Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

T E S I S

Aplicación gráfica para la generación del Clique máximo en grafos

Presenta:

GENARO OSORIO GUARNEROS

Para obtener el grado de: Ingeniero en Ciencias de la Computación
Director: Dr. Pedro Bello López

Puebla, Pue, noviembre, 2025

Agradecimientos

Agradecimiento a la Vicerrectoría de Investigación y Estudios de Posgrado de la Benemérita Universidad Autónoma de Puebla por el apoyo económico brindado para desarrollar este trabajo de tesis dentro del proyecto VIEP 2025: aplicando ciencia de datos para analizar cadenas genómicas.

Tabla de contenido

Resumen

Esta propuesta de tesis plantea el desarrollo de una aplicación gráfica (en web) que permite generar, visualizar y analizar grafos, con el objetivo específico de detectar el clique máximo, una estructura fundamental en la teoría de grafos.

Un clique es un subconjunto de vértices donde cada uno está conectado con todos los demás y el clique máximo representa el mayor de estos subconjuntos en un grafo dado. Este problema es de alta complejidad computacional, clasificado como NP-completo [1], lo que lo convierte en un reto tanto teórico como práctico.

El proyecto se fundamenta en la necesidad de contar con herramientas que no solo resuelvan el problema mediante algoritmos exactos o heurísticos [2], sino que también faciliten su comprensión a través de visualizaciones gráficas accesibles. En la actualidad, existen bibliotecas como NetworkX o soluciones como Cliquer, pero ninguna integra una interfaz intuitiva orientada a la docencia y la investigación aplicada.

El desarrollo de esta aplicación contempla la integración de módulos para la manipulación de grafos, implementación de algoritmos de detección de cliques y visualización de los resultados, con énfasis en la facilidad de uso. La metodología incluye etapas de investigación, diseño, desarrollo, validación y documentación. Como resultado, se cuenta con una herramienta educativa y científica de libre acceso que facilita el estudio de la teoría de grafos, con aplicaciones potenciales en redes sociales, biología computacional, análisis de datos y más.

El impacto esperado es significativo tanto en el ámbito académico como en el tecnológico y de investigación, al proveer una solución que une teoría, práctica y visualización, promoviendo así el aprendizaje activo de los estudiantes y la experimentación con problemas complejos (poco estudiados en la facultad) de forma didáctica y efectiva.

Capítulo 1. Introducción

En el campo de la teoría de grafos, uno de los problemas más estudiados es la detección de cliques dentro de un grafo. Un clique es un subconjunto de vértices tal que cada par de vértices está conectado por una arista (ver Figura 1.1). De particular interés es el problema del clique máximo, que consiste en encontrar el subconjunto de vértices más grande que forme un clique en un grafo dado. Este problema es de gran relevancia en múltiples áreas como bioinformática, redes sociales, diseño de circuitos, y análisis de datos [1]. Sin embargo, la complejidad computacional de este problema representa un gran desafío. El problema del clique máximo está clasificado como NP-completo, lo que significa que no se conoce un algoritmo eficiente que lo resuelva en tiempo polinomial para todos los casos [2]. Esta dificultad se acentúa al intentar aplicar estos algoritmos en grafos grandes o en aplicaciones prácticas.

Actualmente, existen diversas soluciones algorítmicas para abordar este problema, que van desde algoritmos exactos en [2], hasta heurísticas y metaheurísticas como algoritmos genéticos, búsqueda tabú o algoritmos basados en colonia de hormigas [3]. No obstante, la mayoría de estas soluciones carece de una interfaz gráfica accesible que permita a estudiantes, investigadores o profesionales visualizar de manera intuitiva los resultados, analizar el comportamiento del algoritmo, e incluso manipular el grafo de entrada. Por lo que consideramos factible desarrollar una aplicación gráfica interactiva que permita generar grafos, ejecutar algoritmos para hallar el clique máximo y visualizar sus resultados, contribuyendo así al estudio y comprensión del problema desde una perspectiva tanto teórica como práctica.

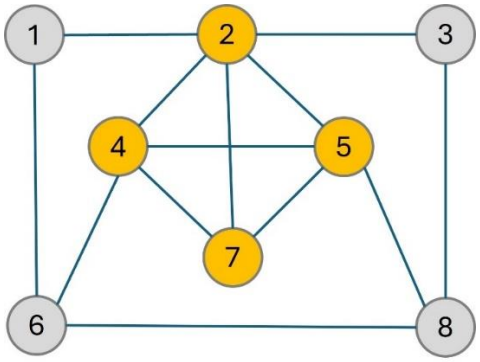

Figura 1.1 Grafo con un clique máximo de cuatro vértices coloreados de amarillo

Un grafo *G = (V, E)* es una estructura matemática compuesta por un conjunto de vértices *V* y un conjunto de aristas *E*, que representan relaciones entre los vértices.

Un clique es un subconjunto de vértices donde cada par está conectado por una arista, es decir, un subgrafo completo. El clique máximo es aquel con el mayor número posible de vértices [4].

Complejidad Computacional: El problema del clique máximo fue uno de los 21 problemas que Richard Karp [1] demostró como NP-completos [10, 11], lo que implica que no se conoce un algoritmo que lo resuelva eficientemente para todos los casos posibles. Por esta razón, en la práctica se utilizan tanto algoritmos [12] exactos como aproximados.

1.1 Planteamiento del problema

El problema del clique máximo [5] es uno de los retos clásicos en la teoría de grafos en ciencias de la computación y su estudio ha sido ampliamente abordado debido a su relevancia teórica y su aplicabilidad en diversos contextos del mundo real. Sin embargo, a pesar de los avances algorítmicos logrados, existe una brecha entre las soluciones computacionales existentes y su comprensión por parte de usuarios no especializados, por lo que al tener una aplicación gráfica permitirá no solo resolver, sino también visualizar y explorar el problema del clique máximo, puede convertirse en una herramienta poderosa tanto para la docencia como para la investigación aplicada, además su desarrollo permitirá vincular conceptos abstractos con representaciones tangibles, lo cual facilita el aprendizaje, la validación de hipótesis y la experimentación con diferentes estructuras de grafos [6, 7, 8, 9].

Algoritmos para encontrar cliques:

- Exactos: Bron-Kerbosch [2] es uno de los más conocidos para hallar todos los cliques máximos en grafos no dirigidos.

- Heurísticos: Algoritmos como GRASP [3] o algoritmos genéticos se utilizan para encontrar buenas soluciones en grafos grandes, sacrificando certeza por eficiencia.

Visualización de grafos:

Herramientas como Gephi o Graphviz permiten visualizar grafos, pero no están diseñadas específicamente para resolver el problema del clique máximo.

1.2 Objetivo general del proyecto de tesis

Desarrollar una aplicación gráfica en web [13, 14] que permita la manipulación de grafos y la ejecución del algoritmo para la detección y visualización del clique máximo, con el fin de apoyar el análisis, la enseñanza y la investigación en teoría de grafos. El presente proyecto se desarrolla como un sistema en web con PHP y las herramientas graficas en JavaScript que permitan la generación y visualización de grafos.

1.3 Objetivos específicos del proyecto de tesis

- Investigar y analizar los principales algoritmos existentes para la detección del clique máximo en grafos.
- Diseñar una arquitectura de software que integre una interfaz gráfica con módulos algorítmicos eficientes.
- Implementar uno de los algoritmos investigados para la detección del clique máximo.
- Desarrollar funcionalidades gráficas que permitan la creación manual o aleatoria de grafos, así como la visualización del proceso y resultado del algoritmo.
- Evaluar el desempeño de la aplicación mediante pruebas con grafos de diferentes tamaños y estructuras.
- Documentar el desarrollo del software y elaborar una guía de usuario para facilitar su uso y comprensión.

1.4 Metodología para el desarrollo del proyecto

La ingeniería de software es esencial para el desarrollo de sistemas de software complejos y de alta calidad, abarca una amplia gama de actividades y disciplinas que contribuyen a la creación de productos de software robustos, eficientes y sostenibles [2].

Enfoque clásico de la Ingeniería de Software. Los modelos clásicos de la ingeniería de software incluyen el modelo en cascada, el modelo en V y el modelo de prototipado, cada uno con enfoques únicos para el desarrollo de software. El modelo en cascada, uno de los más antiguos y conocidos, sigue un proceso secuencial donde cada fase debe completarse antes de iniciar la siguiente, ideal para proyectos con requisitos bien definidos y estables. El modelo en V es una extensión del modelo en cascada que incorpora fases de verificación y validación paralelas a cada etapa del desarrollo, asegurando una mayor calidad del producto. Por otro lado, el modelo de prototipado se centra en la creación rápida de prototipos para comprender y refinar los requisitos del cliente a través de la retroalimentación continua, lo que es especialmente útil cuando los requisitos no están claramente definidos desde el principio. Estos modelos han sido fundamentales en la evolución de las metodologías de desarrollo de software, proporcionando estructuras claras y enfoques disciplinados para la creación de sistemas de software robustos y de alta calidad.

Enfoque ágil de la Ingeniería de Software. El modelo ágil de la ingeniería de software es un enfoque iterativo e incremental que prioriza la flexibilidad, la colaboración y la entrega continua de software funcional. En contraste con los modelos tradicionales, el ágil se adapta rápidamente a los cambios en los requisitos y permite una retroalimentación constante del cliente, lo que mejora la satisfacción del usuario final. Las metodologías ágiles, como Scrum, Kanban y Extreme Programming (XP), fomentan la comunicación abierta y la colaboración estrecha entre todos los miembros del equipo, promoviendo ciclos de desarrollo cortos y la entrega frecuente de incrementos de producto. Esta capacidad para responder rápidamente a las necesidades cambiantes y su enfoque en el valor para el cliente hacen del modelo ágil una opción preferida para proyectos de software complejos y dinámicos.

La metodología para el desarrollo de este trabajo se basa en un enfoque mixto que combina la investigación documental con el desarrollo experimental de software usando la metodología del RAD (Rapid Application Development -por sus siglas en inglés) [15].

El RAD tiene como objetivo reducir el tiempo necesario para desarrollar aplicaciones, enfocándose en la rapidez y flexibilidad, y asegurando que el software cumpla con las expectativas del usuario. A través de un ciclo continuo de retroalimentación, el

desarrollo de la aplicación se va ajustando y mejorando para satisfacer las necesidades del cliente.

Figura 1.2 Diseño visual del RAD

El RAD (figura 1.2) se caracteriza por el desarrollo rápido, ciclos continuos y rápidos, usa el prototipado y pruebas a través de la interacción constante con el cliente [16]. Se caracteriza por cuatro fases:

1. Planificación de Requerimientos. Definir el alcance del sistema y las necesidades del usuario a través de:

- Reuniones con usuarios y analistas.
- Identificación de objetivos, procesos clave y requisitos funcionales.
- Priorización de funcionalidades.

2. Diseño del Usuario (Prototipado). Crear prototipos de interfaces y procesos que permitan al usuario visualizar el sistema:

- Desarrollo de maquetas (mockups) o prototipos interactivos.

- Validación por parte del usuario.
- Retroalimentación continua y ajustes rápidos.

3. Construcción (Desarrollo rápido). Programar e integrar el sistema a partir de los prototipos validados:

- Desarrollo iterativo de módulos de software.
- Pruebas unitarias e integraciones tempranas.
- Corrección de errores en cada iteración.

4. Implementación (Pruebas y Entrega). Pasar el sistema a producción y asegurar que esté listo para usarse:

- Pruebas finales con datos reales.
- Migración de datos desde sistemas anteriores.
- Capacitación de usuarios.
- Instalación y entrega oficial del sistema.

