Instituto Tecnológico de Costa Rica

Unidad de Computación

Proyecto #1

Análisis de Algoritmos

Profesora:

Ana Lorena Valerio Solís

Estudiantes:

Roosevelt Alejandro Pérez González

José Andrés Lorenzo Segura

Arnold Jafeth Alvares Rojas

Campus San Carlos

I Semestre 2024

Índice de contendidos

Índice de figuras

Introducción

Uno de los problemas más trascendentales en la teoría de grafos es el de encontrar el clique máximo de un grafo. Este problema, que, a primera vista podría parecer simple posee una gran complejidad computacional. Este problema ha sido objeto de estudio debido a su aplicación en una amplia gama de disciplinas, que van desde la informática hasta la biología molecular.

El clique máximo de un grafo se refiere a un subconjunto de vértices (nodos) de un grafo que están todos conectados entre sí, la conexión entre todos estos grafos se realiza mediante una arista.

Este proyecto se enfocará en implementar dos posibles algoritmos para afrontar los desafíos que presenta este problema. Los algoritmos por implementar son el de Bron - Kerbosch y el de Búsqueda Local. Estos algoritmos representan dos estrategias distintas para encontrar la solución de este problema. El algoritmo de Bron – Kerbosch se basa en la técnica de backtracking, explorando exhaustivamente todas las posibles combinaciones de vértices para determinar el clique máximo.

Por otro lado, el de búsqueda local se centra en mejorar iterativamente soluciones existentes mediante el movimiento a través del espacio de soluciones vecinas.

A lo largo de este proyecto se explorará en detalle la implementación y el rendimiento de estos dos enfoques algorítmicos en la resolución del problema del clique máximo.

Se analizará sus complejidades computacionales, se pondrán a prueba con una variedad de grafos, y se compararán entre estos mismos.

Al final de la investigación se espera obtener información valiosa sobre sus aplicaciones practicas y su eficacia en diferentes escenarios.

Análisis del problema

Según [1] el objetivo del problema del clique máximo es encontrar un clique máximo en un grafo no dirigido arbitrariamente. Se menciona que este problema es computacionalmente equivalente a otros problemas importantes en teoría de grafos, como el problema del conjunto independiente máximo (o conjunto estable) y el problema de la cobertura mínima de vértices. Dado que estos son problemas NP-duros, no se esperan algoritmos polinomiales para resolverlos. Sin embargo, debido a que estos problemas tienen varias

aplicaciones prácticas relevantes, existe un gran interés en desarrollar algoritmos exactos rápidos para instancias pequeñas.

Según [2] y [3] este problema se clasifica dentro de los problemas de NP-duros, los cuales son muy complicados de resolver. Debido a esta complejidad, es necesario desarrollar algoritmos heurísticos y/o metaheurísticos para lograr una solución cercana al óptimo a un tiempo razonable. Según [2] este problema tiene aplicaciones reales como lo son: La teoría de códigos, diagnósticos de errores, visión computacional, análisis de agrupamiento, recuperación de información, aprendizaje automático, entre otros.

Durante la fase inicial del proyecto, es decir, la investigación de los posibles algoritmos a implementar, se han encontrado diferentes opciones, las cuales se van a enumerar a continuación.

- Algoritmo: Búsqueda local
- Algoritmos Genéticos
- Algoritmo: Bron - Kerbosch
- Algoritmo: Búsqueda tabú
- Algoritmo: Optimización por colonias de hormigas
- Algoritmo: Ramificación y Acota

Luego de la valoración de todos los posibles algoritmos a implementar, se seleccionaron los algoritmos de Bron - Kerbosch y Búsqueda Local.

A través de la revisión de los datos obtenidos en [4] y en [5] se puede hacer una comparativa entre el algoritmo de búsqueda local, el cual fue seleccionado para el proyecto, con respecto al algoritmo genético, esto debido a que en ambos documentos los algoritmos realizan una serie de pruebas brindadas por Center for Discrete Mathematics and Theorical Computer Science (DINAMICS). En el cual se toma el tamaño del clique a través de instancias las cuales brinda el DINAMICS con su respectivo valor a aproximarse, viéndose reflejado en el cuadro del algoritmo de búsqueda local el cómo llega a la solución idónea en la mayoría de casos, siendo la prueba de brock en la única que se logró destacar un poco mejor el algoritmo genético con respecto al de búsqueda local.

A su vez en [6] se aprecia un análisis del algoritmo de colonia de hormigas con el cual se realizan las mismas pruebas y con el cual se denota que llega a ser muy semejante a la

eficiencia del algoritmo genético, sin embargo, este mismo llega a superarlo en la mayoría de pruebas acercándose más al valor exacto del clique máximo que se espera, así mismo en [7] se menciona que el comportamiento y la naturaleza de la búsqueda tabú es muy semejante al algoritmo genético y al de colonia, sin embargo, no se hace mención de cuan eficiente es exactamente siguiendo lo estipulado por las pruebas hechas a los algoritmos ya mencionados.

Con respecto a la estructura de los códigos utilizados, tanto el código de Bron como el de búsqueda local utilizan una estructura de grafo, sin embargo, estas se diferencian en cómo se compone su estructura de datos siendo que en el grafo de Bron se utiliza una lista de adyacencia, que es una colección de conjuntos donde cada conjunto representa los vértices adyacentes a un vértice en particular y en el grafo de búsqueda local se utiliza una matriz de adyacencia, que es una matriz booleana donde cada elemento indica si hay una conexión entre dos vértices.

