TALLER N°2

Título

Teoría de Grafos

usach

Taller de programación 1-2024

Fecha:27/05/2024

Autor: Benjamín Moya

TALLER N°2

Teoría de Grafos

Explicación breve del algoritmo

Dentro del mundo de las comunicaciones, las conexiones entre personas, empresas o distintos usuarios de una plataforma son detalladas mediante el uso de grafos de distintos tipos, los cuales nos pueden dar una idea de cómo estos pueden tener conexiones en común, ya sea en gustos o entre usuarios. Por lo mismo, el encontrar estas redes entre grafos de manera veloz es primordial para el funcionamiento de calidad dentro de la plataforma.

Los cliques son un grupo de nodos dentro de un grafo que están conectados todos entre sí, por lo que puede haber muchos cliques en un mismo grafo, pero el más representativo y el que nos compete siempre será el máximo de estos.

De esta manera el programa desarrollado se enfoca en la búsqueda eficiente y veloz del máximo cliqué dentro de un grafo no dirigido, mediante el uso del algoritmo de Bron-Kerbosch, el cual se enfoca en la enumeración de los vértices conectados por una arista o vecinos y los que no, pero no así en la eficiencia del mismo, dando lugar a que el principal objetivo será la optimización del algoritmo y del entorno de ejecución con tal de proveer ejecuciones cortas en un número alto de elementos de un grafo.

Heurísticas o técnicas utilizadas

- Algoritmo principal: Como pilar de nuestro programa tenemos al algoritmo de Bron-Kerbosh el cual por su naturaleza se clasifica como Backtracking, pero dentro de la implementación a este se le ha añadido un conjunto extra (conjunto X) el cual excluye a aquellos vértices que provoquen cliques repetidas dentro de la recursión, por lo que recorta las ramificaciones del árbol y podemos pasar a considerarlo como Poda del Backtracking.

- Heurísticas:
    1. Pivote: Esta implementación dentro del algoritmo se realiza antes de entrar a la iteración sobre el conjunto de los vértices vecinos, ya que, como queremos encontrar el clique máximo dentro de un grafo, intuitivamente podemos deducir que el vértice con mayor grado del clique, o el que tiene una mayor vecindad, será parte de este mismo, gracias a que comparte conexiones con la mayor parte de los elementos del grafo, donde si el clique máximo es al menos de 2 elementos, el será parte de este subconjunto. Por esta misma razón lo seleccionamos, y aplicamos una diferencia entre el conjunto de vértices de entrada al algoritmo o aquellos a iterar, con los vecinos de este vértice de grado máximo obteniendo un nuevo conjunto de vértices a iterar los cuales son aquellos que no son vecinos del pivote, ya que estos van a ser considerados dentro de la recursión del mismo pivote, recortando ramificación y optimizando el algoritmo.

Por otro lado, se ha añadido una segregación de los posibles pivotes, ya que en un grafo con densidad superior a 9.5, la gran parte de los vecinos del pivote estarán en la diferencia, generando un conjunto vacío donde iterar, dejando así inutilizable el algoritmo. Por ende, cuando esto suceda simplemente se elige el primer vértice del conjunto, luego de la diferencia con el de grado máximo, para no perder esta característica.
Finalmente podemos clasificar esta heurística como Greddy gracias a que buscamos el óptimo de cada iteración con tal de llegar a un óptimo general.

2. Clique máximo: Para esta consideración, dentro del código, añadimos un atributo dentro de la clase Graph, el cual guarda el clique máximo encontrado hasta el momento luego de revisar todo vértice dentro del conjunto iterador. Este mismo nos ayuda a saber si vamos por buen camino en cada ramificación, debido a que si el tamaño del clique que se está formando (conjunto R) más los vértices que quedan por revisar no superan o igualan al tamaño del clique máximo encontrado entonces no se hace la recursión, ya que potencialmente no se encontrara en esa rama un clique mayor que él que se ha guardado en el atributo, acortando la búsqueda, optimizando el algoritmo y clasificándolo como Dynamic Programming gracias a que guardamos una solución con tal de utilizarla cuando se necesite acortando procesos repetitivos que pueden llevar a un extenso tiempo de ejecución.