1.5 Trabajos relacionados

Se realizó una revisión de las tesis relacionadas con grafos que han sido realizadas en la Facultad de Ciencias de la Computación y en general en la BUAP a través de la página que está disponible en: (https://repositorioinstitucional.buap.mx/communities/7ba1ec07-d322-4203-b2dc-b9aa1b6dabb7). Se hallaron trabajos de tesis de licenciatura que están relacionadas con la teoría de grafos como los siguientes:

- Una propuesta algorítmica para aproximar el coloreo de grafos. (Benemérita Universidad Autónoma de Puebla, 2014-02) Pérez Gómez, Raymundo; DE ITA LUNA, GUILLERMO; 57559. "Un grafo consiste en un conjunto de nodos o vértices unidos a través de aristas las cuales pueden o no llevar una dirección, esto implica una clasificación de los grafos en grafos dirigidos o no dirigidos, para nuestro caso de estudio trabajaremos... exclusivamente con grafos no dirigidos. El problema sobre el cual se enfoca nuestra investigación e implementación es un algoritmo para poder colorear cualquier grafo no dirigido con la menor cantidad posible de colores, se muestra un grafo con un coloreo propio.

- Un algoritmo heurístico para el coloreo de grafos (Lilibeth Zuñiga Vargas). En esta tesis, se da una breve explicación de las definiciones básicas de la teoría de grafos, y procedimientos básicos y generales utilizados en algoritmos para el coloreo de grafos, ya que son fundamentales para la comprensión de problemas típicos en la teoría de grafos. El motivo de esta tesis es proponer un algoritmo heurístico, para calcular el número cromático de un grafo de forma eficiente, siendo esté el problema medular en el área de Coloreo de grafos.

- Framework educativo para la modulación en la resolución la aplicación de algoritmos con grafos (Lima Estrada, Efraín De Jesús), "Se propone una herramienta de software con la finalidad de resolver problemas sobre caminos más cortos con grafos dirigidos, grafos no dirigidos y árboles n-arios mediante algoritmos básicos y con implementación de técnicas de inteligencia... artificial, permitiendo la interacción con el usuario mediante una interfaz gráfica de fácil uso, facilitándole al usuario ingresar el grafo mediante un lienzo de dibujo para su posterior procesamiento.

- Grafos para procesamiento morfológico (Gallegos Ávila, Pedro Adair). En un grafo los objetos son representados mediante los vértices; mientras que las relaciones entre los objetos son descritas como aristas. En la década de 1960 fue que se dieron los primeros pasos en el estudio de imágenes; en un principio de forma... el capítulo 2, iniciamos con los elementos básicos de la teoría de grafos. Posteriormente, presentamos los operadores erosión y dilatación en un grafo generados por la función de vecindades NG(v).

- Análisis del cómputo exhaustivo de conjuntos independientes (Juan Antares Perdomo Flández). En este trabajo es un método que busca englobar todas las anteriores soluciones, es decir un algoritmo capaz de resolver, para cualquier grafo simple (tipos de grafos que no presentan aristas múltiples, bucles o aristas dirigidas), el conteo de todos los conjuntos independientes lo que a la vez contiene a los conjuntos maximales y máximos. Adicionalmente se añade el muestreo de los conjuntos de forma ordenada y clasificados por tamaños.

- Sistema web para el conteo eficiente de Conjuntos Independientes en estructuras jerárquicas (Árboles). Morales Alcántara, Luis David. Los conjuntos independientes son fundamentales en combinatoria, y tienen aplicaciones en: visión

por computadora, reconocimiento de patrones, planificación, entre otras. Los conjuntos independientes se utilizan también en física estadística, donde... el problema es un caso especial del modelo núcleo-duro (hard-core).

- Algoritmo para determinar la k-coloración de un grafo. García Limón, Olga Lidia. En la teoría de grafos, la coloración de grafos es un caso especial de etiquetado de grafos; es una asignación de etiquetas llamadas colores a los nodos o vértices del grafo. Es decir, una coloración de los vértices de un grafo es una asignación tal que ningún vértice adyacente comparta el mismo color. Si en la coloración se usan k (k=1,2, …, n) colores distintos diremos que es una k-coloración. Una coloración siempre es posible, dado que podemos asignar a cada vértice del grafo un color diferente si fuera necesario, por ejemplo, para grafos completos. Si existe una k-coloración de G se dice que el grafo G es k-coloreable. El valor mínimo k para el que un grafo G es k-coloreable se denomina número cromático de G, y se designa por $\chi(G)$.

Existen diversas herramientas y estudios que abordan el problema del clique máximo desde enfoques algorítmicos y aplicados:

- Cliquer (Östergård, 2002): herramienta eficiente en C para encontrar el clique máximo utilizando búsqueda exhaustiva con poda. Es robusta, pero carece de interfaz gráfica.

- NetworkX (Hagberg et al., 2008): biblioteca de Python para el análisis de grafos. Ofrece funciones para encontrar cliques, pero requiere programación.

- Gephi: software de visualización para grandes redes, centrado en análisis visual, no en resolución algorítmica.

Existen diversas tesis de licenciatura y herramientas en computación desarrolladas respecto a la teoría de grafos y sus aplicaciones sin embargo de la investigación realizada en los repositorios institucionales de tesis y en internet no se encontró algún trabajo de tesis con nuestra propuesta de crear una aplicación grafica para el problema del Clique Máximo sobre grafos.

Capítulo 2. Algoritmos para hallar el Clique

En esta sección se tratará el tema de los diversos algoritmos para hallar el clique máximo de un grafo buscando, este problema es NP-completo, por lo que no se conocen algoritmos eficientes que lo resuelvan en todos los casos para grafos grandes. Sin embargo, existen enfoques exactos como el algoritmo de fuerza bruta, que prueba todas las posibles combinaciones de vértices, y métodos más optimizados como el algoritmo de Bron–Kerbosch, que realiza una búsqueda recursiva sin generar todos los subconjuntos posibles.

También existen algoritmos aproximados y heurísticos, como los basados en algoritmos genéticos o búsqueda tabú, que ofrecen soluciones cercanas al óptimo en un tiempo razonable, siendo útiles para grafos muy grandes donde los métodos exactos no son viables.

2.1 Conceptos básicos de la teoría de grafos

Un Grafo (Figura 2.1) es una estructura matemática usada para modelar relaciones entre objetos. Está compuesto por: Vértices o nodos, que representan los objetos y Aristas o arcos que representan las conexiones entre los objetos.

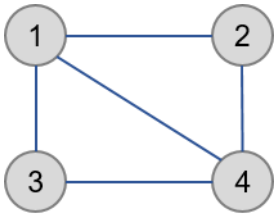

Figura 2.1. Ejemplo de un grafo *G* = {(1,2), (1,3), (1,4), (2,4), (3,4)}

Algunos ejemplos son:

- Un mapa de carreteras, donde las ciudades son los nodos y las carreteras son las aristas
- Una red social, donde las personas son los nodos y las amistades son las aristas

Grafo Dirigido: Las aristas tienen dirección como se muestra en la figura ver ejemplo 2.2.

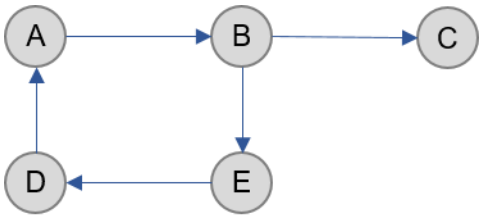

Figura 2.2. Ejemplo de un grafo dirigido

Grafo No dirigido (Grafo): Las conexiones no tienen dirección como se muestra en la Figura 2.3. Por ejemplo, en una red de tráfico o el metro subterráneo, donde las calles son de doble sentido o cada estación del metro está conectada con otras en ambos sentidos.

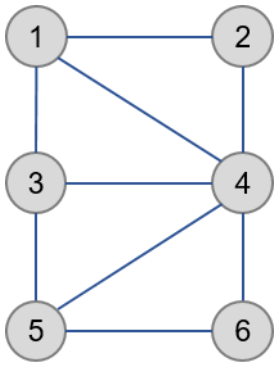

Figura 2.3. Grafo no dirigido

Grafos Mixtos: Algunas conexiones tienen dirección y otras no (Figura 2.4). Por ejemplo, en una Red de transporte público.

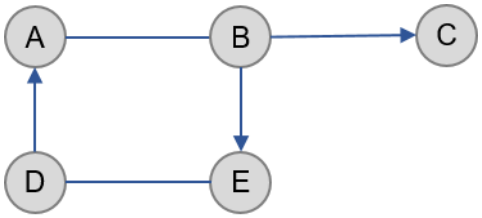

Figura 2.4. Grafo mixto

Grafo Ponderado. En un grafo ponderado (Figura 2.5), las conexiones entre nodos tienen un peso o costo, que se puede representar como la distancia entre las ciudades, tiempo de viaje o costo de alguna transacción.

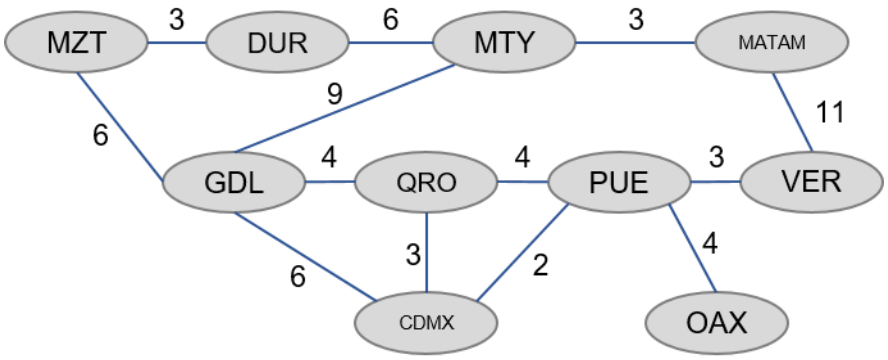

Figura 2.5 Grafo ponderado, representa el tiempo en horas entre cada ciudad

Grado de un Nodo: El grado de un nodo (Figura 2.6 y Figura 2.7), es la cantidad de conexiones que tiene con otros nodos.

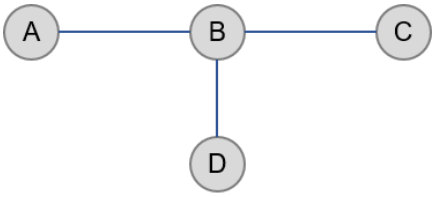

Figura 2.6. Grado de un grafo

- **A** tiene grado 1, porque solo está conectado a B
- **B** tiene grado 3, porque está conectado con A, C y D
- **C** tiene grado 1
- **D** tiene grado 1

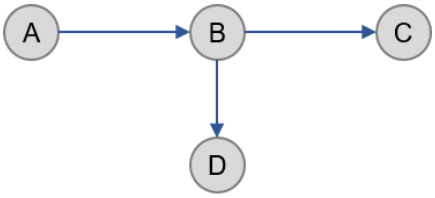

Figura 2.7. Grado de un nodo en un dígrafo

- **A** tiene grado de salida 1, va a B
- **B** tiene grado de entrada 1, viene de A
- **B** tiene grado de salida 2, va a C y C
- **C** tiene grado de entrada 1 y grado de salida 0
- **D** tiene grado de entrada 1 y grado de salida 0

Lista de Adyacencia

La lista de adyacencia es una forma eficiente de representar los datos de grafos dispersos (Figura 2.8), esto es cuando hay pocas conexiones entre los nodos.

Cada nodo tiene una lista de nodos a los que está conectado.

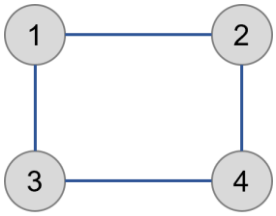

Figura 2.8. Grafo disperso con su lista de adyacencia

Lista de adyacencia

1: [2, 3]

2: [1, 4]

3: [1, 4]

4: [2, 3]

Matriz de Adyacencia

Una matriz de adyacencia (Figura 2.9) usa una matriz de *n* x *n* para representar las conexiones.