En el caso del acceso a la información de conexión, el grafo de Bron, para saber si hay una conexión entre dos vértices, consulta el conjunto de adyacencia del vértice correspondiente y se verifica si el otro vértice está presente en ese conjunto, por otro lado el grafo de búsqueda local, para saber si hay una conexión entre dos vértices, consulta la matriz de adyacencia y se verifica el valor booleano correspondiente a la posición de los vértices en la matriz.

Por último, la lista de adyacencia del grafo de Bron es más eficiente en términos de espacio cuando el grafo es disperso, es decir, cuando la mayoría de los vértices tienen pocos vecinos y la matriz de adyacencia del grafo de búsqueda local es más eficiente en términos de espacio cuando el grafo es denso, es decir, cuando la mayoría de los vértices están conectados entre sí.

A continuación, se presentarán cuatro diagramas de grafos introducidos para probar los algoritmos. Los nodos en color verde son los que forman él clique máximo del grafo.

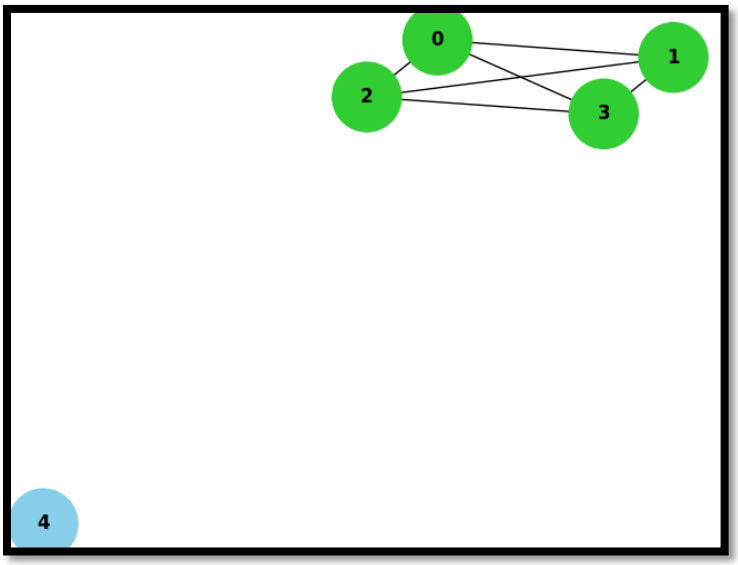

Figura 1: Primer grafo introducido en el sistema

En esta primera prueba de los algoritmos implementados, se ha introducido un grafo de cinco vértices con seis aristas. Las seis aristas se han utilizado para formar un clique de 4 vértices. De hecho, si nos basamos en la siguiente formula

$$\frac{x * (x-1)}{2} = \text{aristas del clique de x vertices}$$

podemos apreciar que en ese clique se utilizaron todas las posibles aristas. El sistema deberá de imprimir que el clique máximo de esta prueba está compuesto por los vértices [0,1,2,3].

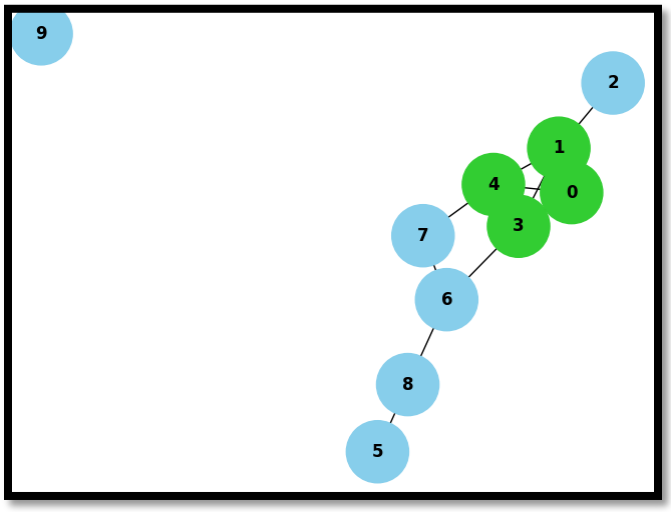

Figura 2: Segundo grafo introducido en el sistema

La segunda prueba introducida en el sistema se trata de un grafo con 10 vértices con 12 aristas. Nuevamente se ha elegido un clique de 4 vértices, el cual, ya sabemos que tiene seis aristas. Las otras seis fueron elegidas están en las conexiones [(2,1), (7,4), (6,3), (6,8), (5,8)]. El clique esperado de esta prueba está formado por los vértices [0,1,3,4].