Funcionamiento del programa

Revisar el anexo 1 para el diagrama representativo de la secuencia.

1. Lectura del grafo: Luego de haber compilado, cuando se ejecute el programa, por consola se solicitará el nombre del archivo que contenga el número de vértices y los pares de aristas para armar la matriz de adyacencia que represente al grafo, el cual se debe considerar que es uno no dirigido por lo que los vértices se conectan bidireccionalmente en la tabla una vez se indique su conexión.
2. Inicialización de conjuntos: Luego de definido el grafo se crean los conjuntos necesarios para inicializar la primera instancia del algoritmo de resolución el cual necesita de 2 conjuntos vacíos, uno para la formación del clique y otro para dejar a los vértices excluidos, 1 lleno de números desde el 0 hasta el tamaño del grafo menos 1 y un conjunto de conjuntos donde se guardarán los cliques que se han encontrado en cada ramificación exitosa.
3. Inicialización del cronómetro: Usado para la medición temporal del algoritmo con tal de comparar estadísticas y resultados.
4. Ejecución del algoritmo:
    a) Verificación de conjuntos y clique máximo: Se comprueba que tanto el conjunto de los vértices a revisar como el de excluidos este vacío, si esto sucede es que no hay más vértices que iterar, se agrega el clique formado (Conjunto R) al conjunto de cliques encontrados. Por otro lado, antes del retorno se comprueba si el clique formado es mayor que el clique máximo actual, si esto sucede, se reemplaza el clique máximo guardado por el recientemente formado y se retorna. Si la primera condición no se cumple, es que quedan vértices por revisar y se omite esta parte.

b) Elección del pivote: Para este paso se llama a la función maxDegree(), la cual compara los grados de cada vértice del conjunto de entrada y retorna el que posea el mayor de todos los elementos. Luego se llama a la función vertexNeighbours(), la cual crea un conjunto de los vértices que son vecinos al vértice de entrada. Todo lo anterior con tal de realizar la diferencia entre los vértices de entrada de la función (Conjunto P) y los vecinos del pivote. Finalmente si el conjunto creado es vacío se realiza la misma operación pero con el primer elemento del conjunto de vértices a revisar e iterando sobre este mismo.

c) iteración: Se realiza este paso para cada vértice dentro del conjunto diferencia creado a raíz del pivote. Así mismo, se crean los nuevos conjuntos de entradas como R al que se le añade el vértice elegido en la iteración, P el cual surge de la intersección entre los vértices de entrada al algoritmo y los vecinos del vértice de la iteración y X que realiza la misma operación que P. Luego de definidos los conjuntos se verifican si el nuevo tamaño del clique más la nueva cantidad de vértices a revisar superan o igualan al tamaño del clique máximo, de ser así se realiza la recursión con los nuevos conjuntos generados, sino se salta este paso, se borra el vértice de la iteración del nuevo conjunto de vértices a revisar y se inserta a los excluidos para no tener que revisarlo una vez más.

d) Fin del algoritmo: El algoritmo termina en cuanto se retorne el conjunto de los cliques encontrados ya sea por no tener más vértices que revisar, ni excluidos o solo por la primera condición al terminar la ejecución de la iteración “for”.

5. Término de la medición temporal: Con tal de aislar el tiempo de ejecución del algoritmo
6. Salida: Se termina el programa mediante la selección del máximo clique dentro del conjunto de cliques encontrados como el clique con mayor número de elementos y se presenta el resultado por consola, así como su tamaño y tiempo de ejecución del algoritmo.