- Si hay una conexión entre dos nodos, se pone un 1
- Si no hay conexión, se pone 0
- Si el grafo es ponderado, se pone el peso en lugar de 1

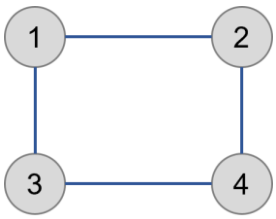

Figura 2.9 Grafo disperso y su matriz de adyacencia

Matriz de adyacencia

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

2.2 Los algoritmos para hallar Cliques

Recordando que un clique es un subconjunto de nodos en un grafo donde todos están conectados entre sí, es decir, forman un subgrafo completo dentro del grafo. En la Figura 2.10 tenemos un grafo yt mostramos los cliques encontrados.

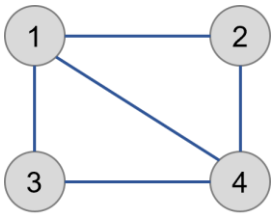

Figura 2.10. Clique de tamaño 2 son: {1, 2}, {1, 3}, {1, 4}, {2, 4}, {3, 4} y Clique de tamaño 3 son: {1, 3, 4} y {1, 2, 4}

En la Figura 2.11. tenemos un grafo donde los nodos están numerados del 1 al 8. El clique de tamaño 6 incluye los nodos: {1, 2, 3, 4, 5, 6}

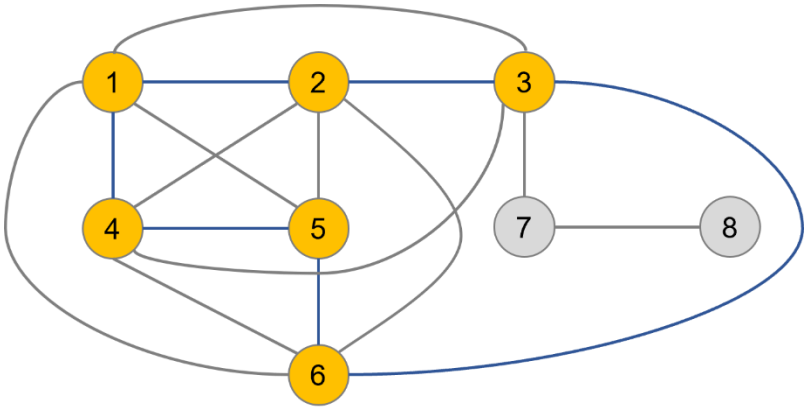

Figura 2.11 Grafo de 8 nodos con un clique máximo de tamaño 6

Clique Máximo

Un clique máximo es el clique más grande posible en un grafo.

2.3 Algoritmo de Clique Máximo: Exhaustivo (Fuerza Bruta)

El método Exhaustivo es la base teórica del problema: propone revisar cada posible solución y elegir la mejor. Aunque es computacionalmente inviable para grafos grandes, es el más simple de entender y el punto de partida para todos los demás algoritmos.

Este enfoque describe el algoritmo con el rigor de la Teoría de Grafos, usando notación estándar.

El objetivo es encontrar el subconjunto de vértices S tal que $S \subseteq V$ y sea de tamaño máximo, formando un subgrafo completo (un clique) en el grafo G.

Glosario de Símbolos

Símbolo	Definición
$G = (V, E)$	El Grafo (la red completa). Se compone de V (todos los Vértices o nodos) y E (todas las Aristas o conexiones).
S	El Subconjunto de vértices (grupo de nodos) que se está revisando en la iteración actual.
C_{max}	El Clique Máximo (la mejor solución) encontrado hasta ahora.
$S \subseteq V$	Significa que S es un subconjunto de todos los vértices del grafo V.
$\{u, v\} \in E$	Significa que existe una arista (conexión) entre los dos vértices u y v.

Estructura y Procedimiento general

1. Inicialización: Se define $C_{max=}\emptyset$ (el clique máximo inicia vacío).
2. Iteración del Conjunto Potencia (2^V):
    - Para cada subconjunto posible S de vértices que puede formarse con V. Esto implica generar y evaluar $2^{|V|}$ posibles subconjuntos.
3. Verificación de Clique: se comprueba si $|S|$ es un clique. Se verifica que para cada par de vértices $\{u, v\} \in S$, debe existir una arista $\{u, v\} \in E$ (deben estar conectados).
4. Actualización: Si S es un clique y su tamaño $|S|$ es mayor que el tamaño de $|C_{max}|$, entonces $C_{max} = S$.

2.4 Algoritmo de Clique Máximo: Bron-Kerbosh (Backtracking)

El algoritmo de Bron-Kerbosh (1973) es un método fundamental de backtracking (búsqueda en retorno) que se usa para enumerar todos los cliques maximales de un grafo. Esto significa que encuentra todos los subconjuntos que no pueden crecer más añadiendo un solo vértice. Una vez encontrados todos, el clique

máximo es simplemente el más grande de ellos. Es significativamente más rápido que la Fuerza Bruta.

Este enfoque describe el algoritmo a través de una función recursiva basada en tres conjuntos de vértices.

El objetivo es enumerar todos los cliques maximales R. La versión más grande encontrada será el clique máximo, ω(R).

Glosario de Símbolos

Símbolo	Definición
$G = (V, E)$	El Grafo (la red completa). Se compone de V (todos los Vértices o nodos) y E (todas las Aristas o conexiones).
R	El conjunto de vértices que forman el Clique Actual (el que se esta construyendo).
P	Los Candidatos Posibles a añadir a R. Solo contiene vértices conectados a los vértices en R.
X	Los vértices Excluidos (ya visitados). Se usan para asegurar que el clique R sea maximal y para evitar revisitar caminos.
$N(v)$	El vecindario de un vértice R. Son todos los vértices directamente conectados a v.
$R \cup \{v\}$	Union: El nuevo clique R con el vértice v añadido.
$P \setminus \{v\}$	Diferencia: El conjunto P sin el vértice v.
$P \cap N(v)$	Intersección: Solo los vértices que están en P y también están conectados a v.

Estructura y Procedimiento general

1. Caso Base (Clique Maximal Encontrado):
    - Si $P = \emptyset$ (no quedan candidatos) y $X = \emptyset$ (no quedan excluidos que revisar)
        - Entonces R es un clique maximal. Se guarda R y se verifica si es el C_{max} general.
2. Iteración y recursión:
    - Para cada vértice v en el conjunto de Candidatos P:
        - Llamada Recursiva: BronKerbosh$\big(R \cup \{v\},\ P \cap N(v),\ X \cap N(v)\big)$
        - Mover a Excluidos: Se actualizan los conjuntos moviendo v de P a X: $P = P \setminus \{v\}$ y $X = X \cup \{v\}$.

2.5 Algoritmo de Clique Máximo: Tomita - Optimización con Pivote

El algoritmo de Tomita, es una de las implementaciones más eficientes del algoritmo de backtraking para encontrar todos los cliques maximales. Se basa directamente en Bron-Kerbosh pero añade la estrategia de Vértice de Pivote para podar (descartar) el espacio de búsqueda.

El enfoque formal de Tomita es idéntico a Bron-Kerbosh, excepto por la adición del Pivote ***(u)*** en la fase de iteración y recursión.

El objetivo es enumerar todos los cliques maximales R. La versión más grande encontrada será el clique máximo, ω(G).

Glosario de Símbolos

Símbolo	Definición
R, P, X	Los mismos conjuntos que en Bron-Kerbosh: R (Clique Actual), P (Candidatos), X (Excluidos).
u	El Vértice Pivote. Un vértice estratégicamente elegido de $P \cup X$.
$N(u)$	El Vecindario (vecinos directos) del Pivote u.
$P \setminus N(u)$	Conjunto de Iteración: Los vértices en P que no son vecinos del Pivote u.

Estructura y Procedimiento general

1. Caso Base (Clique Maximal Encontrado):
    - Si $P = \emptyset$ (no quedan candidatos) y $X = \emptyset$ (no quedan excluidos que revisar):
        - Entonces R es un clique maximal. Se guarda R y se verifica si es el C_{max} general.
2. Selección del Pivote:
    - Seleccionar un vértice Pivote u en $P \cup X$ (idealmente, u es el nodo en $P \cup X$ con la mayor cantidad de vecinos en P).
3. Iteración y Poda (Pruning):
    - Para cada vértice v en el conjunto de iteración $P \setminus N(u)$:
        - Llamada Recursiva:
          Tomita$\big(R \cup \{v\},\ P \cap N(v),\ X \cap N(v)\big)$

- Mover a Excluidos (Backtracking): Se actualizan los conjuntos moviendo v de P a X: $P = P \setminus \{v\}$ y $X = X \cup \{v\}$.

2.6 Algoritmo de Clique Máximo: Branch and Bound (Ramificación y Poda)

El enfoque Branch and Bound (B&B) no se limita a enumerar todos los cliques maximales (como Bron-Kerbosh), sino que se enfoca directamente en encontrar el clique máximo $\omega(G)$. Su eficiencia radica en el uso de una Cota Superior (Upper Bound) para podar (descartar) ramas enteras del árbol de búsqueda que nunca podrán llevar a una solución mejor que la mejor encontrada hasta el momento.

El método se basa en una función recursiva que se "poda" si la mejor solución posible en la rama actual no supera el récord global.

El objetivo es encontrar el clique R tal que $|R|$ sea el máximo posible.

Glosario de Símbolos

Símbolo	Definición
R	El conjunto de vértices que forman el Clique Actual (el que se está construyendo).
P	Los Candidatos Posibles a añadir a R.
$U(P)$	Cota Superior (Upper Bound): El tamaño máximo posible de un clique que se puede formar usando solo vértices en P.
S_{max}	El Tamaño del Mejor Clique encontrado hasta el momento (el récord global).
$X(G)$	Número Cromático: El mínimo número de colores necesarios para colorear el grafo G. Es la cota superior más utilizada, ya que $\omega(G) \leq X(G)$.

Estructura y Procedimiento general

1. Regla de Poda (Pruning Rule):
   - Si $|R| + U(P) \leq S_{max}$:
     - Detener la búsqueda. El clique actual $|R|$ más el tamaño máximo que podría lograrse con los candidatos restantes $U(P)$ no supera el récord global S_{max}. No tiene sentido explorar esta rama.
2. Actualización de la Solución (Récord):

- Si $|R| > S_{max}$:
  - $S_{max} = |R|$. Guardar R como el nuevo clique máximo global.

3. Iteración y Recursión:
   - Para cada vértice v en el conjunto de Candidatos P:
     - Llamada Recursiva: BranchAndBound$\big(R \cup \{v\},\ P \cap N(v)\big)$.
     - Backtracking: $P = P \setminus \{v\}$.

2.7 Algoritmo de Clique Máximo: Voraz o Greedy (Aproximación)

El algoritmo Voraz (Greedy) es un enfoque de aproximación diseñado para encontrar un clique grande rápidamente, aunque no garantiza que sea el clique máximo ω*(G).* Se utiliza en grafos muy grandes donde la lentitud de los métodos exactos es inaceptable. Su filosofía es simple: en cada paso, se toma la decisión que parece mejor en ese momento.

El método construye el clique de forma iterativa, seleccionando en cada paso el vértice que tiene el potencial localmente más alto para extender el clique.

El objetivo es construir un clique R lo más grande posible. Se busca un clique aproximado al óptimo.

Glosario de Símbolos

Símbolo	Definición
R	El conjunto del Clique Actual que se está construyendo.
V^1	El conjunto de Vértices Restantes que son candidatos a ser añadidos a R.
$N(v)$	El Vecindario de un vértice v (sus vecinos directos).
$N_R(v)$	Ganancia Local: La métrica clave. Es el número de vecinos de v que aún están en V^1.
$v^* \in V^1$	El vértice candidato que tiene la ganancia local más alta.

Estructura y Procedimiento

1. Inicialización: $R = \emptyset$ y V^1 contiene todos los vértices del grafo G.
2. Bucle de Construcción:

- Mientras V^1 no este vacío (quedan candidatos):
    - Seleccionar el vértice $v^* \in V^1$ con la Ganancia Local máxima.
    - Añadir a R: $R = R \cup \{v^*\}$.
    - Actualizar V^1: El nuevo V^1 será la intersección del V^1 antiguo con el vecindario de v^*, excluyendo al propio v^*.