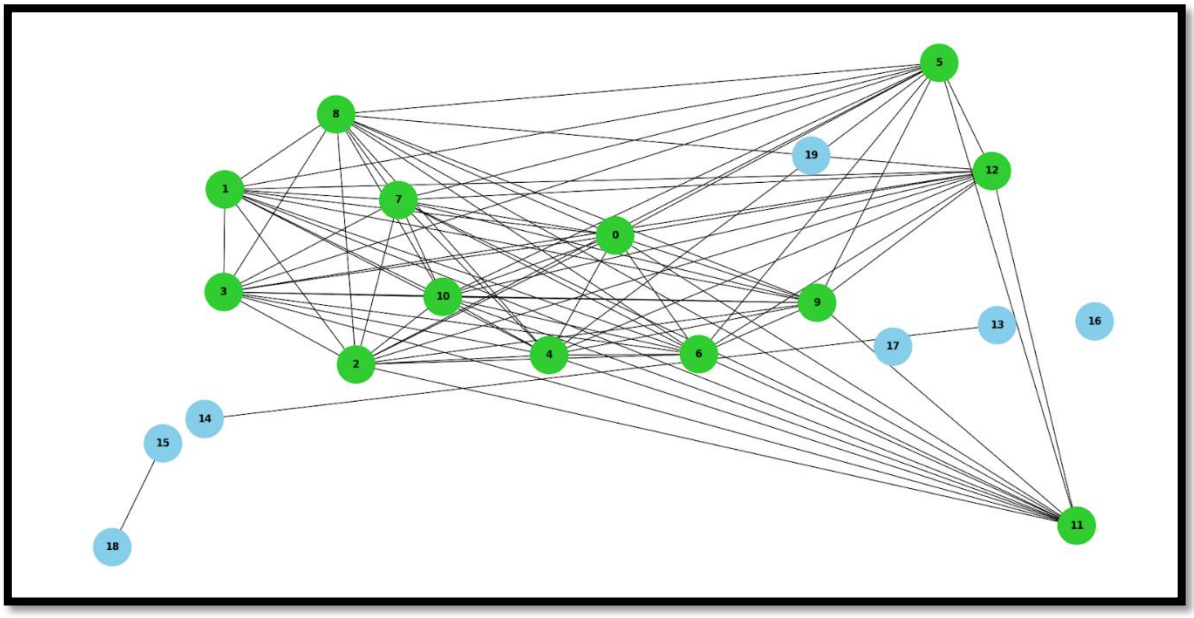

Figura 3: Séptimo grafo introducido en el sistema

En la séptima prueba podemos apreciar un grafo más grande que los otros dos vistos con anterioridad. En esta prueba se ha introducido un grafo de 20 vértices con 80 aristas. En esta prueba se buscó realizar un clique con el máximo número de aristas posibles. Basándonos en la formula presentada con anterioridad, podemos notar que el clique máximo a realizar en este grafo cuenta con 78 aristas, que es el resultado de hacer un clique de 13 vértices.

$$\frac{13 * (12)}{2} = 78$$

Por ende, se realizó la formación de ese clique que va desde el vértice 0 hasta el 12. El clique esperado es [0,1,2,3,4,5,6,7,8,9,10,11,12] y las dos conexiones restantes son [(13, 14), (18, 15)].

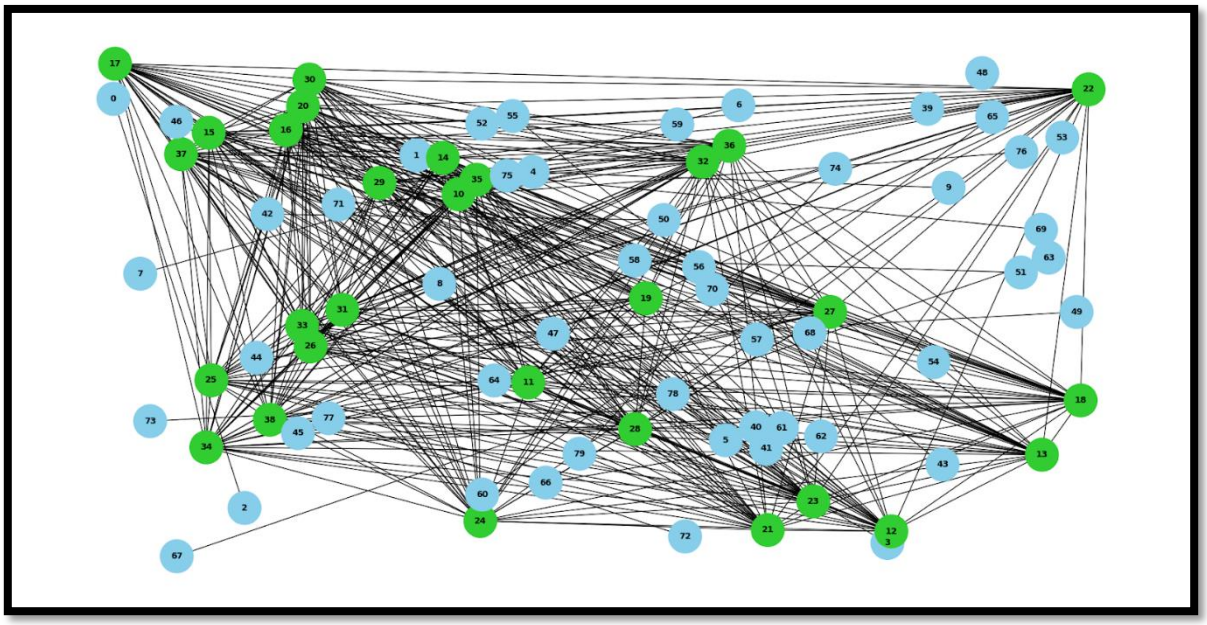

Figura 4: Décimo grafo introducido en el sistema

La ultima prueba realizada en los algoritmos desarrollados es un grafo que cuenta con 80 vértices y 400 aristas. En este grafo se realizó un clique de 28 vértices, es decir, se usaron para definir este clique 378 aristas.

$$\frac{28 * 27}{2} = 378$$

El clique que se espera de grafo es [10,11,12…37,38].

