BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"ANÁLISIS DEL CÓMPUTO EXHAUSTIVO DE CONJUNTOS INDEPENDIENTES"

T E S I S

Para obtener título de:
LICENCIADO EN INGENIERIA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
Juan Antares Perdomo Flández

Asesor:
M. C. Pedro Bello López

Puebla, Pue. Junio 2015

Tabla de contenido

Introducción

Los grafos han extendido su uso a varias áreas de la ciencia en donde es posible representar estructuras por medio de éstos y de igual forma, resolver problemas de diferentes tipos. Algunas de las ciencias en las que se ha aplicado son física [25], química [21] para la representación de las estructuras moleculares, sociología [27] para el estudio de las redes sociales humanas e igualmente en áreas de computación como son las redes computacionales en las cuales existen topologías de conexión como estrella, bus o anillo y que para analizar su desempeño hay que usar algoritmos ya existentes en la teoría de grafos.

A su vez los grafos son usados en el campo de la matemática, en donde es posible representarlos como conjuntos y aplicar reglas para extender las herramientas para la resolución de problemas.

Algunos de los trabajos más famosos realizados en este campo son los hechos por Dijkstra [28], cuyo algoritmo propuesto resuelve el problema del camino más corto y Kruskal [29], que propone un método para encontrar el árbol recubridor mínimo. Los algoritmos inicialmente propuestos se han modificado para adaptarse a la naturaleza de los problemas, es posible ver el algoritmo de Dijkstra aplicado al momento de usar el GPS y los servicios de mapas digitales.

En internet para búsquedas son igualmente usados como árboles de información, en donde se indican jerarquías de relevancia o en inteligencia artificial para la toma de decisiones, para éstos últimos se suelen usar árboles binarios.

La necesidad de presentar herramientas y métodos que faciliten y mejoren el estudio y la aplicación respectivamente se vuelve relevante para problemas que continúan surgiendo en las diferentes áreas.

Los conjuntos independientes, al igual que lo anterior mencionado, juegan un papel importante en la investigación y se han presentado varios problemas y soluciones en esta área. Algunos problemas se enfocan en encontrar los conjuntos maximales, algunos otros en encontrar al primer conjunto independiente máximo. Así mismo existen técnicas que han dado buenos resultados en la mayoría de los casos enfocándose a alguna clase de grafos como pueden ser del tipo cadena o ciclos anidados como el presentado en [14], grafos conexos [26], grafos con algún limite o número exacto de grado [16], y grafos circulares [17] entre otros.

Lo que se presenta 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.

Objetivos Generales y Específicos del Proyecto

Objetivos Generales

- Diseñar e implementar un algoritmo que dando como entrada la estructura de un grafo cualquiera, obtenga como resultado la cantidad de conjuntos independientes que existen en él y muestre dichos conjuntos ordenados por tamaño.

Objetivos Particulares

- Realizar un análisis de la complejidad del algoritmo.
- Crear tres versiones de la aplicación en las plataformas: De escritorio, Aplicación Web y Aplicación Móvil.
- Analizar los límites en memoria y en procesador en las diferentes plataformas.
- Comparar el comportamiento con diferentes estructuras de grafos y obtener cuales son los factores que influyen en su desempeño.

Capítulo 1. Marco teórico

En la teoría de grafos se han estudiado problemas que han dado a conocer diferentes estructuras, propiedades o elementos los cuales requieren una explicación de cómo están formados y cómo es que funcionan dadas sus características.

A continuación se exponen diversos conceptos que se han considerado tienen relación con el problema de los conjuntos independientes, de igual forma se mencionan otros como los grafos ponderados y grafos dirigidos que a la vez se explica la razón por la cual se descartan del estudio o la investigación.

1.1 Conceptos de grafos

Un grafo es un a tupla de conjuntos de nodos que están unidos por aristas. Estos pueden ser representados de muchas formas dependiendo de su aplicación.

Reinhard Diestel [9] define a un grafo como:

- Un par $G = (V, E)$ de conjuntos tal que $E \subseteq [V]^2$; así, los elementos de E son 2 subconjuntos de V. Los elementos de V son los vértices (nodos o puntos) del grafo G, los elementos de E son sus aristas (o líneas).

Mientras que Reinaldo Giudici [7] define en su libro:

- Un grafo finito G es un par $(V(G), E(G))$, donde $V(G)$ es un conjunto finito, no vacío, cuyos elementos son llamados vértices, y $E(G)$ es un conjunto de pares de vértices de $V(G)\, x\, V(G)$ que definen una relación R, de modo que si los vértices están en la relación, existe al menos una línea que los une.

En la anterior definición también se pueden incluir los multígrafos, los cuales son aquellos que tienen más de una arista para un par de nodos o aristas que relacionan al mismo nodo. En el presente trabajo solo se estudiarán los grafos simples, los cuales por el contrario solo tienen una sola arista para cada par de nodos y no incluyen bucles o aristas paralelas (aristas que relacionan al mismo nodo en sus extremos).

