierdo*.
- **Sizer:** Te permite modificar el tamaño de uno o múltiples nodos.
- **Brush:** Te permite cambiar el color de un nodo y todos los nodos alrededor del nodo pintado.
- **Node Pencil:** Te permite añadir un nuevo nodo.
- **Edge Pencil:** Te permite agregar un vértice entre dos nodos.
- **Shortest Path:** Te permite saber cual es el camino más corto entre dos nodos.
- **Heat Map:** Te permite colocar un mapa de color en un grupo de nodos. El color varía dependiendo el peso (en nuestro caso, no hya peso).
- **Edit:** Te permite editar los atributos de un nodo.
- Tambien te permite cambiar los colores y propiedades del ambiente de la gráfica.
- Por último (entre las funciones que nos interesan), nos permite *exportar* la visualización del grafo en un formato de .gephi e formato de *imágen*.

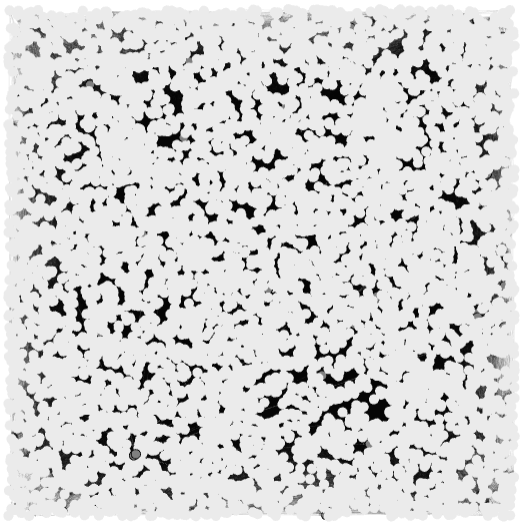

Fig. 1. Representación visual en Gephi

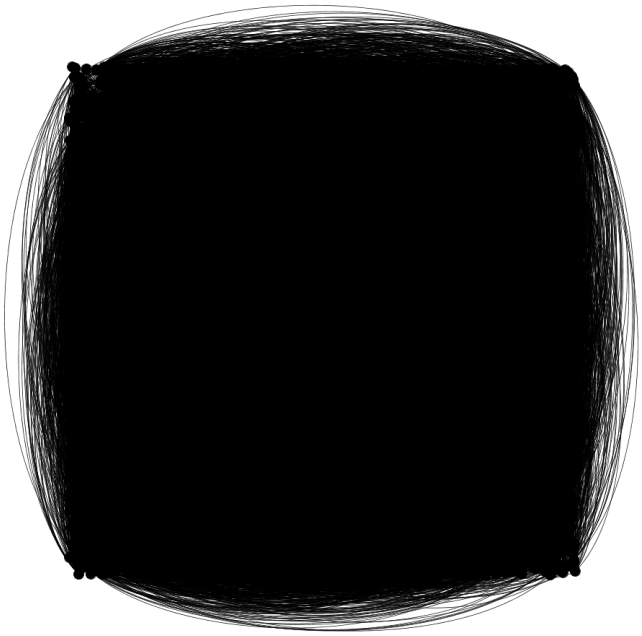

Fig. 2. Archivo de imágen (PNG) exportado de Gephi

5 VENTAJAS Y DESVENTAJAS

- Una de las mayores desventajas del formato JSON es que no es soportado por Gephi, en comparación a los otros tres que sí son soportados. En términos de su tiempo de ejecución [1], tuvo un promedio de *68.6 ms*, poniendolo en el segundo más rápido en comparación a las otras funciones.
- Ya que los otros tres formatos **si** son soportados por Gephi, se muestra en la figura [3] una tabla comparativa de todas las funciones disponibles para cada formato. Como se muestra en la tabla comparativa, la función con más funciones es GEXF, siguiendo GraphML y por último GDF. Por esa razón (entre otras), Gephi recomienda el uso del formato GEXF sobre cuaquier otro [2]. Tambien, el formato GEXF es el único que soporta el uso de pesos *dinamicos*. En términos de tiempos de ejecución [1], fue el más que se tardó (promedio de *68.6 ms*) en exportar el grafo (casi a la par con GraphML).
- Una de las ventajas de utilizar los formatos de GraphML, GEXF y GDF es que, son los únicos que formatos que pueden reconocer la posición de los nodos, su color y el tamaño de sus atributos.

