Proyecto final FEDA 2025: Analisis de datos twitter en Digrafos

DIEGO NEIRA SALGADO*, Universidad de Concepción, Chile

En el presente informe se trabaja sobre datos de la red social X (Ex-twitter) y se revisan relaciones (de seguimiento) entre los usuarios.

Se busca encontrar la mejor forma de estudiar dichas relaciones y las mejores formas de trabajar con ellas para obtener conclusiones relevantes.

En particular, se busca estimar la ideología de los usuarios. Para esto, se utiliza la relación entre ellos, y en particular, con 4 medios de comunicación escrita con tendencias políticas conocidas. Todo esto haciendo uso de herramientas de estructuras de datos, de algoritmos computacionales y teoría de grafos.

Se utiliza conexidad fuerte y distancia entre nodos. Tomando como supuesto que la existencia de un camino entre un usuario y un medio ideologizado, tiene relación con la ideología del usuario.

Se encuentran las comunidades representadas por Componentes Fuertemente Conexas y se establece para cada usuario valores porcentuales de cuán influenciados están por cada ideología. Se concluye sobre la pertinencia de las estructuras y cálculos para los datos dados.

Additional Key Words and Phrases: Twitter, Users, Graphs, Kosaraju, FEDA

ACM Reference Format:
Diego Neira Salgado. 2025. Proyecto final FEDA 2025: Analisis de datos twitter en Digrafos. 1, 1 (July 2025), 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Este trabajo analiza una red social basada en datos de usuarios de Twitter, modelada como un dígrafo con más de 40.000 usuarios y 80.000 conexiones unidireccionales. El objetivo principal fue identificar comunidades (componentes fuertemente conexas, CFC), usuarios influyentes y estimar su tendencia ideológica en función de su proximidad a ciertos medios de comunicación representativos.

Para el tratamiento de datos, se utilizaron estructuras eficientes como unordered_map y Struct, optimizando la carga y búsqueda de información. Se aplicó el algoritmo de Kosaraju para detectar CFCs con una complejidad temporal O(V + E), y se implementaron funciones auxiliares para hallar los usuarios más influyentes dentro de cada comunidad. Dentro de los cuales un BFS que se debió usar para cada Vértice tuvo un gran impacto temporal O(V * E).

*

Author's Contact Information: Diego Neira Salgado, dneiras@udec.cl, Universidad de Concepción, Concepción, Bio-Bio, Chile.

ACM XXXX-XXXX/2025/7-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Datos y estructuras

2.1 Usuarios (Vértices)

archivo: twitter_users.csv
Contiene una linea de encabezado y luego lineas con 6 campos separados por ";", que corresponden a:

- User_ID
- User_Name
- Followers_Count
- Number_Tweets
- Friends_Count
- Created_At

2.1.1 Estructuras. Como estructura principal para los usuarios se decide utilizar un unordered map (unordered_map) usando como clave User_ID y un Struct para el resto de datos, Struct tiene sentido ya que tiene una cantidad fija de campos que almacenar y unordered_map ya que la mayoría de las operaciones a realizar será búsqueda sobre el ID.

```
#include <iostream>
unordered_map<long long, Perfil> usuarios;

struct Perfil {
    long long User_ID;
    string User_Name;
    int Friends_Count;
    int Followers_Count;
    long long CFC; // Inicialmente igual al User_ID
};
}
```

Table 1. Datos capturados de usuarios

Campo	Descripción
`User_ID`	Se buscó verificar si el tipo `int` soportaba el mayor valor de los datos. Se concluyó cambiarlo por `long long`.
`User_Name`	Necesario para mostrar en pantalla en lenguaje humano. Se utiliza el tipo `string`.
`Followers_Count`	Se mantuvo porque es útil para determinar los líderes de influencia. Se usa el tipo `int`.
`Friends_Count`	Se mantuvo porque permite identificar a los usuarios más influenciables. Se usa el tipo `int`.
`Number_Tweets`	Se eliminó para reducir el tamaño de las estructuras en memoria, sacrificando cierta flexibilidad del código. No se utilizó en el análisis, pero podría añadirse fácilmente si se amplía el estudio.
`Created_At`	Se descartó por la misma razón anterior.
`CFC`	Representa la componente fuertemente conexa. Es un `long long` y su valor inicial es igual al `User_ID`.