Las conexiones restantes que son 22, son las siguientes.

[(0, 2), (3, 8), (4, 9), (6, 7), (1, 5), (40, 45), (41, 46), (42, 50), (43, 55), (44, 49), (51, 58), (52, 59), (60, 65), (61, 70), (62, 66), (63, 67), (64, 68), (69, 75), (71, 76), (72, 77), (73, 78), (74, 79)].

Solución del problema

En cuanto a la lógica detrás del código de Bron - Kerbosch se observan los siguientes puntos claves:

Inicialización: Se inicializan tres conjuntos vacíos: clique, candidates y excluded. El conjunto candidates contiene todos los vértices del grafo.

Exploración recursiva: Se explora recursivamente el espacio de soluciones. En cada llamada recursiva, se elige un vértice de los candidatos, se agrega al conjunto clique y se actualiza el conjunto candidates para contener sólo los vértices adyacentes al vértice elegido que no están en el conjunto excluded. Luego, se llama recursivamente al algoritmo con estos conjuntos actualizados.

Caso base: La recursión termina cuando no quedan vértices en el conjunto candidates y excluded. En este punto, se verifica si el conjunto clique es un clique válido. Si lo es y es más grande que el clique máximo encontrado hasta el momento, se actualiza el clique máximo.

Intersección de conjuntos: La función intersection se utiliza para calcular la intersección entre dos conjuntos.

Las mejoras que se implementan conforme se ajustaba el funcionamiento del código fueron:

Optimización del bucle de inicialización: En lugar de iterar sobre todos los vértices del grafo para agregarlos al conjunto candidates, se utiliza un bucle for simple que agrega los vértices al conjunto candidates durante la inicialización del algoritmo.

Optimización del manejo de conjuntos: Se utilizan conjuntos (HashSet) para almacenar los vértices y sus adyacencias, lo que permite una búsqueda eficiente y la eliminación rápida de vértices durante el proceso de backtracking.

Mejora en el cálculo de intersección: La función intersection se implementa de manera eficiente utilizando el método retainAll de la clase HashSet, lo que reduce el tiempo de cálculo de la intersección entre conjuntos.

Para el caso de la lógica utilizada en búsqueda local se tratan los siguientes puntos claves:

Inicialización: Se inicializa un conjunto currentClique con un vértice aleatorio como punto de partida. También se inicializa un conjunto bestClique para mantener el mejor clique encontrado hasta el momento.

Búsqueda Local: Se realiza una búsqueda local iterativa, donde en cada iteración se intenta mejorar la clique actual agregando o eliminando un vértice. Se repite este proceso hasta que no se puedan realizar más mejoras.

Agregar vértices: En cada iteración, se verifica si agregar un vértice no presente en la clique actual resultaría en un clique válido. Si es así y la nueva clique es más grande que la mejor encontrada hasta el momento, se actualiza la mejor clique.

Eliminar vértices: Si no se pueden agregar más vértices a la clique actual, se intenta eliminar un vértice del clique actual para ver si esto mejora la solución. Se repite este proceso hasta que ya no se puedan realizar más mejoras.

Criterio de parada: El algoritmo termina cuando no se pueden realizar más mejoras en la clique actual.

Las mejoras que se implementan conforme se ajustaba el funcionamiento del código fueron:

Selección de vértice aleatorio: Se selecciona un vértice aleatorio como punto de partida en lugar de utilizar un vértice fijo. Esto puede ayudar a explorar diferentes regiones del espacio de soluciones.

Optimización de la verificación de cliques: Se utiliza un método eficiente para verificar si un conjunto de vértices forma un clique, lo que reduce el tiempo de cálculo y mejora la eficiencia del algoritmo.

Exploración local eficiente: El algoritmo realiza exploración local eficiente, intentando agregar y eliminar vértices de la clique actual de manera inteligente para mejorar la solución.

Análisis de resultados

Al realizar la toma de datos no se presentó ningún inconveniente, se pudieron obtener 2 algoritmos para realizar la comparación entre los mismos y esto a su vez se desarrolló con pequeños inconvenientes iniciales al ejecutar el código que fueron corregidos logrando la toma de los datos que se solicitaban.

A continuación, se mostrarán las tablas correspondientes a la medición empírica, donde se encuentran los datos de los tiempos de ejecución, cantidad de comparaciones, cantidad de asignaciones, cantidad de líneas de código y el total de líneas ejecutadas en cada prueba de ambos algoritmos.

Nótese que cuando los vértices son iguales, la variante son la cantidad de aristas que tiene cada grafo.