3. Fin: El conjunto R es el clique aproximado.

2.8 Algoritmo de Clique Máximo: GRASP (Metaheurística)

GRASP (Greedy Randomized Adaptive Search Procedure) es una metaheurística que busca un balance entre velocidad de los algoritmos Voraces y la calidad de los algoritmos exactos. Es un método de aproximación avanzado que itera múltiples veces, combinando un paso de construcción voraz aleatorizada con un paso de mejora local.

El algoritmo opera en ciclos, donde cada ciclo genera una nueva solución inicial (fase de construcción) y luego intenta mejorarla (fase de mejora local).

Glosario de Símbolos

Símbolo	Definición
R	El conjunto del Clique Actual que se está construyendo.
V^1	El conjunto de Vértices Restantes elegibles para ser añadidos a R.
RCL	Restricted Candidate List: Una lista de candidatos que son casi tan buenos como el mejor, permitiendo una selección aleatoria y no estrictamente voraz.
R_{mejor}	El Mejor Clique encontrado después de todas las iteraciones.
S^*	Solución mejorada por la Mejora Local.

Estructura y Procedimiento general

1. Inicialización: $R_{mejor} = \emptyset$.
2. Bucle de Iteraciones:
    - Para cada iteración (ej. 100 veces):
        - Fase 1: Construcción Aleatorizada:
          $R_{inicial} = ConstruirVorazAleatorio(G)$.

- Fase 2: Mejora Local:
  $R_{final} = MejoraLocal(R_{inicial})$.

- Actualización:
  - Si $|R_{final}| > |R_{mejor}|$:
    - $R_{mejor} = R_{final}$.

3. Fin: Devolver R_{mejor}.

2.9 Aplicaciones del Clique

El enfoque en encontrar cliques (subgrafos completos) dentro de una red compleja se utiliza para la detección de estructuras y el agrupamiento local. Mientras que el Clique Máximo busca el récord global, el análisis de todos los cliques es fundamental para mapear módulos o comunidades. A continuación, se enlistas algunas aplicaciones del Clique.

- Detección de Módulos y Agrupamiento
- Identificación de Comunidades en Redes Sociales
- Análisis de Compatibilidad en Bases de Datos
- Análisis de Vías Bioquímicas (Bioinformática)
- Planificación y Consistencia de Sistemas
- Verificación de Consistencia de Reglas (Sistemas Expertos)
- Detección de Bugs en Software

2.10 Aplicaciones del Clique Máximo

- Optimización en Bioinformática y Estructuras Moleculares
- Detección de la Subestructura Común Máxima
- Análisis de Polimorfismos Genéticos Consistentes
- Planificación y Asignación de Recursos (Maximización de Uso)
- Planificación de Horarios con Máxima Ocupación
- Agrupamiento para Marketing o Publicidad (Segmentación Máxima)
- Resolución de Conflictos y Sistemas de Seguridad
- Reducción Mínima de Interferencias (Telecomunicaciones)
- Reconstrucción Óptima de Imágenes (Visión por Computadora)

Capítulo 3. Algoritmo propuesto

Como se mencionó en el capítulo anterior, el algoritmo BronKerbosch es un método clásico utilizado en teoría de grafos para encontrar todos los cliques máximos de un grafo no dirigido.

Recordemos que un clique es un conjunto de vértices que están completamente conectados entre sí y un clique máximo es aquel que no puede ampliarse agregando otro vértice adyacente a todos los del conjunto.

Este algoritmo funciona mediante una técnica de búsqueda recursiva con backtracking, explorando combinaciones de vértices y extendiendo progresivamente conjuntos candidatos que podrían formar cliques. En cada paso, se manejan tres conjuntos: *A*, el conjunto actual de vértices en el clique; *B* los vértices que pueden ampliarlo y *C* los vértices que ya se han considerado. Cuando los conjuntos *B* y *C* están vacíos, se ha encontrado un clique máximo.

A pesar de que la complejidad de este algoritmo es exponencial en el peor caso, es muy bueno en la práctica y constituye la base de muchas variantes y optimizaciones modernas. A continuación, se describe el algoritmo general de BronKerbosh.

BronKerbosch (*A, B, C*)

{

Si (*B* y *C*) están vacíos,

Entonces informa *A* como un clique máximo

Para cada vértice *v* en *B*, Hacer

BronKerbosch (*A* ∪ {*v*}, *B* ∩ *N* (*v*), *C* ∩ *N* (*v*))

P ← *P* \ {*v*}

C ← *C* ∪ {*v*}

}

3.1 Pseudocódigo del algoritmo completo

//Función utilizada para el Algoritmo Bron-Kerbosch

```
Function BronKerbosch(R, P, X, Grafo, REFERENCIA MaxClique)
Inicio
   // Caso base: si no hay vértices para expandir ni excluir
   SI (P está vacío) Y (X está vacío) ENTONCES
      SI tamaño(R) > tamaño(MaxClique) ENTONCES
         MaxClique ← copia de R
      FIN SI
      RETORNAR
   FIN SI

   // Elegir pivote u del conjunto (P ∪ X)
   U ← NULO
   Union ← unión(P, X)
   SI Union no está vacío ENTONCES
      U ← primer elemento de Union
   FIN SI

   // Vecinos del pivote
   SI U ≠ NULO Y Grafo contiene U ENTONCES
      VecinosU ← Grafo[U]
   SINO
      VecinosU ← conjunto vacío
   FIN SI

   // Candidatos: vértices de P que no son vecinos de U
   Candidatos ← diferencia(P, VecinosU)

   // Explorar recursivamente cada candidato
   PARA cada vértice v EN Candidatos HACER
      SI Grafo contiene v ENTONCES
         VecinosV ← Grafo[v]
      SINO
         VecinosV ← conjunto vacío
      FIN SI

      // Llamada recursiva con vértice agregado
      BronKerbosch(
         R ∪ {v},
         intersección(P, VecinosV),
         intersección(X, VecinosV),
         Grafo,
         MaxClique
      )

      // Actualizar conjuntos P y X
      P ← P - {v}
```

```
        X ← X ∪ {v}
    FIN PARA
FIN PROCEDIMIENTO
```

//Pseudocódigo del algoritmo principal para hallar el Clique máximo

```
Algoritmo Clique_Máximo
INICIO
        PARA i<-1 HASTA LEN(P) HACER
        Bandera<-0
            PARA j<-1 HASTA LEN(vecinosU) HACER
            SI (P[i]=vecinosU[j] AND vecinosU<>0)
                    bandera<-1
            FIN_SI
            FIN_PARA
            SI (bandera<>1) ENTONCES
                longitud<-longitud+1
                candidatos[longitud]<-P[i]
            FIN_SI
         FIN_PARA

        PARA m<-1 HASTA LEN(candidatos) HACER
              PARA i<-1 HASTA LEN(R) HACER
                  nuevoR[i]<-R[i]
              FIN_PARA
              nuevoR[LEN(R)+1]<-candidatos[m]
               longitud<-0
               PARA i<-1 HASTA LEN(P) HACER
                  PARA j<-1 HASTA LEN(vecinosV) HACER
                     SI (P[i]=vecinosV[j])
                        longitud<-longitud+1
                        nuevoP[longitud]<-P[i]
                     FIN_SI
                  FIN_PARA
               FIN_PARA

               longitud<-0
               PARA i<-1 HASTA LEN(X) HACER
                  PARA j<-1 HASTA LEN(vecinosV) HACER
                     SI (X[i]=vecinosV[j])
                        longitud<-longitud+1
                        nuevoX[longitud]<-X[i]
                     FIN_SI
                  FIN_PARA
               FIN_PARA

// Llamado función
bronKerbosch(nuevoR, nuevoP, nuevoX, grafo, maxClique)
```

```
longitud<-0
PARA i<-1 HASTA LEN(P) HACER
    SI(P[i]<>candidatos[m])
        longitud<-longitud+1
        tempP[longitud]<-P[i]
    FIN_SI
FIN_PARA
P<-tempP
PARA i<-1 HASTA LEN(X) HACER
        tempX[i]<-X[i]
FIN_PARA
tempX[LEN(X)+1]<-candidatos[m]
X<-tempX
FIN_PARA

// Pseudocodigo de la función principal

FUNCION MAIN () RET:vacio
ESCRIBIR “Ingresar número de nodos”
LEER numNodos
Dimension nodos[numNodos]
PARA i<-1 HASTA numNodos HACER
                        nodos[i] <- i
FIN_PARA
Dimension matrizAdyacencias[numNodos][ numNodos]
ESCRIBIR “Ingresar Matriz de Adyacencias”
PARA i<-1 HASTA numNodos HACER
      PARA j<-1 HASTA numNodos HACER
        ESCRIBIR “valor  ”,i,” “,j,”:”
        LEER valor
        SI (valor =1) ENTONCES
                matrizAdyacencias[i][j]<-j
        SINO
                matrizAdyacencias[i][j]<-0
        FIN_SI
     FIN_PARA
FIN_PARA

bronKerbosch([], nodos, [], matrizAdyacencias, maxClique);
FIN
```

3.2 Prueba de escritorio del algoritmo de planificación

El algoritmo de Bron-Kerbosch es un algoritmo recursivo de "backtracking" (retroceso) que encuentra eficientemente todos los cliques máximos en un grafo. Donde el clique máximo es un subconjunto de vértices en un grafo que cumple dos condiciones:

1. Cada vértice en el subconjunto está conectado por una arista a todos los demás vértices dentro de ese mismo subconjunto (un subconjunto completamente conectado).
2. No se puede añadir ningún otro vértice al subconjunto sin romper la propiedad de conectividad completa.

El algoritmo va construyendo el clique máximo con un vértice a la vez, considerando los vecinos de cada nodo, y posteriormente los vecinos de los vecinos, con cada llamada recursiva.

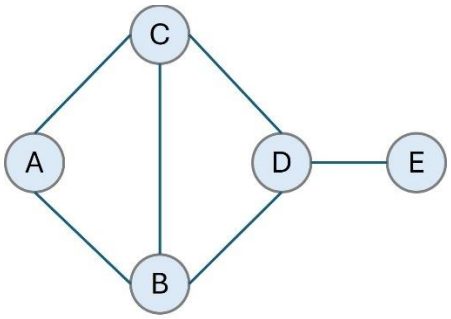

Figura 3.1. Grafo ejemplo para el Clique

Ejemplo: sea el grafo de la Figura 3.1. vamos a calcular el Clique máximo paso a paso, De acuerdo con el grafo, los vértices V = {A, B, C, D, E} y las aristas E = {(A, B), (A, C), (B, C), (B, D), (C, D), (D, E),)}

1. Inicializamos el algoritmo. El algoritmo Bron–Kerbosch (R, P, X) busca cliques máximos con:

   R = conjunto actual del clique (inicialmente vacío)
   P = vértices candidatos (inicialmente todos los vértices)
   X = vértices ya explorados (inicialmente vacío)
   R = {}
   P = {A, B, C, D, E}
   X = {}

2. Llamada recursiva por cada vértice en P
   Tomamos A como pivote
   Vecinos de A: {B, C}
   Llamada:
   R = {A}
   P = {B, C}
   X = {}

3. Expandimos el clique con B
   Vecinos(B) = {A, C, D}
   Intersecciones:

$P \cap N(B) = \{C\}$
$X \cap N(B) = \{\}$

Nueva llamada:
$R = \{A, B\}$
$P = \{C\}$
$X = \{\}$
Agregamos C
$Vecinos(C) = \{A, B, D\}$
$P \cap N(C) = \{\}$
$X \cap N(C) = \{\}$

Clique encontrado: {A, B, C}

Retrocedemos:
Añadimos C a X → $X = \{C\}$
Eliminamos C de P → $P = \{\}$

4. Retroceso y exploración con otros vértices
   Volvemos a:
   $R = \{A, B\}$ → ya no hay vértices en P → fin de rama.
   Retrocedemos más.
   Ahora en la llamada
   $R = \{A\}$, siguiente vértice: C
   $Vecinos(C) = \{A, B, D\}$
   $P \cap N(C) = \{B\}$
   $X \cap N(C) = \{\}$
   Llamada:
   $R = \{A, C\}$
   $P = \{B\}$
   $X = \{\}$
   Agregamos B → clique {A, B, C} (ya encontrado).