- Extras: Funciones auxiliares para el algoritmo principal.
  a) degree(int v): Mediante el vértice de entrada se realiza una iteración dependiendo del tamaño de la matriz de adyacencia, fijando en la declaración de esta la fila correspondiente al vértice y avanzado en columnas verificando si existe una conexión entre cada vértice con el de entrada, si lo hay se agrega al contador. Al finalizar el ciclo, se devuelve este contador.

  b) maxDegree(set<int> P): Mediante el conjunto de vértices de entrada y un ciclo, se verifica si el grado del vértice actual es el mayor hasta el momento comparándolo con el grado del vértice guardado como máximo. Se finaliza retornando el vértice con el mayor grado en el conjunto de entrada.

  c) vertexNeighbours(int v): Se inicializa un conjunto que guarde todos los vecinos del vértice de entrada. Luego se realiza un ciclo del tamaño de la matriz de adyacencia, donde se fija el vértice en la declaración de la matriz en su posición de fila y se itera sobre las columnas, donde si en la combinación actual se encuentra un 1 es que hay conexión y son vecinos, agregando el vértice revisado al conjunto de vecinos del vértice de entrada. Se finaliza retornando el conjunto armado a partir del ciclo concluido.

Aspectos de implementación y eficiencia

Para este apartado debemos tener en consideración los componentes esenciales dentro del entorno de ejecución del programa, como lo es un procesador Intel I3 10100 con cache de 6 MB y una frecuencia básica de 3.6 GHz, una SSD donde se aloja el programa con un bus de 6.0 Gb/s, 16 Gb de RAM a 2666 MHz y todo lo anterior montado en una placa madre Gigabyte H410M-H

Por otro lado, en temas de ejecución del programa, se pueden destacar 3 aspectos muy importantes dentro de la consideración de eficiencia de la implementación desarrollada:

1. Gestión de memoria estática y dinámica: Para el desarrollo de la implementación se desecharon modelos donde se utilizaban demasiados punteros que generaban una saturación en la memoria dinámica de los procesos, los punteros pueden generar una mejora en la gestión de la información, pero su excesivo uso puede ser costoso si no se limita su implementación, dando errores en tiempo de ejecución como std::bad_alloc, que es una excepción de C++, indicándonos que la memoria donde se alojan las estructuras dinámicas tiene fallo o está llena, dando lugar a que el programa termine súbitamente. Por lo mismo se utilizó memoria estática para los objetos o conjuntos que potencialmente podían referenciarse o inicializarse en muchas más ocasiones de las que puede aguantar la memoria dinámica incluso con cantidades grandes de RAM y cache dentro del procesador.
2. Proceso de compilación: Para la compilación se ha utilizado la segmentación del programa mediante un archivo makefile, el cual otorga una gran mejora en mantenibilidad y eficiencia dentro del tiempo de ejecución e incluso la escalabilidad del programa, ya que los archivos que no se hayan modificado y tiendan a ser invocados por la línea de comando simplemente se verán omitidos por el compilador y la gestión de recursos dentro del mismo no exigirá una revalidación de los componentes de este. Por otro lado, se usó la bandera -O2 la cual otorga una mejor gestión interna de los elementos del programa ofreciendo mejoras notables dentro del programa.
3. Disminución de escenarios: Gracias a la gestión del algoritmo detallado en aparatados anteriores y en compañía del primer punto revisado en la sección, las distintas instancias del programa se han visto reducidas considerablemente tanto en ramificaciones como en uso de memoria, por lo que si bien el algoritmo original dota de usabilidad dentro de la búsqueda del máximo clique, para casos donde el grafo supera una densidad de 0.3 en aristas, se torna muy lento e incluso inservible para cierta cantidad de vértices, donde en comparación, el algoritmo trabajado, deja un tiempo 24 milisegundos para instancias con densidad 0.816327,234 milisegundos para instancias con densidad 0.930612 y para instancias con densidad de 0.95 o más, el promedio fue de 400 a 500 milisegundos.

Ejecución del código

Para la ejecución del código son indispensables las librerías iostream, fstream, ctime, algorithm, iterator, vector, set y bits/stdc++.h.

1. Gestión de archivos: Se debe asegurar que los archivos Graph.h, Graph.cpp, main.cpp, makefile y el texto que describa el grafo en versión .txt (Primera línea: cantidad de vértices. Segunda línea hacia abajo: conexiones uno a uno, ejemplo: 0 1) este en el mismo directorio.
2. Compilación: Mediante una terminal alojada en el directorio correspondiente a la locación de los archivos (o mediante cualquier terminal y el comando cd “dirección del directorio”), se

debe ejecutar el comando: make all, para que el ejecutable principal del programa se produzca.