2.1.2 Tratamiento. Hay cuatro usuarios especiales para efectos del proyecto:

Cooperativa (7668952;Cooperativa) es reportada como un medio de izquierda
El Mostrador (12815132;elmostrador) es reportado como un medio libertario
Soy Valdivia (239534277;soyvaldiviacl) es reportado como un medio de derecha
La Tercera (3222731;latercera) es reportado como un medio de centro

los cuales se busca la distancia para determinar la ideología

2.2 Conexiones (Arcos)

archivo:twitter_connections.csv

Contiene una línea de encabezado y luego lineas con 2 datos separados por ;, que corresponden a:

- nodo de entrada
- nodo de salida para cada conexión (de seguimiento unidireccional) entre usuarios

2.2.1 Estructuras. Dado que la operación que más se utilizará será la búsqueda, se elige una lista de adyacencia almacenada en un unordered_map

```
unordered_map<long long, vector<long long>> adyacencia;
```

2.2.2 Tratamiento. Debido a la decisión de utilizar los User_ID como clave en el unordered_map es que se debe realizar una conversion de las aristas de User_Name->User_Name a User_ID->User_ID para poder relacionarla la estructura Perfiles (más detalles en las discusiones)

2.3 Componentes fuertemente conexas CFC

Se utilizará un vector de Struct CFC para almacenar los datos de las estas. Con un ID Correspondiente al User_ID del miembro más popular.

El struct incluye una subestructura miembros, para almacenar los usuarios que la componen, 4 floats que guardarán el porcentaje de cada una de la tendencias políticas que se le asignará a cada componente, finalmente top_influyentes para almacenar los (hasta 5) miembros más influyentes de la componente.

En los miembros y en top_influyentes, solo se buscará presencia por lo que unordered_set es suficiente (primero se utilizaría un vector pero se determinó que las búsquedas son más eficientes en esta estrcutura).

```
#include <iostream>
struct CFC {
    long long id;
    unordered_set<long long> miembros;
    float porcentaje_izquierda = 0.0;
    float porcentaje_libertario = 0.0;
    float porcentaje_derecha = 0.0;
    float porcentaje_centro = 0.0;
    unordered_set<long long> top_influyentes;
};
```

3 Implementación

3.1 Captura de datos y carga en estructura de datos

3.1.1 Captura de datos desde archivo usuarios. Se leyó el archivo twitter_users.csv y se organizaron los datos en el struct definido en la sección anterior. Ignorando por el momento los datos que no se utilizarán.

Adicional: se realizará un mini experimento para calcular de número de caracteres promedio de los User_names para evaluar el impacto de la modificación del índice y evaluar el cambio de las aristas de strings a números (que finalmente serían long long)

3.1.2 Captura desde archivo conexiones y modificación. Se capturaron los datos desde twitter_connections.csv, dado que ya se tenían cargados los datos de los usuarios se buscó cada uno de los nombres de usuarios y se reemplazó el valor de v.User_name y de u.User_name para cada arista por v.User_ID y u.User_ID respectivamente. Dado que los usuarios ya se encontraban en una estructura eficiente, unordered_map que entrega un valor esperado constante para búsqueda, la complejidad temporal de este proceso es de O(E).

