TALLER N°2

Mayor Clique Máximo

usach

Taller de programación 1-2024

Fecha: 31 de Mayo

Autor: Sebastián Cassone

TALLER N°2

Mayor Clique Máximo

Explicación breve del algoritmo

El algoritmo que se plantea para resolver el problema del Clique Máximo de un grafo es el algoritmo Bron-Kerbosh, que se realizó con heurísticas que se detallarán más adelante. Lo que significa que lo que busca el algoritmo es buscar el subgrafo que se conectan correctamente a un conjunto completo de vértices, es decir, un clique, de tamaño máximo en el grafo dado. Este algoritmo es ampliamente utilizado en la teoría de grafos y es eficiente para encontrar cliques en grafos de tamaño moderado a grande.

Heurísticas o técnicas utilizadas

Las heurísticas que se utilizaron fueron las siguientes: Primero, se comienza seleccionando un pivote. Para esta selección, se eligió el vértice con la mayor cantidad de vecinos en común con los vértices en el conjunto P. Luego de esto, se hace una poda a las ramificaciones que podrían generar llamadas recursivas innecesarias. Esto se logra verificando si la suma de los tamaños de los conjuntos P y R en alguna llamada recursiva es menor o igual al tamaño del clique máximo que se tenga temporalmente. Si esta condición se cumple, la rama se poda, evitando así más exploración en esa dirección.

Adicionalmente, se paraleliza el proceso creando hebras para la ejecución del programa. Esto asegura que el programa se ejecute de la manera más rápida posible, ya que las hebras se ejecutarán en forma paralela, trabajando en diferentes partes del problema simultáneamente. Esto aprovecha la capacidad de procesamiento multicore para acelerar la búsqueda del clique máximo.

Funcionamiento del programa

El código comienza pidiendo al usuario que ingrese el nombre de un archivo que contiene la matriz de adyacencia de un grafo. Si la lectura del archivo falla (indicada por una matriz vacía), el programa notifica al usuario y termina la ejecución de esta sección.

A continuación, se reinician los conjuntos y estructuras de datos utilizados en el algoritmo: R, P, X, maxClique y C se limpian para asegurar que no contengan datos residuales de ejecuciones anteriores. Luego, se crea una instancia de la clase Clique con la matriz de adyacencia y su tamaño. El conjunto P se inicializa con todos los vértices del grafo, llenándolo con índices que van de 0 al tamaño de la matriz de adyacencia.

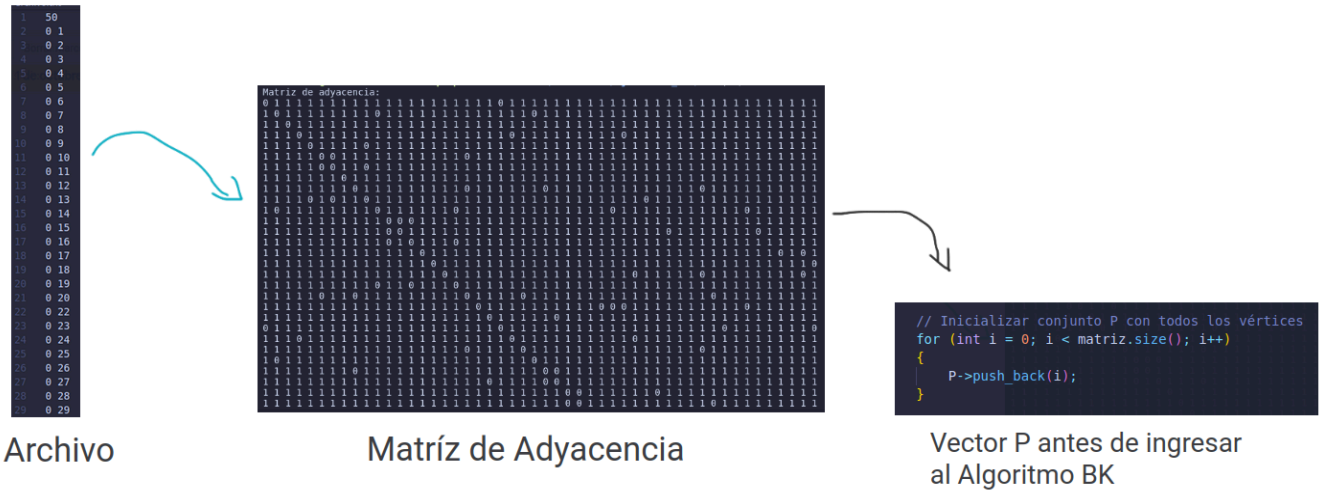

Figura 1: Proceso de Transformación del Grafo.

Para medir el tiempo de ejecución del algoritmo, se registra el tiempo inicial utilizando high_resolution_clock::now(). El algoritmo Bron-Kerbosch modificado se ejecuta en paralelo usando OpenMP. La directiva #pragma omp parallel asegura que las estructuras R, P, X, C y maxClique se compartan entre las hebras. Dentro de esta región paralela, se llama a la función BK de la clase Clique para iniciar el algoritmo desde el primer vértice.

El algoritmo Bron-Kerbosch modificado empieza con una comprobación de condición de parada: si los conjuntos P y X están vacíos, significa que R es un clique maximal. En este caso, R se añade al conjunto de cliques C, y si R es mayor que el maxClique actual, se actualiza maxClique. Después, el algoritmo selecciona un pivote u utilizando una heurística que elige el vértice con el mayor número de vecinos comunes en P y X. Esta selección de pivote ayuda a reducir el espacio de búsqueda, enfocando la exploración en los vértices más prometedores.