   Retrocedemos.

5. Paso 6. Continuamos con D
   $Vecinos(D) = \{B, C, E\}$
   $R = \{D\}$
   $P = \{B, C, E\}$
   $X = \{A\}$
   Probamos con B → {B, C, D} también forma un clique.
   Segundo clique máximo: {B, C, D}
   E con D no forma clique con B ni C → descartado.

6. Último vértice E

Vecinos(E) = {D}
No puede formar un clique mayor que tamaño 2.

Resultado final
Los cliques máximos son de tamaño 3:
{A, B, C}
{B, C, D}

3.3 Complejidad del Algoritmo propuesto

De acuerdo con la teoría de la complejidad computacional (ver Figura 3.1), las clases P y NP se utilizan para clasificar los problemas según la dificultad de resolverlos o verificarlos usando un algoritmo. P son problemas que se pueden resolver y verificar rápido, mientras que NP son problemas que se pueden verificar rápido, aunque encontrarlos pueda ser muy costoso.

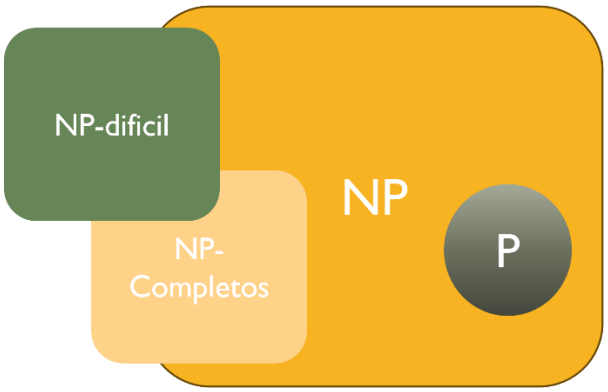

Figura 3.1. Clases principales de complejidad algorítmica

La clase P (en ingles Polynomial time) agrupa todos los problemas que pueden resolverse en un tiempo polinomial, es decir, aquellos para los cuales existe un algoritmo que encuentra la solución en un tiempo que crece como una potencia del tamaño de la entrada *n*, por ejemplo: n^2, n^3, etc. Los problemas en P se consideran "eficientemente resolubles" por una computadora.

Ejemplo de clase P.

El problema de ordenar un conjunto de números aplicando el algoritmo de Quicksor pertenece a la clase P, es decir es del orden

$O(n \log n)$. Es decir, resolverlo y verificar la solución es rápido y eficiente.

La clase NP (en ingles Nondeterministic Polynomial time) incluye todos los problemas cuyas soluciones pueden verificarse en tiempo polinomial, aunque no necesariamente puedan encontrarse en ese tiempo. Es decir, si alguien nos da una posible solución, podemos comprobar fácilmente si es correcta, pero encontrar la solución o soluciones ya no se puede hacer en tiempo polinomial.

Ejemplo de clase NP.

El problema del Clique Máximo pertenece a NP. Dado un grafo, si alguien nos da un conjunto de vértices del grafo, podemos verificar rápidamente si forman un clique (todos los vértices conectados entre sí) en tiempo polinomial, pero encontrar el clique más grande en el grafo no tiene un algoritmo conocido que funcione en tiempo polinomial, debido a que la búsqueda puede requerir explorar combinaciones exponenciales de todos los vértices.

El problema de hallar el Clique Máximo en grafos es uno de los problemas más estudiados en teoría de la computación y pertenece a la clase NP y en especial a los problemas NP-completos. Se dice que un problema es NP-completo cuando pertenece al conjunto NP, pero, además, es tan difícil como cualquier otro problema en NP, lo que significa que el problema de Clique es un problema característico de la clase NP.

Decidir si existe un clique de tamaño k en un grafo con n vértices es un problema NP-completo, lo que significa que no se conoce un algoritmo que lo resuelva en tiempo polinomial para todos los casos.

Encontrar el clique máximo (es decir, el más grande de todos los posibles) es incluso más difícil: es un problema NP-difícil (NP-hard), ya que no solo requiere verificar una solución, sino buscarla entre un número potencialmente exponencial de combinaciones.

En el peor caso, los algoritmos exactos como Bron–Kerbosch, tienen una complejidad exponencial, aproximadamente $O(2^n)$, porque deben considerar subconjuntos de vértices.

Los algoritmos aproximados o heurísticos pueden encontrar cliques grandes en menor tiempo, aunque no garantizan que sea optima la solución encontrada.

Capítulo 4. Sistema web para hallar el Clique máximo

En esta sección se presenta el desarrollo de la aplicación web visual para calcular el *clique máximo* en grafos es una herramienta interactiva y visual que permite a los usuarios cargar el archivo con los datos de un grafo y generar el clique máximo y graficar el grafo resaltando el clique máximo, además la aplicación permite generar grafos aleatorios para realizar diversos estudios de este problema. La visualización es completamente interactiva: los nodos se pueden arrastrar, ampliar y reducir, lo que facilita la comprensión de la estructura del grafo. Una vez definido el grafo, la aplicación ejecuta un algoritmo de Bron–Kerbosch para encontrar el *clique máximo*, es decir, el subconjunto más grande de nodos completamente conectados entre sí. Este cálculo se realiza en un servidor local apache. Una vez encontrado el clique, los nodos correspondientes se resaltan visualmente en la interfaz, permitiendo al usuario identificar fácilmente este subconjunto.

Este sistema tiene aplicaciones en campos como teoría de grafos, redes sociales, biología computacional y análisis de redes complejas, ya que el problema del clique máximo es fundamental en muchos escenarios donde se necesita identificar grupos fuertemente conectados. Además, al tratarse de una aplicación web puede ser utilizada desde cualquier dispositivo con conexión a Internet, lo que la hace ideal para propósitos educativos, de investigación o incluso profesionales.

4.1 Herramientas y Lenguaje de desarrollo utilizadas

Para la implementación del algoritmo Bron–Kerbosch para hallar el clique máximo se utilizó un servidor Web (Apache), el lenguaje de programación del lado del servidor PHP, HTML para la vista, CSS para el diseño de la aplicación y la librería Vis.js que permite visualizar los grafos de forma gráfica. De esta forma se utilizó:

- Servidor Apache
- PHP
- HTML
- CSS
- Vis.js
- Formato Json

- Archivos de texto

A continuación, se describen los elementos principales.

4.1.1 Servidor Apache

El servidor web es un componente fundamental en el funcionamiento de las aplicaciones web, ya que actúa como intermediario entre el cliente (normalmente un navegador) y el contenido o servicios alojados en un servidor. Su principal función es recibir solicitudes HTTP del cliente, procesarlas y devolver la respuesta correspondiente, que puede ser una página web, un archivo, una imagen o el resultado de una operación dinámica.

Dentro de los servidores web más utilizados, Apache HTTP Server, comúnmente conocido como Apache, ha desempeñado un papel crucial desde los inicios de la web moderna. Su importancia radica en sus características:

- Robustez,
- Flexibilidad
- Compatibilidad

Apache (Figura 4.1) es un servidor de código abierto que ha sido clave en la expansión de internet, permitiendo a millones de desarrolladores y empresas alojar sitios y aplicaciones web de forma segura y eficiente. Debido a su arquitectura modular, permite integrar funcionalidades avanzadas como la gestión de certificados SSL, la reescritura de URLs, el manejo de múltiples sitios (virtual hosts), y la ejecución de scripts mediante tecnologías como PHP, Python o Perl.

Figura 4.1. Logo del servidor web Apache
(https://plataforma.josedomingo.org/pledin/cursos/apache24/)

La constante evolución de Apache respaldada por una amplia comunidad, lo mantiene vigente como una de las opciones más

confiables para proyectos de todo tipo, desde sitios personales hasta grandes infraestructuras empresariales.

Para la implementación del sistema se utilizó la herramienta Wampserver.

WampServer es un paquete de software gratuito que permite instalar en un entorno Windows un servidor web local (como se muestra en la Figura 4.2) para el desarrollo y prueba de aplicaciones. Su nombre proviene de las siglas Windows, Apache, MySQL/MariaDB y PHP, ya que integra estos componentes en una sola instalación sencilla, además de herramientas adicionales como phpMyAdmin para la administración de bases de datos. Con WampServer los desarrolladores pueden crear, configurar y probar sitios web o sistemas antes de publicarlos en un servidor en línea, lo que lo convierte en una herramienta muy útil para el aprendizaje, la programación web y el trabajo en proyectos de desarrollo de software.

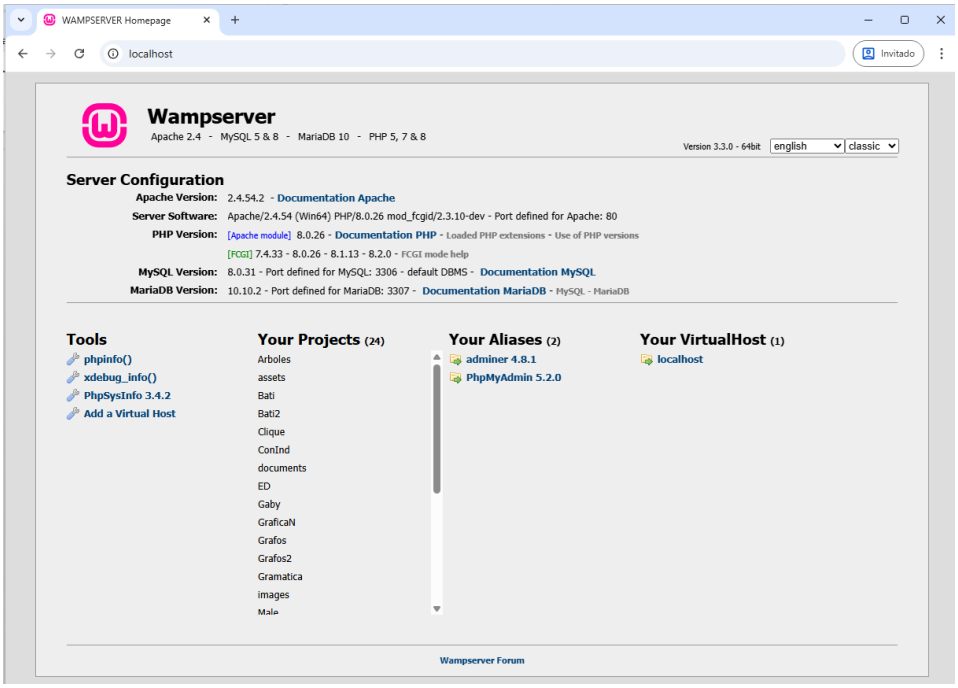

Figura 4.2. Herramienta de desarrollo Wampserver instalado como servidor local

4.1.2 Lenguaje del lado del servidor PHP

Los lenguajes del lado del servidor son esenciales en el desarrollo de aplicaciones web dinámicas, ya que permiten procesar la lógica de negocio, interactuar con bases de datos, gestionar sesiones de

usuario, validar formularios y generar contenido personalizado antes de enviarlo al navegador del cliente.

A diferencia del código del lado del cliente, que se ejecuta en el navegador, el código del lado del servidor se ejecuta en el servidor web, lo que garantiza mayor seguridad y control sobre los datos.

Entre los lenguajes más utilizados en este ámbito, PHP (Figura 4.3) ha sido históricamente uno de los más influyentes y populares. Su importancia radica en su simplicidad, facilidad de integración con HTML y bases de datos como MySQL, y su compatibilidad con la mayoría de los servidores web, especialmente Apache. PHP ha sido la base de numerosos sistemas de gestión de contenido (CMS) como WordPress, Drupal y Joomla, que alimentan una gran parte del contenido web actual.

PHP es de código abierto soportado por una gran comunidad que han permitido el desarrollo de frameworks modernos como Laravel y Symfony, que ofrecen herramientas avanzadas para construir aplicaciones robustas, seguras y escalables.