3.2 Busqueda de personas más influyentes y más influenciables (del dataset)

Se realizó (por requisito del enunciado) un recorrido por todos los usuarios rescatando los 10 usuarios con mayor numero de followers y los 10 con mayor numero de friends, dato que se encuentra directamente almacenado en el Struct. Esto tiene una complejidad temporal de O(V)

3.3 Busqueda de componentes fuertemente conexas:

Para realizar la búsqueda de componentes fuertemente conexas se utilizó el algoritmo de Kosaraju por su buen desempeño, ya que recorre solo una vez (asintóticamente) cada arista, consiste en:

- Hacer un recorrido DFS y acumular en un stack desde el último al primero
- Revertir la dirección de las aristas del grafo
- Recorrer por DFS los nodos no visitados y todos los que son alcanzables son parte de una componente y los que no, son su propia componente

Fuente: https://www.youtube.com/@basicsstrong

3.4 Busqueda de personas más influyentes (de cada componente)

- Se seleccionó, con un ciclo for, las 5 personas más influyentes de cada componente, entendiendo por infuyente, que tenga mayor número de seguidores. Dato capturado desde la base de datos y almacenada en el Struct de cada usuario. En caso de que el componente esté compuesto por menos de 5 miembros, todos ellos son se consideraron los más influyentes)
- La persona más influyente determinó el ID de la componente

3.5 Determinación de ideología directa

- Se definió una función bfs_distancias que con uso de DFS nos permite determinar la distancia de un nodo u a un nodo v
- Se definió la función para ideología calcular la ideología directa de un nodo con la formula:
  - (hay que incluir que v-v hace que el nodo sea 100 de su propia ideología)
  - Si no existe camino a alguno de las ideologías la el % es 0 y se ignora para los siguientes cálculos
  - Calculamos con la función bfs_distancias, la distancia del nodo a cada uno de los (a lo mas) 4 medios y los almacenamos localmente para calcular: % de ideología directa I = 1 - (Distancia a medio con ideología I/Suma Distancias)

3.6 Definición de la ideología de cada componente

Se establecieron los porcentajes de ideología de la componente como el promedio de las ideologías de cada uno de los miembros influyentes utilizando los resultados del paso anterior

3.7 Determinación de ideología contextual para un usuario en concreto

-
- Se definió el porcentaje de la ideología social para los miembros de un CFC como un rango de porcentajes para cada una de las cuatro ideologías cuyos limites son:
  - Limite inferior: min(ideología directa, ideología de su CFC)
  - Limite superior: max(ideología directa, ideología de su CFC)
- Para los usuarios cuya CFC son ellos mismos, se estableció su ideología directa como ideología contextual

3.8 Estructura del programa

Se utilizará para los cálculos, 4 archivos cpp y sus 3 respectivos headers:

- captura.cpp: archivo con las funciones para captura de datos, su archivo de encabezados contiene el struct de usuarios
- cálculos.cpp: código para las funciones que calculan las CFC y las ideologías, su .h incluye el struct de los CFC
- salidas.cpp: archivo para escribir las funciones que entregarán las salidas de forma organizada de la información
- main.cpp: archivo para llamar todas las funciones y ejecutar el programa

Para la compilación de los archivos de calculos se utilizará

- Makefile

Para representar datos se utilizarán archivos que apliquen formato para lectura humana:

- ej_CFCs.cpp
- ej_usuarios.cpp
- ej_relevantes.cpp

4 Experimentación y discusión

4.1 Datos de entrada

Lo primero que se notó al observar los archivos con los datos, fue el tamaño relativo de las arcos con respecto al número de nodos. Por lo que si se quitaran estos 4 nodos el numero de conexiones sería de 79.272 lo que da

Table 2. Datos de entrada

Descripción	**Valor**
Número de usuarios (V)	41.858
Número total de arcos	83.072
Grado de entrada de *Cooperativa*	1.000
Grado de entrada de *Soy Valdivia*	1.000
Grado de entrada de *La Tercera*	700
Grado de entrada de *El Mostrador*	1.000