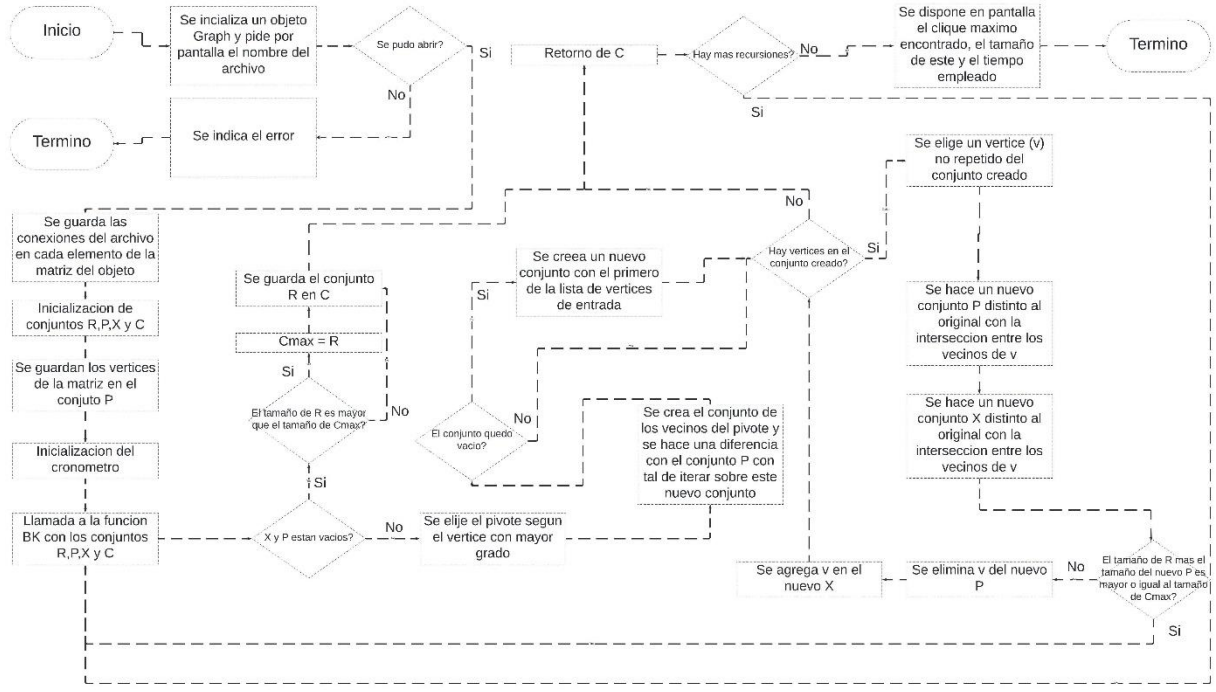
3. Ejecución: En la misma terminal de compilación, se debe escribir el comando ./main para iniciar la ejecución del programa el cual en primera instancia pedirá el nombre del archivo .txt donde se aloje la información del grafo, el cual debe ser escrita de la misma manera está en el directorio al igual que con su respectiva extensión (.txt).
4. Resultados: Una vez realizada esta secuencia se le indicara pantalla el clique máximo encontrado, así como el tiempo utilizado en la búsqueda de este. Por otro lado, si quiere ingresar otro clique deberá volver al paso 3.

Bibliografía

1) Östergård, P. R. (2002, August 1). *A fast algorithm for the maximum clique problem*. Discrete Applied Mathematics. https://doi.org/10.1016/s0166-218x(01)00290-6

2) Suyudi, M., Mamat, M., & Talib, A. (2018, March 6). *Branch and Bound Algorithm for Finding the Maximum Clique Problem*. https://ieomsociety.org. Retrieved May 15, 2024, from https://ieomsociety.org/ieom2018/papers/648.pdf

Anexos



Anexo 1: Diagrama de flujo/Programa de la búsqueda del máximo clique