Medición Empírica

Vertices	Bron-K				
	Asignaciones	Comparaciones	Cantidad de lineas ejecutadas	Tiempo de ejecucion	Cantidad de lineas de codigo
5	279	57	336	0 ms	25
10	464	101	565	0 ms	
20	1206	254	1460	0 ms	
20	2439	484	2923	1 ms	
20	1731	357	2088	1 ms	
20	4232	819	5051	1 ms	
20	131264	24628	155892	37 ms	
40	525276	98533	623809	78 ms	
60	67109762	12853142	79962904	3329 ms	
80	8589935965	1610613064	10200549029	442645 ms	

Tabla 1: Medición empírica del algoritmo Bron - Kersboch

Vertices	Busqueda local				
	Asignaciones	Comparaciones	Cantidad de lineas ejecutadas	Tiempo de ejecucion	Cantidad de lineas de codigo
5	69	97	166	36 ms	20
10	234	325	559	0 ms	
20	864	1263	2127	0 ms	
20	864	1357	2221	1 ms	
20	865	1321	2186	1 ms	
20	864	1858	2722	0 ms	
20	864	2201	3065	0 ms	
40	3324	6413	9737	1 ms	
60	7386	15847	23233	4 ms	
80	13045	30943	43988	7 ms	

Tabla 2: Medición empírica del algoritmo de Búsqueda Local

Como se observa en las Tablas 1 y 2, el algoritmo de búsqueda local (BL) muestra un rendimiento superior en términos de asignaciones comparado con el algoritmo de Bron-Kerbosch (BK). Específicamente, en grafos de apenas cinco vértices, el algoritmo BK realiza casi cuatro veces más asignaciones que el algoritmo BL. Además, en cuanto al número de líneas ejecutadas, el algoritmo de BK muestra una frecuencia 2,02 veces

mayor que la del algoritmo BL. Y la superioridad del algoritmo BL sobre el de BK se repite en todos los grafos con algunas excepciones.

Estas diferencias son particularmente notables en el análisis de las comparaciones efectuadas desde grafos de 5 vértices hasta grafos de 20 vértices con 80 aristas (el último grafo de 20 vértices analizado), donde el algoritmo de Bron-Kerbosch (BK) demuestra superioridad. No obstante, a partir de este punto, específicamente en grafos de más de 20 vértices con 80 aristas, se observa un deterioro notable en el rendimiento de dicho algoritmo en este aspecto.

Sin embargo, es importante destacar la superioridad general del algoritmo de búsqueda local (BL) sobre el de Bron-Kerbosch (BK), especialmente evidente en el grafo de mayor tamaño evaluado, que cuenta con 80 vértices y 400 aristas. En este caso, el algoritmo BK realiza aproximadamente 658,484.93 veces más asignaciones que el algoritmo BL. Además, la diferencia en el tiempo de ejecución es significativa, con una ventaja de hasta siete minutos a favor del algoritmo de búsqueda local.

Tamaño del grafo en vertices	Bron-K				
	Factor talla	Factor Asig	Factor comparaciones	Factor cantidad de líneas ejecutadas	Factor Tiempo de ejecucion
10/5	2	1,663082437	1,771929825	1,681547619	0
20/10 (40 Arcos)	2	3,730603448	3,534653465	3,695575221	0
40/20	2	303,4523397	1,905511811	2,002054795	78
60/20	3	38769,35991	36003,19888	38296,40996	42,67948718
80/40	2	16353,18569	16345,92537	16352,03889	5674,935897
80/20	4	4962412,458	4511521,188	4885320,416	442645
80/10	8	18512793,03	15946664	18054069,08	0
80/5	16	30788300,95	28256369,54	30358776,87	0

Tabla 3: Factor de crecimiento del algoritmo Bron – Kerbosch

Se puede apreciar en la tabla del factor de crecimiento

Tamaño del grafo en arcos usando solo las pruebas de 20 vértices	Factor talla	Factor Asig	Factor comparaciones	Factor cantidad de lineas ejecutadas	Factor Tiempo de ejecución
30/20	1,5	2,02238806	1,905511811	2,002054795	0
40/20	2	1,435323383	1,405511811	1,430136986	0
50/20	2,5	3,509121061	3,224409449	19,88582677	0
80/20	4	108,8424544	96,96062992	106,7753425	0
80/50	1,6	31,01701323	30,07081807	30,86359137	37

Tabla 4: Factor de crecimiento del algoritmo Bron - Kerbosch usando solo los grafos de 20 vértices

Tamaño del grafo en vertices	Busqueda Local				
	Factor talla	Factor Asig	Factor comparaciones	Factor cantidad de líneas ejecutadas	Factor Tiempo de ejecucion
10/5	2	3,391304348	3,350515464	3,36746988	0
20/10 (40 Arcos)	2	3,692307692	4,064615385	3,910554562	0
40/20	2	3,842774566	4,854655564	4,454254346	1
60/20	3	8,538728324	11,99621499	10,62808783	4
80/40	2	3,924488568	4,825042882	4,517613228	7
80/20	4	15,08092486	23,42392127	20,12259835	7
80/10	8	55,74786325	95,20923077	78,69051878	0
80/5	16	189,057971	319	264,9879518	0,1944

Tabla 5: Factor de crecimiento del algoritmo Búsqueda Local

La tabla de factores de crecimiento del algoritmo de Búsqueda Local (BL) destaca uno de los hallazgos más significativos de esta investigación: la complejidad algorítmica cuadrática del algoritmo, es decir, O(n^2). Este resultado es fundamental, ya que indica que estamos frente a un algoritmo con un tiempo polinomial aceptable. En comparación con otros algoritmos de complejidades exponenciales o factoriales, el algoritmo BL ofrece una solución en un tiempo considerablemente menor.