un promedio aproximado de 1,8964 conexiones por nodo. Además, por simple observacion hay usuarios que tienen muchos seguidores, por lo que es altamente probable que existan multiples puentes y por lo tanto varias componentes altamente conexas. Lo que implicaría que muchos usuarios no formaran parte de CFCs, esto no se puede asegurar de los datos pero sí nos entrega una intuición.
De los datos se observa que, evaluando con los criterios que tenemos sobre grafos en general, podemos decir que, si relajamos la direccionalidad de las conexiones: es decir si se evalúa por el número de aristas, el grafo fundamental está mucho más cerca de un arbol (A = V - 1) que de un grafo completo (A = V$\hat{2}$). Lo que refuerza la intuición anterior.
Lo cierto es que esto no es definitorio, ya que con A >= V existe un completamente conexo (con una formación circular).
También se puede notar que el mostrador es el único medio que no sigue a nadie en la muestra y la tercera sigue a muchos usuarios, los otros dos siguen al menos un usuario.

4.2 Elección de estructuras:

4.2.1 Usuarios (nodos V). Antes de definir las estructuras y los algoritmos ya se podía intuir que las operaciones que más se repetirían serian las de búsqueda, es por esto, que determinamos que la estructura más eficiente para almacenar los nodos o usuarios serían unordered_map.

La siguiente discusión fue la elección de la clave para para el unordered_map. Fue necesario discriminar si convenía utilizar el User_ID o User_Name como clave.

Usar User_ID hacía necesario cambiar cada dato (O(E)) y para cada una realizar una búsqueda en el U-Map (O(V)) y reemplazarla (O(1)) y la ventaja se genera por la comparación de strings de un tamaño de 1 byte por character del nombre con una cantidad promedio por determinar v/s 8 bytes por ser de typo long long (se trabajó con arquitectura de 64 bits). Una vez escrito y ejecutado el código se encontró que el tamaño promedio de los strings era de 11.58, lo que genera una ventaja aunque no fue tan grande como se esperaba, sobre todo porque cuando se plantó inicialmente se esperaba usar int pero este tipo no soportaba los datos.

Para el cálculo de mayor costo temporal: la determinación de distancias con BFS para cada nodo (O(V * E)), la diferencia podría ser menor, ya que el costo de reemplazar (O(E)) + el costo del algoritmo suman, asintóticamente, lo mismo que no reemplazar. Pero si se espera una diferencia a en las constantes que acompañan las operaciones, y debido a que el reemplazo se realiza una sola vez, y se realizan varias operaciones de busqueda (no solo el BFS) estimamos que esta diferencia sí tendrá efecto.

Para el resto de los datos, la primera decisión que se tomó fue, arbitrariamente, ignorar los campos de twitts y fecha de creación, ya que para todo el analisis de este trabajo no se utilizaron estos datos para facilitar la lectura y escritura del código.

Luego, se analizó si era mejor trabajar con una tupla conociendo el orden de los campos, se determinó que un struct hacía más fácil de leer y/o corregir el código, y además, se pueden incorporar más adelante los campos ignorados en caso de que queramos ampliar el estudio. Modificando solo el struct, la función de captura y la función en la que queramos incorporar o ampliar.

Diferencia de tupla v/s Struct: Para diferencias en tamaño hicimos el siguiente mini experimento:

```
struct Perfil {
    long long User_ID;
    std::string User_Name;
    // int num_tweets;
    int Friends;
    int Followers;
    // std::string Created_At;
    long long CFC;
};

int main() {
    std::tuple<int, std::string, /*int*/, int, int, /*std::string*/> arr_t[1000];
    Perfil arr_p[1000];

   std::cout << "Tamaño total del array de 1000 tuplas: " << sizeof(arr_t) << " bytes" << std::endl;
   std::cout << "Tamaño total del array de 1000 perfiles: " << sizeof(arr_p) << " bytes" << std::endl;
}
```

Lo que mostró que no existía diferencias en tamaño
Y estas fuentes:

- https://en.cppreference.com/w/cpp/utility/tuple.html
- https://eel.is/c++draft/class.mem
- Prueba: el assembly generado (-O2 o -O3) es el mismo. Puedes verlo en Compiler Explorer.
- https://quick-bench.com/

El resultado fué que no había diferencia significativa, pero los Struct eran más fáciles de trabajar

4.3 Cálculos

4.3.1 Cálculo de CFC:. Como se mencionó antes, se decidió utilizar el algoritmo de Kosaraju por su buen desempeño para digrafos grandes ya que asintóticamente tiene una complejidad temporal de O(V+E). Esto se debe a que recorre un multiplo fijo (independiente del tamaño de los datos) de veces cada arista y cada vertice. Para esto fue conveniente tener almacenadas las aristas en una lista de adyacencia.