Figura 4.3. modelo de funcionamento de PHP (https://www.aprenderaprogramar.com/)

4.1.3 Lenguaje marcas HTML y de diseño CSS

HTML y CSS son pilares fundamentales en el desarrollo de cualquier aplicación web, ya que se encargan de la estructura y presentación visual de la interfaz con la que interactúan los usuarios. HTML (HyperText Markup Language) define el contenido y la organización de los elementos en una página web, como textos,

imágenes, botones, formularios y enlaces, actuando como el esqueleto del sitio.

CSS (Cascading Style Sheets) se encarga de dar estilo y diseño a ese contenido, permitiendo controlar aspectos como colores, tipografías, tamaños, márgenes, alineaciones y adaptabilidad a diferentes dispositivos.

La combinación de ambos lenguajes permite crear interfaces atractivas, accesibles y bien organizadas, lo cual es clave para ofrecer una buena experiencia de usuario.

El CSS moderno incluye capacidades como animaciones, transiciones y diseño responsivo, que son esenciales para aplicaciones web que se usan en múltiples plataformas y tamaños de pantalla. Sin HTML y CSS (Figura 4.4), incluso las aplicaciones web más avanzadas carecerían de estructura visual y resultarían inusables para el usuario final.

Figura 4.4. Logos de HTML y CSS (https://niixer.com/index.php/2024/02/15/css-en-html-combinalo-para-potenciar-tu-diseno-web/#google_vignette)

4.1.4 Herramienta de visualización de grafos

Para la visualización de los grafos en forma gráfica existen diferentes herramientas esenciales para representar datos complejos en forma de nodos (entidades) y aristas (relaciones) de manera clara e interactiva, facilitando su análisis y comprensión. Estas herramientas permiten observar la estructura de redes como redes sociales, sistemas de transporte, relaciones biológicas,

estructuras de internet o modelos matemáticos, entre otros. Entre las más populares se encuentran: Cytoscape.js, Sigma.js, D3.js y Vis.js, cada una con características específicas que se adaptan a distintos tipos de proyectos. Estas bibliotecas suelen ofrecer funciones como el arrastre de nodos, el zoom, la personalización de estilos, y la capacidad de responder a eventos del usuario.

En particular en el proyecto utilizamos Vis.js que es una biblioteca de JavaScript muy utilizada para la visualización de grafos de forma simple e intuitiva. Su componente más conocido, vis-network, permite crear gráficos interactivos donde los nodos pueden moverse libremente, y el diseño del grafo se actualiza automáticamente con algoritmos de físicas para una distribución más natural.

Vis.js (ver Figura 4.5) destaca por su facilidad de uso, integración sencilla en proyectos web, y soporte para características como etiquetas personalizadas, agrupación de nodos, jerarquías, y eventos como clics o selecciones. Es muy eficiente para proyectos educativos, prototipos o aplicaciones web donde se requiere una visualización rápida y funcional de grafos sin configuraciones complejas.

Vis.js fue desarrollado originalmente por Almende B.V., una empresa de investigación tecnológica con sede en los Países Bajos, enfocada en soluciones de software descentralizado e inteligencia distribuida. El desarrollo principal de la biblioteca fue liderado por Alex de Mulder, junto con contribuciones de varios desarrolladores de la comunidad.

Vis.js fue concebido como una biblioteca modular para la visualización de datos dinámicos, incluyendo componentes como gráficos de redes (grafos), líneas de tiempo, gráficos en 2D y otros. Su componente más conocido, vis-network, es ampliamente utilizado para la visualización de grafos interactivos.

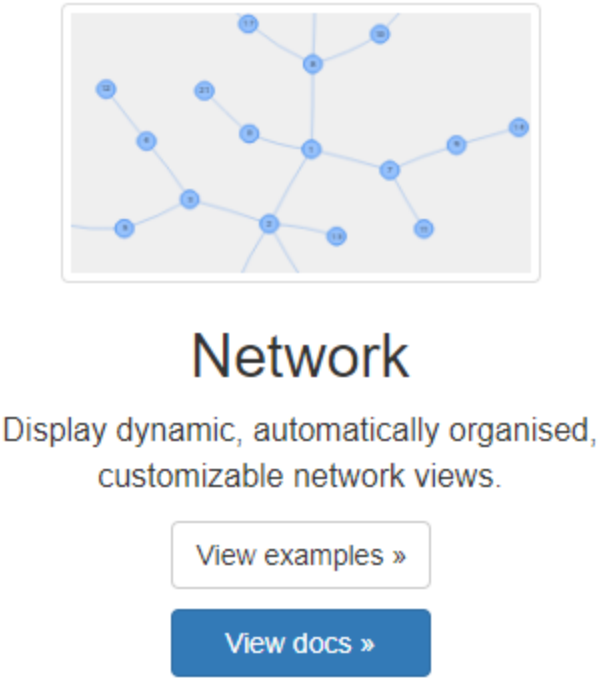

Figura 4.5. https://visjs.org/

4.2 Archivos de entrada

La entrada de datos para ejemplificar el cálculo del clique máximo se realiza generando archivos de texto utilizando la representación de lista de adyacencias.

4.2.1 Lista de adyacencias

La lista de adyacencia es una de las formas más comunes y eficientes de representar un grafo. En este tipo de representación, cada nodo del grafo se asocia con una lista que contiene todos los nodos con los que está conectado directamente. Esta lista refleja las adyacencias del nodo, es decir, sus vecinos inmediatos en el grafo.

Suponga que se tiene el grafo de la figura 4.6.

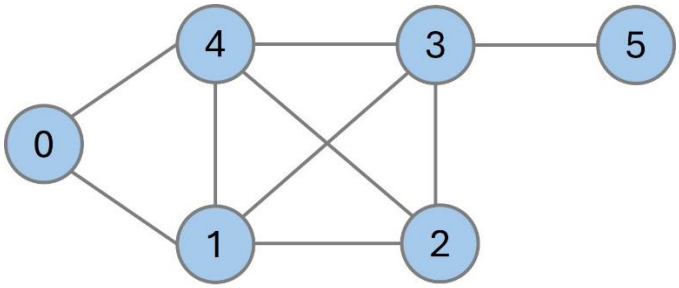

Figura 4.6. Grafo para ejemplificar la lista de adyacencias

La lista de adyacencias asociada es:

0 => [1, 4]

1 => [0, 2, 3, 4]

2 => [1, 3, 4]

3 => [1, 2, 4, 5]

4 => [0, 1, 2, 3]

5 => [3]

Esta lista de adyacencia se almacena en un archivo de texto para posteriormente ser procesada por la aplicación desarrollada.

4.3 Pantallas y pruebas del sistema

Las pruebas de funcionalidad de un sistema web consisten en verificar que todas las características, procesos y servicios del sitio operen conforme a los requisitos establecidos, garantizando que el usuario pueda interactuar correctamente con la aplicación.

Existen diferentes tipos de pruebas que generalmente abarcan aspectos como la validación de formularios, la correcta ejecución de operaciones sobre datos, la gestión de usuarios y roles, la navegación entre páginas y la respuesta adecuada del sistema ante entradas válidas o erróneas.

A continuación, se describe el funcionamiento del sistema a través de pruebas de navegación, las cuales se enfocan en comprobar que la estructura de menús, enlaces y rutas entre páginas funcionen correctamente, garantizando que el usuario pueda moverse por el

sitio de manera intuitiva y sin errores. El sistema incluye las siguientes funcionalidades:

- Pantalla principal
- Opción para calcular el clique máximo
- Generación de un grafo aleatorio
- Contacto

4.3.1 Pantalla principal del sistema

El sistema web desarrollado inicia después de activar el servidor web y con el llamado a través de un navegador web (http://localhost/clique).

Figura 4.7. Grafo para ejemplificar la lista de adyacencias

En la Figura 4.7 se muestra la pantalla principal del sistema la cual incluye una breve descripción del problema de Clique máximo, además, se muestra el menú principal que incluye las opciones:

- Inicio: pantalla principal

- Calcular clique: permite visualizar el clique máximo de un grafo introducido por un archivo de texto.
- Generar grafo: genera un grafo aleatorio especificando el número de nodos y aristas.
- Contacto: describe los datos de los autores del proyecto

4.3.2 Calcular clique

A continuación, vamos a describir la opción del sistema para “Calcular clique“, lo que permite determinar y visualizar el clique máximo de un grafo.

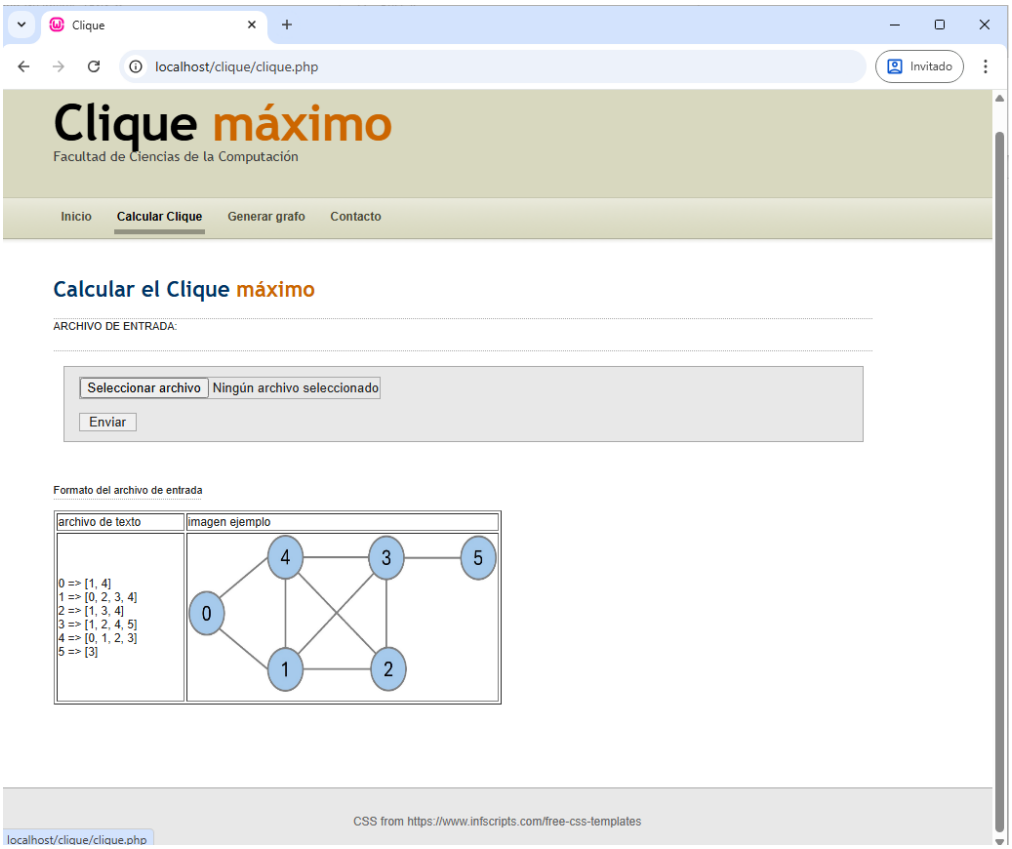

Figura 4.8. Grafo para ejemplificar la lista de adyacencias

En la pantalla mostrada de la figura 4.8 se indica el formato utilizado

en archivos txt del grafo, en el ejemplo siguiente:

----- entrada1.txt ----------

0 => [1, 4]

1 => [0, 2, 3, 4]

2 => [1, 3, 4]

3 => [1, 2, 4, 5]

4 => [0, 1, 2, 3]

5 => [3]

Aquí los datos del archivo entrada1.txt tiene 6 nodos etiquetados del 0 a1 5, donde cada nodo tiene asociada la lista de los nodos adyacentes, por ejemplo, el nodo 2 tiene como adyacentes los nodos 1, 3 y 4. Al dar clic en seleccionar archivo se abre una ventana emergente (ver Figura 4.9) que permite buscar el archivo de entrada correspondiente, recordando que debe ser un archivo txt con el formato especificado antes.