Es importante tener en cuenta que el enfoque de este algoritmo es heurístico, lo que significa que su principal objetivo es proporcionar una respuesta en un tiempo aceptable, como lo está demostrando. Sin embargo, es esencial recalcar que, debido a su naturaleza heurística, el algoritmo BL podría no garantizar la respuesta correcta en todo momento. Existe la posibilidad de que se estanque en un máximo local, que no necesariamente coincide con el máximo global del problema.

Tamaño del grafo en arcos usando solo las pruebas de 20 vértices	Factor talla	Factor Asig	Factor comparaciones	Factor cantidad de líneas ejecutadas	Factor Tiempo de ejecución
30/20	1,5	1	1,07442597	1,0441937	0
40/20	2	1,001157407	1,045922407	1,027738599	0
50/20	2,5	1	1,471100554	1,279736718	0
80/20	4	1	1,742676168	1,440996709	0
80/50	1,6	1	1,184607104	1,126010287	0

Tabla 6: Factor de crecimiento del algoritmo Búsqueda Local usando solo los grafos de 20 vértices

A continuación, se mostrarán las notaciones para cada comportamiento de los algoritmos.

Notación Bron - Kerbosch

Clasificación en notación O Grande según sus comparaciones, asignaciones, líneas ejecutadas y tiempo de ejecución. Contemplando el tamaño de los vértices del grafo:

Usar la notación O	
Clasificación del comportamiento de las **asignaciones**	
Clasificación del comportamiento de las **comparaciones**	
Clasificación del comportamiento de las **líneas ejecutadas**	
Clasificación del comportamiento en el **tiempo de ejecución**	

Clasificación en notación O Grande según sus comparaciones, asignaciones, líneas ejecutadas y tiempo de ejecución. Contemplando el tamaño fijo de vértices del grafo cambiando la cantidad de arcos:

Usar la notación O	
Clasificación del comportamiento de las **asignaciones**	
Clasificación del comportamiento de las **comparaciones**	
Clasificación del comportamiento de las **líneas ejecutadas**	
Clasificación del comportamiento en el **tiempo de ejecución**	

Notación Búsqueda Local

Clasificación en notación O Grande según sus comparaciones, asignaciones, líneas ejecutadas y tiempo de ejecución. Contemplando el tamaño de los vértices del grafo:

Usar la notación O	$O(n^2)$
Clasificación del comportamiento de las **asignaciones**	$O(n^2)$
Clasificación del comportamiento de las **comparaciones**	$O(n^2)$
Clasificación del comportamiento de las **líneas ejecutadas**	$O(n^2)$
Clasificación del comportamiento en el **tiempo de ejecución**	O(1)

Clasificación en notación O Grande según sus comparaciones, asignaciones, líneas ejecutadas y tiempo de ejecución. Contemplando el tamaño fijo de vértices del grafo cambiando la cantidad de arcos:

Usar la notación O	
Clasificación del comportamiento de las **asignaciones**	
Clasificación del comportamiento de las **comparaciones**	
Clasificación del comportamiento de las **líneas ejecutadas**	
Clasificación del comportamiento en el **tiempo de ejecución**	

A continuación, se mostrarán las gráficas correspondientes al proyecto.