Una vez que se encontraron las componentes fuertemente conexas, se decidió que se las utilizaría para calcular la tendencia de los usuarios. Segun lo conversado en clases las CFC suelen ser interpretadas como comunidades humanas.

Una vez determinado como se calcularían e interpretarían las CFC, se pasó a definir cómo se usaría la información que el grafo entrega para determinar las tendencias políticas de los diferentes usuarios. El primer gran supuesto que se hizo fue que las personas que son más populares en la comunidad, suelen representar de buena manera la forma en que piensan los miembros de la comunidad.

Dado que una sola persona (la más popular) podría generar algun sesgo por algún tema incidental, se buscó disminuir anomalías estadísticas tomando una muestra de 5.

De forma análoga a la estructura Union-Find, se utiliza un líder para identificar las diferentes CFC, pero en este caso particular se utiliza el miembro con mayor influencia (número de seguidores) como líder.

Análisis asintótico: Como mencionamos anteriormente escogimos el algoritmo de Kosaraju ya que garantiza una complejidad temporal del algoritmo que es O(V + E). Y con esto entrega todos los CFC, es decir no se necesita repetir para encontrar cada uno de ellos, lo que es eficiente en este caso que notamos que habría un gran numero de ellos (si consideramos los de tamaño 1), ambas cosas que se verificaron experimentalmente con la ejecución del código.

4.4 Cálculo de tendencia política:

La primera opción que discutimos es la posibilidad de que las particularidades de cada persona, que puede ser si participan mucho o poco en la red social, si siguen o no personas que piensan diferente o suelen seguir solo a personas con la misma ideología, son detalles que son inconsistencias a las cuales se les puede disminuir utilizando grupos en lugar de individuos. Por lo que nuestra primera intuición fue utilizar la comunidad a la que pertenecen, es decir, su componente fuertemente conexa para determinar la corriente política del usuario.

Lo siguiente fue reconocer que todos los usuarios son particulares y distintos por lo que debíamos incorporar las decisiones que tomó cada usuario en su participación el twitter como a qué personas y/o medios de comunicación sigue. En este sentido decidimos incorporar una característica propia de un grafo dirigido que nos entrega mucha información de la relación o cercanía de un nodo con otro, que es la distancia en el gráfico, por lo que decidimos determinar una relacion inversamente proporcional pero relativa (a la todas las distancias). También nos encontramos con el caso particular en que toda la CFC podrían no tener camino a alguno de los medios, para esto incorporamos el caso en que no exista camino definiendo la tendencia como 0, y como el camino sería infinito se deja de considerar para el resto de cálculos: matemáticamente esto haría que la tendencía de cada uno de los otros fuera 100% lo que no hace sentido y además elimina la información de como la proporción de las

diferentes tendencias políticas.

Finalmente tenemos el caso de que cada medio no tiene necesariamente un camino a si mismo, pero en teoría el resultado (lo comprobamos experimentalmente es infinito, por lo que, analizamos 3 opciones:

(1) Crear el arco v-v forzando a que haya un camino siempre a si mismo de distancia 1
(2) Modificar el código para el caso específico que de el valor infinito y asigne el 0
(3) Finalmente la opcion que escogimos es calcular todos iguales y luego reemplazar los valores por los deseados para esos 4 usuarios