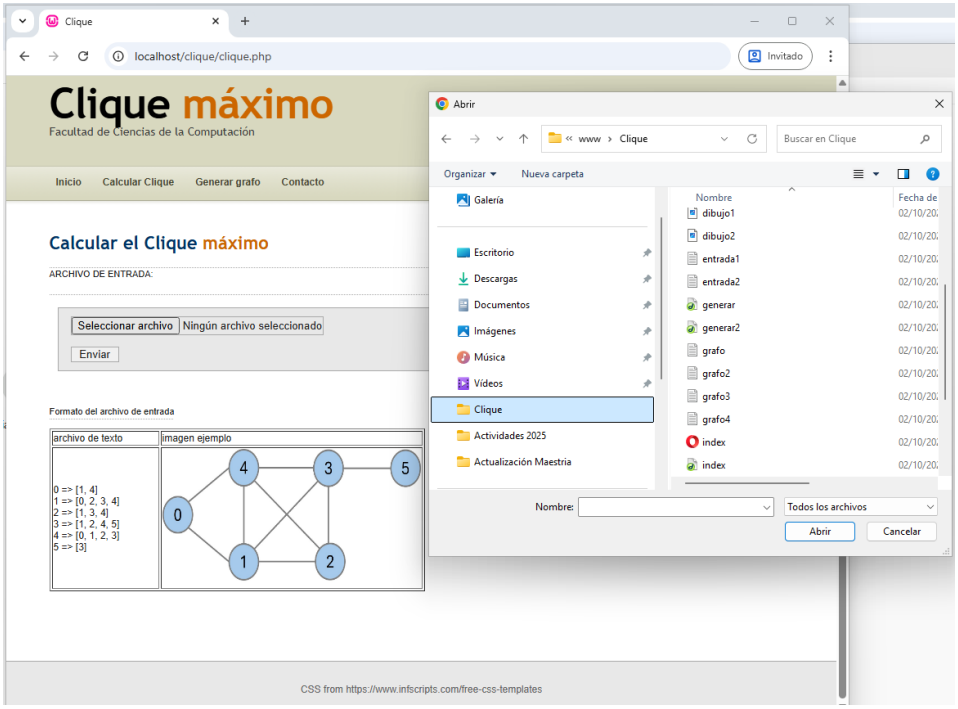

Figura 4.9. Seleccionar y cargar el archivo de entrada para calcular el clique máximo

Una vez seleccionado el archivo con el grafo de entrada entonces se carga el archivo como se muestra en la Figura 4.10 y ya podemos

dar clic en el botón “Enviar” para realizar el cálculo del clique máximo en el grafo especificado.

Figura 4.10. Archivo de entrada1.txt seleccionado

Posteriormente en la Figura 4.11 se muestra la ejecución del algoritmo Bron–Kerbosch para hallar el clique máximo. Aquí se indica el numero de nodos, aristas, el tamaño del clique y el conjunto de nodos que forman el clique máximo.

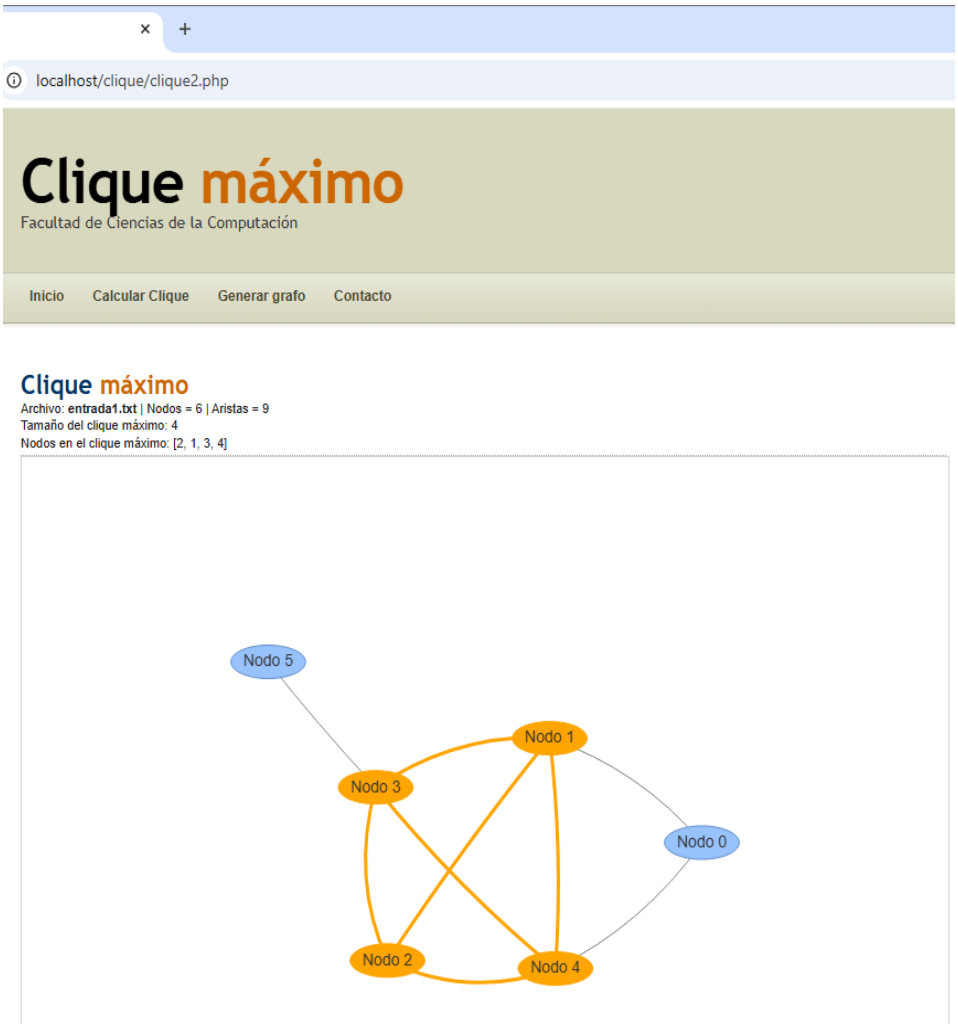

Figura 4.11. Prueba de ejecución del archivo de entrada1.txt para generar el clique máximo

El sistema permite ampliar y reducir el tamaño de la vista del grafo, por ejemplo, en la Figura 4.12 en la parte derecha se visualiza el grafo completo mientras que en la parte derecha se amplio el grafo para visualizar los nodos que forman el clique máximo.

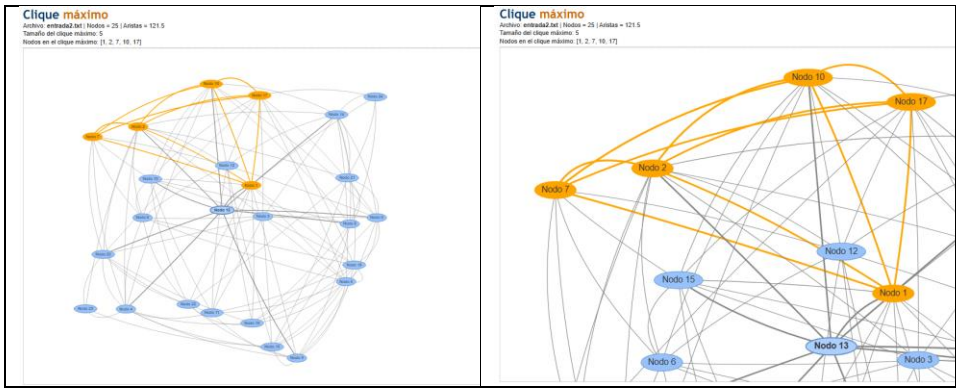

Figura 4.12. Prueba de ejecución del archivo de entrada1.txt para generar el clique

4.3.2 Generar gafo

Esta opción permite generar un grafo aleatorio, en la figura 4.13 se muestran los datos que solicita esta opción: Numero de Nodos, Numero de Aristas, ambos datos se espera que se introduzca un numero entero y posteriormente se da clic en el botón “Generar”.

Figura 4.13. Datos de entrada para generar un grafo aleatorio

Tome en cuenta que, si el número de aristas es significativamente menor que el número de nodos se puede dar el caso de generar un grafo no conexo, es decir con nodos aislados. Además, recuerde que el número máximo de aristas para *n* nodos debe ser (n*(n-1))/2.

En la Figura 4.14 mostramos en la parte izquierda un ejemplo de un grafo con pocas aristas (7 nodos y 6 aristas), mientras que en la parte derecha es un ejemplo de grafo completo (7 nodos y 21 aristas).

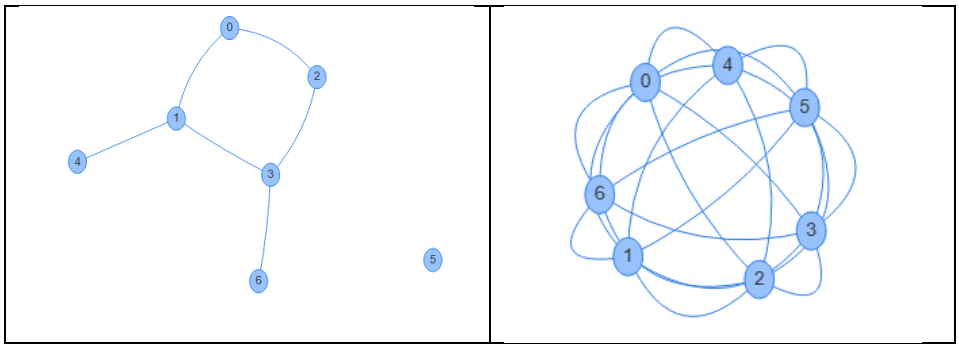

Figura 4.14. Ejemplo de grafos aleatorios, con pocas aristas y un grafo completo

Vamos a generar como ejemplo un grafo de 20 nodos y 100 aristas, en la Figura 4.15 observamos la ejecución del programa que genera el grafo aleatorio. Notemos que en la parte inferior se pide ingresar el nombre del archivo para almacenar el grafo generado si así lo considera el usuario.

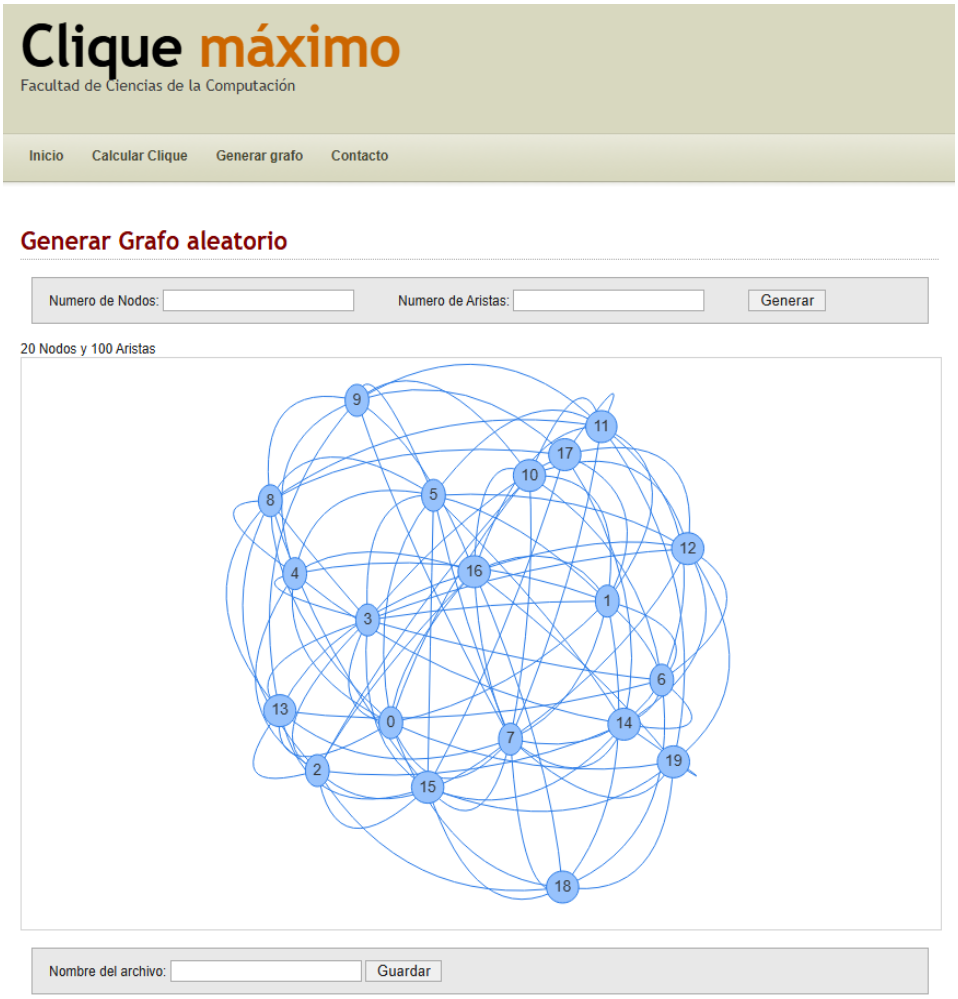

Figura 4.14. Datos de entrada para generar un

Si el usuario desea guardar el grafo generado aleatoriamente, debe dar el nombre del archivo y el sistema indica que el archivo se guardó de forma correcta como se muestra en la figura 4.15.