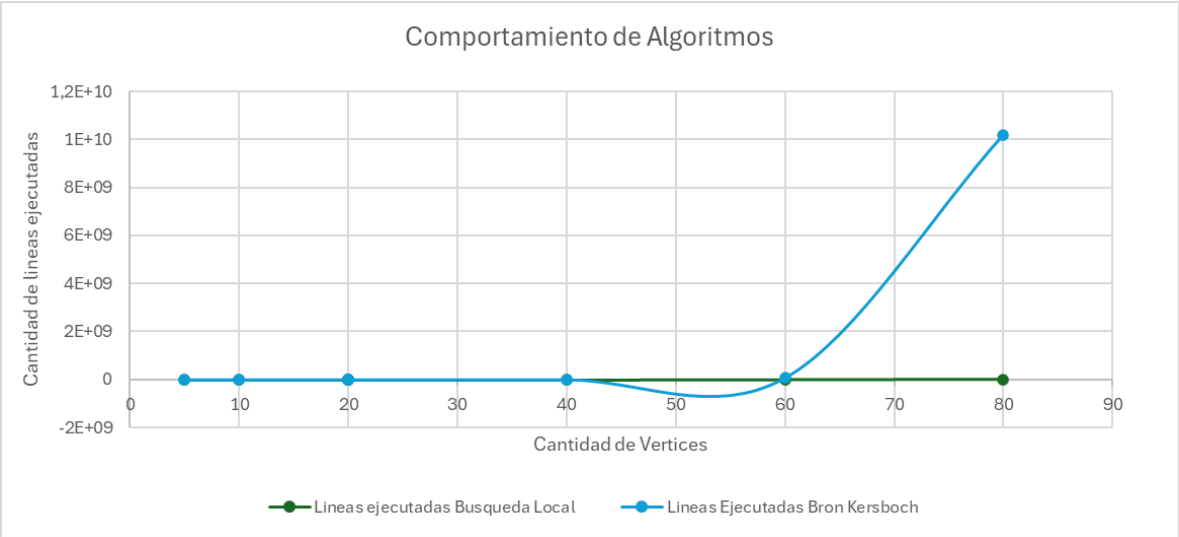

Tabla 7: Comportamiento de los Algoritmos

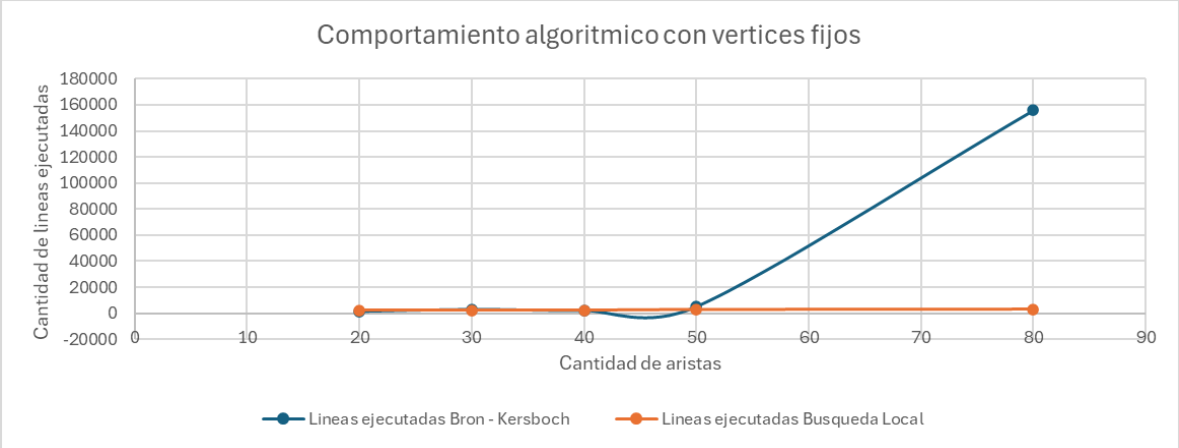

Tabla 8: Comportamiento de los Algoritmos con 20 vértices y aristas variantes.

Al revisar los datos obtenidos se observaron los siguientes aspectos entre ambos algoritmos:

El algoritmo de Bron-Kerbosch, en contraste con la Búsqueda Local, tiene una complejidad temporal exponencial en el peor de los casos. Esto significa que para grafos densos o con muchos cliques, el rendimiento puede deteriorarse significativamente. Bron-Kerbosch realiza una exploración exhaustiva de las posibles combinaciones de vértices para encontrar cliques máximos siendo en el peor de los casos un comportamiento de O(3^(N/3)), esto según la literatura, lo que puede resultar en un alto costo computacional incluso para grafos relativamente pequeños. Aunque Bron-Kerbosch puede encontrar el clique máximo de un grafo, puede volverse ineficiente para grafos grandes debido a su naturaleza exhaustiva y su enfoque poco selectivo.

Por otro lado, la Búsqueda Local tiene una complejidad temporal que depende del número de iteraciones y la cantidad de operaciones realizadas en cada iteración. En comparación con Bron-Kerbosch, la complejidad puede ser menor en muchos casos. La Búsqueda Local se enfoca en buscar soluciones locales óptimas, explorando solo un subconjunto de todas las posibles combinaciones de vértices. Este enfoque más selectivo y la evitación de explorar todas las posibles combinaciones de vértices hacen que la Búsqueda Local sea más adecuada para grafos grandes, donde puede ser más eficiente en términos de tiempo de ejecución y recursos computacionales.

Esto se puede apreciar mejor en [5] donde se muestra que el comportamiento de la búsqueda local al aplicar las pruebas de DINAMICS, logra obtener los resultados esperados, logrando ser más eficiente que el Bron el cual según [8], hace mención del comportamiento inevitable que llega a poseer el Bron en entradas grandes, siendo su comportamiento exponencial.

El aumentar la cantidad de arcos sin aumentar la cantidad de vértices no provoca un cambio muy grande, generando solo que el cambio de donde se produce la exponencial del bron pase de ser en la cantidad de 60 aristas a la cantidad de 50, para el caso de la búsqueda local esta sigue un crecimiento muy semejante, siendo un poco más elevado a comparación que para la prueba sin limitar la cantidad de vértices.

Medición analítica

Bron - Kersboch

Código fuente Solo se analiza el código del algoritmo.	Medición de líneas ejecutadas en el peor de los casos (línea por línea)

public class BronKerboschMaxClique {
  public static Set<Integer> bronKerbosch(GraphKerbosch graph) {;
  Set<Integer> clique = new HashSet<>();
  Set<Integer> candidates = new HashSet<>();
  Set<Integer> excluded = new HashSet<>();
  for (int v = 0; v < graph.getVertexCount(); v++) {
  candidates.add(v);
}
  bronKerbosch(graph, clique, candidates, excluded);
  return clique;
  }
  private static void bronKerbosch(GraphKerbosch graph, Set<Integer> clique, Set<Integer> candidates, Set<Integer> excluded) {
  if (candidates.isEmpty() && excluded.isEmpty()) {
  if (clique.size() > maxCliqueKersboch.size()) {
  maxCliqueKersboch = new HashSet<>(clique);
  }
  return;
  }
  List<Integer> candidatesList = new ArrayList<>(candidates);
  for (Integer v : candidatesList) {
  Set<Integer> neighbors = graph.getNeighbors(v);
  Set<Integer> newClique = new HashSet<>(clique);
  newClique.add(v);
  bronKerbosch(graph, newClique, intersection(candidates, neighbors), intersection(excluded, neighbors));
  candidates.remove(v);
  excluded.add(v);
  }
  }
  private static Set<Integer> intersection(Set<Integer> set1, Set<Integer> set2) {
  Set<Integer> intersection = new HashSet<>(set1);
  intersection.retainAll(set2);
  return intersection;
  }
	0
	1
	1
	1
	1
	2n
	1
	