Elegimos incorporar ambos piezas de información en el cálculo de cada definitivo de las tendencias, la siguiente decisión fue determinar como agregaríamos cada dato, vimos dos opciones que nos parecieron razonables, la primera fue establecer un promedio simple entre la tendencia calculada por distancias de usuario y la de su CFC, lo que podría generar porcentajes que sumen más de 100%, y luego ponderar para llegar a la suma 100. La segunda opción y la que elegimos de forma arbitraria fue entregar, en lugar de un valor un rango de valores, determinados por los limites de las dos opciones que se promediaban en el cálculo anterior. Escogimos esto ya que nos pareció que: por un lado, incorpora la información del cálculo anterior y por otro lado el tamaño del rango nos entrega información adicional que nos parece relevante, rangos más amplios informan que tan distantes estan las ideologías del usuario de la de su comunidad (CFC)

Análisis asintótico: El cálculo de las distancias a cada uno de los nodos de los medios de comunicación es la parte de todo el proyecto que tiene una mayor complejidad temporal, el hecho de que se deba recorrer con BFS dos veces tiene una complejidad temporal de O(E) pero el se debe realizar para cada uno de los usuarios, por lo el algoritmo en su conjunto para obtener los datos para todos los usuarios de O(V * E) por lo que es el algoritmo u operación más costosa que pudimos encontrar. Lo que se ve claramente reflejado en el resultado experimental de los tiempos obtenidos.

La modificación de los datos de los medios de comunicación tiene tiempo constante (suponiendo que el numero de medios es fijo) e insignificante en comparación al resto del código.

Finalmente, el cálculo de la tendencia contextual tiene una complejidad para cáda módulo constante, ya que en el struct de los usuarios está almacenado el CFC al que pertenece, donde debemos hacer consultas en unordered_map por lo que la complejidad temporal total de esta operación en su conjunto (para todos los usuarios) es de O(V)

Problema encontrado para usuarios influyentes: Durante el desarrollo del código, nos encontramos con una situación que no habíamos previsto de antemano: cuando intentamos imprimir los 10 usuarios más influyentes nos dimos cuenta de que el uso de unordered map en lugar de un vector para almacenar los usuarios, hacía necesario copiar todos los datos en otra estructura que permita ordenar, por ejemplo un vector y luego realizar un ordenamiento cada vez que quisieramos encontrar un top de usuarios, lo que eliminaba la ventaja de elegir esa un unordered_map

En lugar de esto, encontramos una solucion diferente, hicimos un cambio en el que creamos una función que nos permite conocer el los top k usuarios, generalizada en k para poser usarla en el digrafo completo pero también en cada CFC para encontrar el líderes y el conjunto de líderes internos que necesitabamos para el resto de cálculos.

Esto genera una mejora en terminos temporales de O(E log E) a O(E log k) y como k es constante esto es O(E) que es recorrer solo una vez todos los usuarios (E) para encontrar el top K.

5 Resultados

5.1 Ambiente de trabajo

Se realizaron los experimentos en un computador con windows 11 corriendo WSL2 con un procesadro ARM 64 bits y con las bases de datos entregadas

Table 3. Tiempos de lectura de datos

Proceso	**Tiempo (ms)**
Carga del archivo de usuarios	70
Lectura de conexiones	52
Conversión de `User_Name` a `User_ID`	45
Cálculo total de las CFCs	190

Table 4. Resultados de los datos

Métrica	**Valor**
Total de CFCs encontradas	39.230
CFCs de tamaño 1	39.214
CFCs de tamaño > 1	16

Table 5. Tiempos de procesamiento de los datos

Proceso	**Tiempo (ms)**
Cálculo de ideologías directas	78.999
Cálculo de tendencias ideológicas de las CFCs	22
Cálculo de ideología contextual	57
Generación de `ideología_contextual.csv`	102
Generación de `ideología_directa.csv`	75
Generación de `CFCs.csv`	89

5.2 Ubicación de los resultados

- Las componentes fuertemente conexas están reportadas en CFCs.cvs para lectura computacional y en CFCs_legibles.txt para lectura humana
- Los usuarios relevantes están reportados en usuarios_relevantes.csv
- Los usuarios con sus ideolgías calculadas estan en ideología_contextual.csv para lectura computacional y en usuarios_legibles.txt para lectura humana

6 Conclusiones

6.1 Resultados de los experimentos

Con respecto a los resultados de la ejecución del código, pudimos comprobar la complejidad temporal de lectura de nuestro unordered_map de usuarios, esto se ve por que el tiempo de convertir es muy cercano al tiempo de carga de las conexiones (O(E)).