Figura 4.15. Almacenamiento del grafo aleatorio

El resultado de almacenar el grafo se indica en el archivo denominado grafo20-100.txt a continuación se describe la lista liga del grafo generado.

---

0 => [2, 18, 7, 11, 6, 13, 4, 9]
1 => [16, 17, 12, 6, 8, 7, 10, 18, 14, 3, 13, 9, 5, 2]
2 => [0, 8, 14, 13, 18, 4, 5, 9, 1, 6, 7]
3 => [5, 18, 15, 1, 10, 8]
4 => [14, 15, 10, 19, 7, 2, 9, 17, 18, 12, 0]
5 => [3, 15, 13, 17, 7, 12, 2, 11, 8, 1, 6]
6 => [15, 8, 11, 18, 1, 0, 17, 12, 2, 5]
7 => [10, 0, 5, 4, 1, 15, 14, 13, 19, 18, 11, 2]
8 => [19, 6, 2, 15, 14, 1, 12, 13, 5, 10, 3]
9 => [16, 11, 12, 1, 4, 19, 2, 15, 14, 0]
10 => [18, 7, 4, 1, 14, 3, 12, 8, 19]
11 => [13, 6, 17, 0, 9, 5, 7]
12 => [1, 13, 5, 8, 9, 17, 14, 15, 10, 6, 4]
13 => [11, 5, 17, 12, 18, 19, 2, 1, 8, 0, 15, 7]
14 => [4, 15, 2, 8, 1, 16, 10, 12, 7, 19, 9]
15 => [6, 5, 14, 4, 8, 18, 3, 19, 7, 12, 13, 9]
16 => [1, 18, 19, 9, 14]
17 => [1, 5, 13, 11, 12, 6, 4]
18 => [10, 0, 19, 6, 3, 16, 13, 2, 15, 1, 7, 4]
19 => [8, 18, 4, 16, 13, 15, 9, 7, 14, 10]

Si el usuario desea calcular el clique máximo de el grafo generado aleatoriamente tendrá que ir a la opción de Calcular Clique y seleccionar el archivo grafo20-100.txt, el resultado de este proceso se muestra en la Figura 4.16.

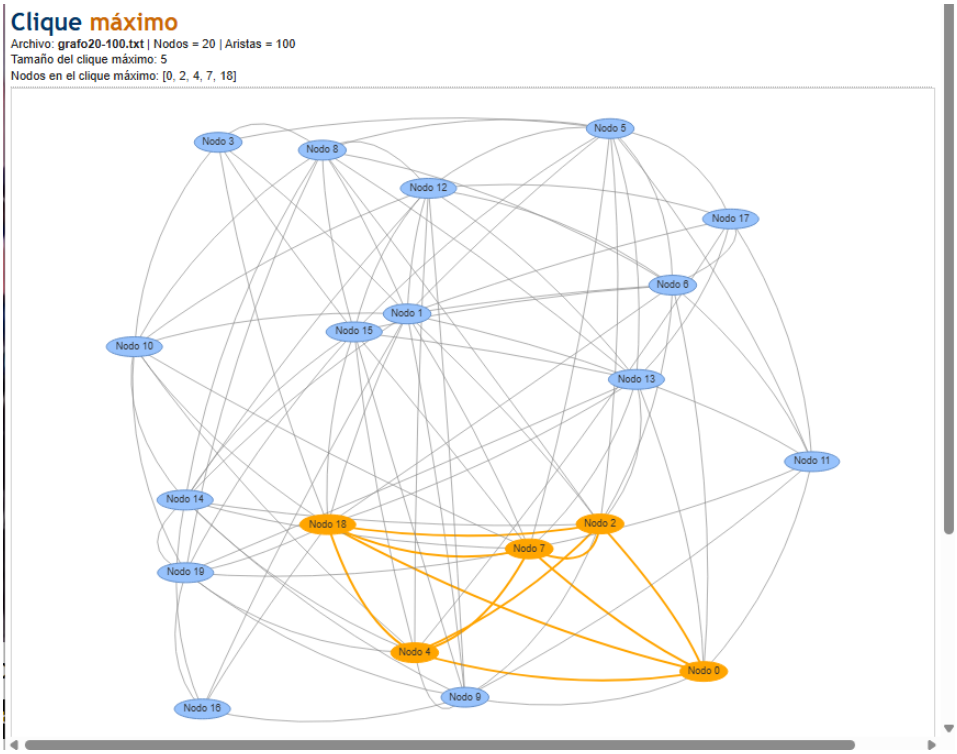

Figura 4.16. Clique máximo del archivo grafo20-100.txt

La última opción del sistema (Figura 4.17) es la sección de contacto donde se describe los autores de este trabajo.

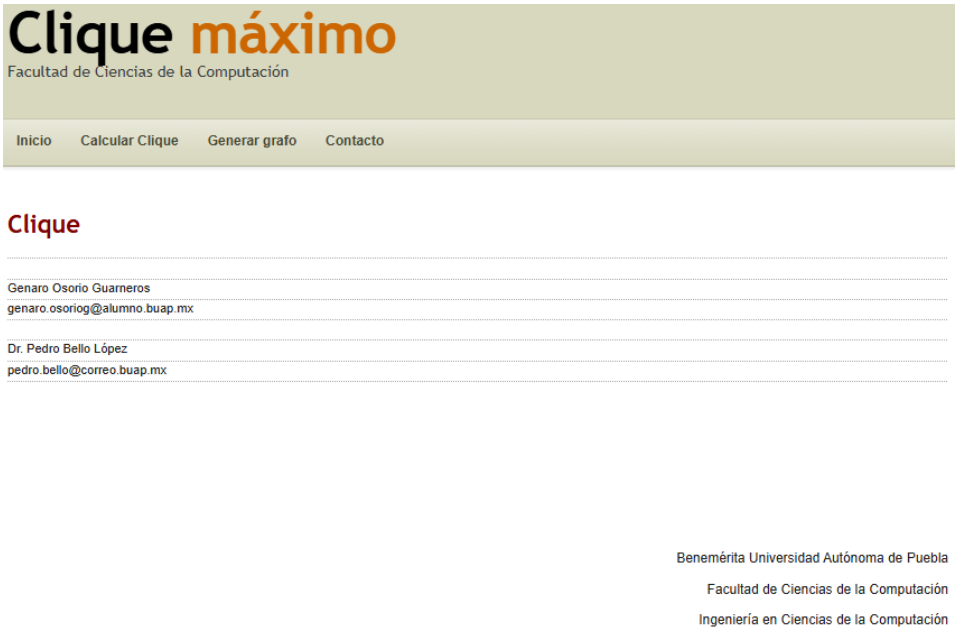

Figura 4.17. Datos de los autores del trabajo desarrollado

Conclusiones y perspectivas

En el presente trabajo se desarrolló una aplicación web interactiva e intuitiva que permite la experimentación con grafos de diferentes estructuras y la obtención del Clique Máximo, resaltando de manera gráfica los nodos que lo integran. Además, el sistema permite crear un repositorio con ejemplos y estancias de prueba aleatorias que fortalecen la aplicabilidad y validación de la herramienta web.

El proyecto evidenció un impacto en dos dimensiones principales. En el ámbito educativo, ofrece un recurso didáctico valioso para cursos de algoritmos, teoría de grafos y optimización, fomentando la comprensión visual y el desarrollo del pensamiento computacional. En el ámbito tecnológico y científico, la aplicación constituye un prototipo inicial aplicable a investigaciones en análisis de comunidades en redes sociales, detección de fraudes y estudio de clusters genéticos.

En general el trabajo contribuye al fortalecimiento de las ciencias básicas, especialmente la matemática discreta y la computación teórica, al proporcionar un medio práctico para experimentar con conceptos fundamentales de la teoría de grafos. La implementación y visualización del algoritmo de detección del Clique Máximo facilita la comprensión de estructuras combinatorias complejas, promoviendo el aprendizaje significativo, el razonamiento lógico y la consolidación de competencias en modelado computacional.

Perspectivas de Trabajo Futuro

Asimismo, se plantea como línea de mejora la optimización de los algoritmos implementados, con el fin de manejar grafos de mayor tamaño y complejidad, incrementando la eficiencia computacional. De igual manera, resulta pertinente incorporar funcionalidades adicionales, como la integración con librerías de análisis de datos, la exportación de resultados en distintos formatos y la posibilidad de ejecutar pruebas comparativas entre diferentes algoritmos de detección de cliques.

Finalmente, una perspectiva relevante consiste en la vinculación de la aplicación con proyectos interdisciplinarios en áreas como ciencias sociales, biología computacional o ciberseguridad, en los que el estudio de comunidades y la identificación de cliques representan una herramienta de análisis fundamental.

Bibliografía

[1] Bron, C., & Kerbosch, J. (1973). Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM, 16(9), 575-577.

[2] Karp, R. M. (1972). Reducibility among combinatorial problems. In Complexity of computer computations (pp. 85-103). Springer.

[3] Feo, T. A., & Resende, M. G. C. (1994). A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters, 8(2).

[4] West, D. B. (2001). Introduction to Graph Theory. Prentice Hall.

[5] Östergård, P. R. J. (2002). A fast algorithm for the maximum clique problem. Discrete Applied Mathematics, 120(1-3), 197-207.

[6] Ismael Gutierrez García, Yesneri Maider Zuleta Saidarriaga (2024). Introducción a la teoría de grafos, conceptos, algoritmos y aplicaciones. Editorial uninorte, Colombia.

[7] Jiri Matousek, Jaroslav Nesetril. (2008). Invitación a la matemática discrete. Editorial Reverté.

[8] Oscar Meza, Maruja Ortega. (2006). Grafos y Algoritmos. Editorial Equinoccio, Universidad Simon Bolivar.

[9] Adan Dorzdek, (2007). Estructuras de datos y Algoritmos con Java, International Thomson Editores.

[10] Rudich, S., Wigderson, A. (2000). Computational Complexity Theory. Vol. 10. American Mathematical Society

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

[12] Kleinberg, J., & Tardos, É. (2006). Algorithm Design. Pearson Education.

[13] Abraham Gutiérrez, Ginés Bravo. (2005). PHP5, Alfaomega Ra-MA, México.

[14] Luke Welling, Laura Thomson. (2009). Desarrollo Web con PHP y MySQL Editorial Anaya.

[15] Braude (2003), Ingeniería de Software, una perspectiva orientada a objetos, Alfaomega, México.

[16] Ian Sommerville (2005). Ingeniería del software, Séptima edición, Pearson Educación. S.A. Madrid.