	idr
	1
	
	4
	1
	
	1
	
	1
	
	1
	idk
	1
	1
	1
	4
	
	1
	1
	
	4
	1
	idk cause is for
	1

Total (la suma de todos los pasos)	
Clasificación en notación O Grande	

Búsqueda Local

Código fuente Solo se analiza el código del algoritmo.	Medición de líneas ejecutadas en el peor de los casos (línea por línea)

public class LocalSearchMaxCliqueSameResult {
  public static Set<Integer> localSearchMaxClique(Graph graph) {
  Set<Integer> candidates = new HashSet<>() ;
  for (int v = 0; v < graph.getCantidadDeVertices(); v++) {
  candidates.add(v);
  }
 	
  Set<Integer> bestClique = null;
 	
  for (int v : candidates) {
  Set<Integer> currentClique = new HashSet<>();
  currentClique.add(v);
 	
  for (int u : candidates) {
  if (currentClique.contains(u)) continue;
  boolean isConnectedToAll = true;
  for (int w : currentClique) {
  if (!graph.tieneConexion(u, w)) {
  isConnectedToAll = false;
  break;
  }
  }
  if (isConnectedToAll) {
  currentClique.add(u);
  }
  }
 	
  if (bestClique == null || bestClique.size() < currentClique.size()) {
  bestClique = currentClique;
  }
  }
  return bestClique;
  }
	1
	1
	1
	
	1
	
	1
	
	1
	1
	
	1
	
	1
	1
	
	1
	
	1
	
	1

Total (la suma de todos los pasos)	
Clasificación en notación O Grande	

Conclusiones

En conclusión, el análisis comparativo entre el algoritmo de Bron-Kerbosch y la Búsqueda Local para el problema del máximo clique revela diferencias significativas en términos de complejidad temporal y eficiencia computacional.

Por un lado, el algoritmo de Bron-Kerbosch presenta una complejidad temporal exponencial en el peor de los casos, lo que lo hace menos eficiente para grafos densos o con muchos cliques. Su enfoque exhaustivo de explorar las posibles combinaciones de vértices puede resultar en un alto costo computacional, especialmente para grafos relativamente pequeños. Aunque puede encontrar el clique máximo, su rendimiento puede deteriorarse para grafos grandes debido a su naturaleza poco selectiva.

Por otro lado, la Búsqueda Local ofrece una alternativa con una complejidad temporal que depende del número de iteraciones y la cantidad de operaciones realizadas en cada iteración. Su enfoque más selectivo de explorar solamente un subconjunto de combinaciones de vértices la hace más adecuada para grafos grandes. Esto se traduce en un mejor rendimiento en términos de tiempo de ejecución y recursos computacionales en comparación con Bron-Kerbosch, especialmente para grafos densos.

En nuestra evaluación analítica y empírica, la Búsqueda Local demostró ser más eficiente en la mayoría de los casos, especialmente para grafos grandes con densidad variable. Sin embargo, es importante tener en cuenta que la eficiencia de cada algoritmo puede variar según la entrada de los datos, incluyendo la cantidad de vértices y arcos en el grafo.

Bibliografía

[1] P. R. Östergård, “A fast algorithm for the maximum clique problem,” Discrete Applied Mathematics,vol. 120, no. 1, pp. 197–207, 2002. Special Issue devoted to the 6th Twenty Workshop on Graphs and Combinatorial Optimization.

[2] J. C. Ponce, E. E. P. de León, A. Padilla, F. Padilla, and A. O. O. Zezzatti, “Algoritmo de colonia de hormigas para el problema del clique máximo con un optimizador local k-opt,”Hífen, Uruguaiana, vol. 30, no. 58, pp. 191–196, 2006.

[3] G. V. GÓMEZ, El problema del clique máximo: Análisis, resolución e implementación. PhD thesis, Universidad Nacional Autónoma de México, 2019.

[4] S. Cagnoni, "Applications of Evolutionary Computing," Springer, pp. 112-120, 2002.

[5] Wayne Pullan, "Dynamic Local Search for the Maximum Clique Problem," 2006, pp. 159-185.

[6] J. Ponce, E. Ponce de Leon Senti, A. Padilla, F. Padilla, y A. Ochoa-Zezzatti, "Algoritmo de Colonia de Hormigas para el Problema del Clique Máximo con un Optimizador Local K-opt," págs. 190-196, 2008.

[7] D. Smith, R. Montemanni, y S. Perkins, "The Use of an Exact Algorithm within a Tabu Search Maximum Clique Algorithm," *Algorithms*, vol. 13, pág. 253, octubre 2020. doi: 10.3390/a13100253.

[8] G. Villeda Gómez, "El problema del clique máximo: Análisis, resolución e implementación," Tesis de Licenciatura, Universidad Nacional Autónoma de México, Facultad de Ciencias, Ciudad Universitaria, Ciudad de México, 2019, pp. 35-36.