También pudimos comprobar la eficiencia temporal de Kosaraju (O(V + E)) que tuvo un tiempo algo mayor que la suma de capturar cada uno (O(E) + O(V))

Para el cálculo de distancias para establecer las ideologías notamos claramente como afecta un solo cálculo al tiempo de ejecución de todo el proyecto por tratarse de O(V * E)

Nos llamó poderosamente la atención la enorme cantidad, en primer lugar de usuarios que no fueron asignados a ninguna CFC y, en segundo lugar, el numero de usuarios sin tendencia política asignada.

6.2 Conclusiones finales

Una vez observados los resultados podemos concluir:

6.2.1 Con respecto a los resultados de ideología política: Por un lado, nos parece que el análisis de lo que podría significar la relación entre datos de entrada y los obtenidos puede seguir teniendo sentido con respecto a la ideología política de los usuarios. Para poder mejorar los resultados, en el sentido de entregar más información, el dataset que nos podría permitir encontrar estas relaciones con los supuestos y cálculos utilizados debería ser mucho más centrados en las conexiones o arcos en relación a la cantidad de usuarios.

Formalizando, deberíamos elegir un dataset con una cantidad de arcos lo suficientemente grande en relación a la cantidad de vértices que nos entregue una probabilidad alta de que para particiones medianas del grafo, los d-(D')>0 para todo D' de modo de garantizar componentes fuertemente conexas. La determinación de ese tamaño de subgrafo, de la probabilidad y por tanto del cálculo de esa proporción, escapan al alcance de este trabajo y del curso. Pero de forma arbitraría, me gustaría intentar, por ejemplo, con un grafo de 10.000 usuarios y 300.000 a ver como varía la cantidad de CFCs y de resultados de ideología distintas de 0.

6.2.2 Con respecto a los resultados de las estructuras, algoritmos y código. Creemos que las respuestas a los cuestionamientos, como la elección de Structs sobre tuplas, el uso de unordered_map y unordered_set en varias partes del código cuando las operaciones consistirían principalmente en buscar, fueron acertadas. Su impacto fue siempre positivo y nos entregan lineamientos y enseñanzas sobre cuando es conveniente usar estas estructuras, pero creemos que su impacto total global fue limitado.

La elección de Kosaraju para el CFC también fue acertado ya que dentro del desempeño global del código el tiempo de ejecución fue razónable. Y nos entregó sin problemás una de las respuestas solicitadas por el enunciado.

Para la determinación de los elementos más influyentes y más influenciables, nos hizo generar un aprendizaje importante, como unordered_map era recomendable para las búsquedas por Indice. Como la estructura no nos permitía ordenar (O(V log V)) creamos la funcion encontrar los K usuarios mayores, de tiempo lineal (O(v)), mejor que la idea que teníamos de ordenar y recuperar los primeros valores.

Finalmente y más importante de acuerdo a los tiempos totales, nos parece que el encontrar los caminos más cortos a los diferentes medios de comunicación fue tenía un costo temporal muy alto, y durante la redacción de estas últimás lineas encontramos una manera que podría ser más eficiente pero no tuvimos tiempo de probarla. Que sería generar un grafo compacto, en el que reemplazamos los arcos a cualquier miembro de la CFC por arcos directos al líder y así reducir el tamaño del grafo y mejorar el resultado. Aunque de los datos de CFS sabemos que el numero de vertices se reduciría solo a 39.230 en este dataset particular.

Received 14 July 2007; revised 14 July 2007; accepted 14 July 2007