REFERENCES

[1] alberto911. 2015. exportGraphs. Retrieved October 23, 2019 from https://github.com/alberto911/exportGraphs/commit/e61455c521b7b3bcf45a6c677862d65a2fec8bb8
[2] Gephi 2017. Supported Graph Formats. Retrieved October 30, 2019 from https://gephi.org/users/supported-graph-formats/
[3] SNAP 2009. Stanford Large Network Dataset Collection. Retrieved October 28, 2019 from https://snap.stanford.edu/data/index.htmll
[4] SNAPd 2009. SNAP Library 5.0, Developer Reference. Retrieved October 28, 2019 from https://snap.stanford.edu/snap/doc/snapdev-ref/

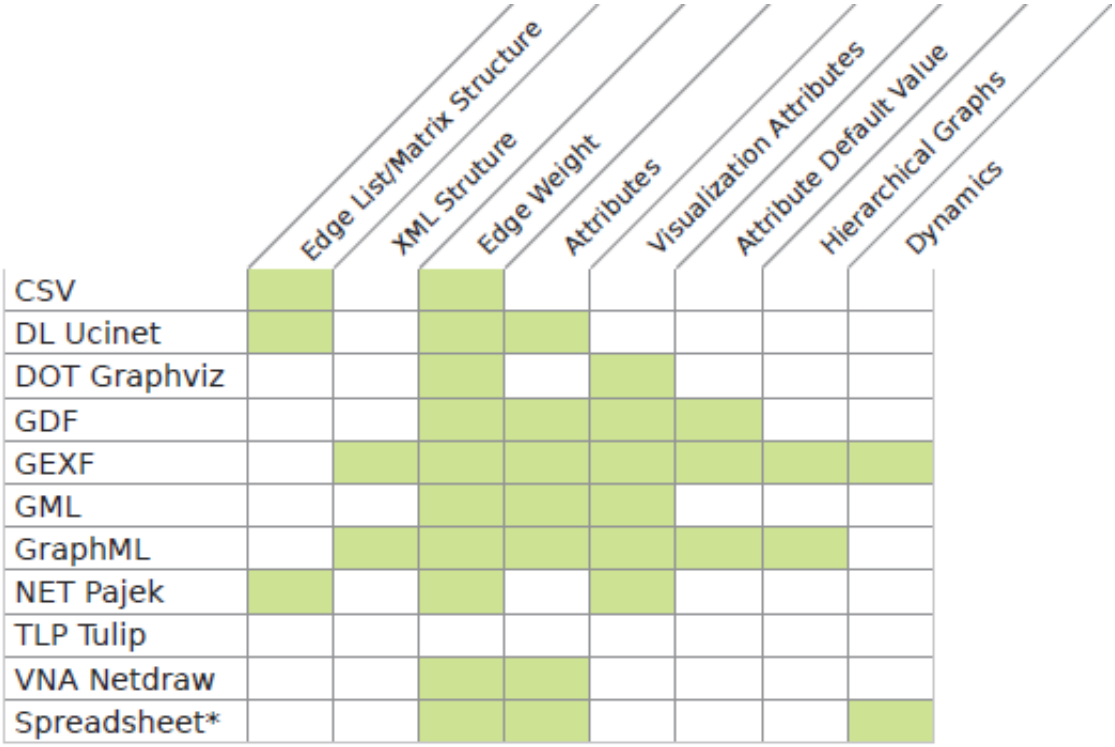

Fig. 3. Tabla comparativa de las funcionalidades disponibles en Gephi de cada formato de grafo. Imagen obtenida de: Supported Graph Formats (https://gephi.org/users/supported-graph-formats/)

6 URL GITHUB

https://github.com/tec-csf/TC2017-T2-Otono-2019-Tlacuachi