Si el tamaño combinado de R y P no puede superar el tamaño del maxClique actual, la rama se poda porque no puede generar un clique mayor. Luego, se calculan los vecinos de u y se crea P_excl_neigh_u, que contiene los vértices de P excluyendo los vecinos de u. A continuación, el algoritmo paraleliza la iteración sobre los vértices en P_excl_neigh_u utilizando OpenMP, lo que permite explorar diferentes expansiones de R en paralelo. Para cada vértice v en P_excl_neigh_u, se crea un nuevo conjunto R1 añadiendo v a R, y se calculan P1 y X1 como las intersecciones de P y X con los vecinos de v, respectivamente. Se llama recursivamente a BK con estos nuevos conjuntos. Al finalizar cada iteración paralela, en una sección crítica, se actualizan los cliques locales y maxClique para evitar condiciones de carrera. Este enfoque optimiza la búsqueda exhaustiva de cliques maximales mediante poda inteligente y paralelización, asegurando una exploración eficiente del grafo.

Una vez finalizada la ejecución del algoritmo, se registra el tiempo final y se calcula la duración en milisegundos. Si no se encuentra ningún clique máximo (indicando que maxClique está vacío), se informa al usuario. Si se encuentra un clique máximo, se ordena y se muestra. Además, se imprime el tamaño del máximo clique encontrado y el tiempo total de ejecución. Finalmente, se añade una línea en blanco para mejorar la legibilidad de la salida y se termina el bloque de código.

Aspectos de implementación y eficiencia

El código realiza varias optimizaciones para mejorar su eficiencia. Por ejemplo, utiliza heurísticas para seleccionar un pivote u que maximiza el número de vecinos comunes entre u y los vértices en P, lo que ayuda a reducir la búsqueda y evitar ramificaciones innecesarias. Esta selección inteligente de pivote reduce el número de llamadas recursivas necesarias y, por lo tanto, mejora la eficiencia del algoritmo.

En cuanto a la gestión de memoria, el código utiliza punteros inteligentes y contenedores STL para manejar eficientemente las estructuras de datos dinámicas, minimizando las fugas de memoria y maximizando la eficiencia de la asignación y liberación de memoria.

La paralelización del algoritmo utilizando OpenMP también contribuye significativamente a su eficiencia. Al distribuir la exploración del grafo entre múltiples hilos de ejecución, el algoritmo puede aprovechar mejor los recursos de hardware disponibles, lo que resulta en una mayor velocidad de procesamiento.

En cuanto a los vecinos que se guardaron en la clase Clique, se precalculan y almacenan para cada vértice durante la inicialización del objeto. Esto permite un acceso rápido a los vecinos de cada vértice durante la ejecución del algoritmo, lo que mejora la eficiencia en la comprobación de vecindades y la selección de pivotes.

Con respecto a los tiempos de ejecución, uno de los grafos que se encuentran en Classroom del curso (Link en el Anexo), en el grafo más difícil se obtuvo una mediana poblacional menor que 360 milisegundos (Estudio realizado en el anexo en R). Además, durante el desarrollo del programa se pudo ver una mejora en el tiempo cuando se reemplazó los conjuntos P, X, R, y C por Vector en vez de utilizar Set como en un inicio, también al comienzo se calculaba constantemente los vecinos de los vértices, por lo que cuando se comenzó a calcular los vecinos cuando se crea un objeto con la clase Clique se obtuvo una reducción importante en el tiempo de ejecución habiendo diferencias significativas entre ellos (Estudio realizado en el anexo en R).

Ejecución del código

Para compilar y ejecutar el programa, se requiere tener instalado un compilador de C++ (como g++) y el utilitario Make.

Para compilar el programa, simplemente ejecute el comando `make` en su terminal desde el directorio raíz del repositorio. Esto generará tres ejecutables: `testClique`, `testMenu` y `main`.

Con respecto a los archivos, `testClique` y `testMenu` son ejecutables de los dos test unitarios de las dos clases del proyecto solo con la finalidad de comprobar que las clases y sus respectivos métodos funcionan. Por lo que se ejecutan sin necesidad de introducir parámetros por consola.

Finalmente, el archivo ejecutable principal del programa `main` en cualquier distribución Linux (también válido para los test unitarios) se realizaría con `./main` de esta forma se comenzaría a ejecutar el programa principal.

Mostrará en la consola un menú con diferentes opciones, “1. Para ejecutar el algoritmo” y “2. Para salir”. Si se pulsa “1. Para ejecutar el algoritmo” inmediatamente pedirá el nombre del archivo, una vez introducido y pulsado `Enter` comenzará a ejecutarse mostrando posteriormente el Clique Máximo junto con su tamaño y el tiempo de ejecución.

Bibliografía

https://www.geeksforgeeks.org/introduction-to-parallel-programming-with-openmp-in-cpp/

https://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf

https://drive.google.com/file/d/1jNaNKrykboJ7_5lsyjSXhiPFAb1m0TyN/view

Anexos

Anexo 1:

https://drive.google.com/file/d/1LmZ4jjrw8lELm4zUYJQTI8Fk4P2C0q5e/view

Anexo 2:

https://drive.google.com/file/d/1veJ7OSuZbx5kF4t1sOOlkWR3ohb5sk_A/view?usp=sharing

Anexo 3:

https://drive.google.com/file/d/1PPpfGQOd33sFP9Vn4Rz4xnsaa0wIPGmx/view?usp=sharing