Cardinalidad

La cardinalidad de un conjunto no es más que la cantidad de elementos que dicho conjunto contiene. La cardinalidad de los vértices o aristas de un grafo es a menudo mencionada y se suele denotar entre plecas (||) como $|V|$ y $|E|$. En algunos libros se hace referencia de igual forma al hablar de la cardinalidad de los vértices de un grafo como $|G|$ y de las aristas como $||G||$.

Grafo no dirigido

Los grafos no dirigidos son un tipo de grafos donde no importa el orden en que se hagan los recorridos a través de las aristas de los nodos del grafo, por lo que se puede recorrer de igual forma del nodo 1 al nodo 2 como del 2 al 1 mientras exista una arista que una a ambos nodos.

Una definición formal de esto se presenta a continuación:

- Un grafo no dirigido se define como el conjunto de nodos N y aristas A que forman el grafo en donde el orden no importa, es decir que un grafo $G(N, A)$ no será dirigido si para toda arista $a\,|a \in A$ y cualquier par de nodos $ni, nj|\ ni, nj \in N$ se cumple que $a(ni, nj) = a(nj, ni)\ \forall\ a \in G$.

Grafos no ponderados

Al hablar de grafos ponderados se habla de aquellos que atribuyen un peso a la arista, esto es común verlo en el análisis del camino más corto, donde se hace una búsqueda a través de las aristas por las que es necesario pasar para que se vaya de un nodo inicial a un nodo final y, que este conjunto de aristas sea el que presente el menor peso. Sin embargo en el estudio de los conjuntos independientes de los grafos es irrelevante la ponderación de las aristas.

Los grafos no ponderados no presentan algún peso dentro de las aristas por lo que no existe diferencia al recorrer los grafos sea cual sea su camino de un nodo a otro.

Dentro del estudio ya se ha mencionado que se incluyen grafos simples que pueden contener ciclos, lo cual eleva la complejidad en muchos casos en el desarrollo de los algoritmos. También se estudiarán los casos más simples en donde se presenten grafos sin ciclos (árboles).

Grafos conexos y no conexos

Un grafo es conexo si para cualquier par de vértices existe una ruta por medio de las aristas que pueda unir ambos vértices (Fig. 1.1), por el contrario si no existen rutas posibles el grafo será no conexo o disconexo (Fig. 1.2). A pesar de que ambos tipos de grafos se incluyen dentro del estudio de los conjuntos independientes, es necesario separarlos en la clasificación debido a que podrían elevar o disminuir la complejidad al ser procesados.

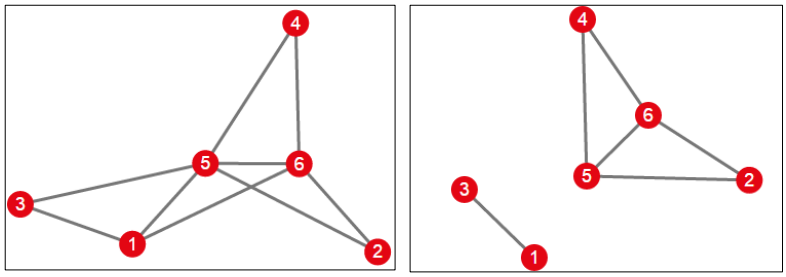


Fig. 1.1 (izquierda) Grafo conexo
Fig. 1.2 (derecha) Grafo no conexo o disconexo.

Formalmente:

- Un grafo no vacío G es llamado conexo si cualquier par de vértices están conectados por una ruta en G.

Grafos completos y desconectados

Estos tipos de grafos también se toman en cuenta ya que de igual forma pueden elevar o disminuir la complejidad del proceso.

Un grafo simple será completo si cada nodo está conectado a todos los demás nodos del grafo (Fig. 1.3), por el contrario de los grafos desconectados que no tienen absolutamente ninguna arista (Fig. 1.4). Estos bien pueden presentar tanto el mejor como el peor caso de el algoritmo respectivamente ya que son descartables las posibilidades cuando se analiza y cuenta la cantidad de conjuntos independientes existentes en los grafos.

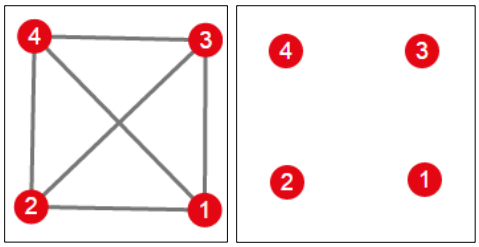


Fig. 1.3 Grafo completo (izquierda).
Fig. 1.4 Grafo disconexo (derecha).

Densidad de un grafo

Tomando en cuenta que la cantidad de conjuntos independientes se basa en la cantidad de aristas existentes, es decir, si hay pocas aristas habrá más conjuntos independientes y viceversa si hay muchas aristas habrá pocos conjuntos independientes, es necesario hacer énfasis en la densidad de los grafos cuyo valor indica el porcentaje de aristas existentes con respecto al máximo número de aristas para la cantidad de nodos que tiene el grafo.

La densidad de un grafo es una medida en base a la relación entre el número de nodos existentes en el grafo, así como, la cantidad de aristas.

Para grafos simples no dirigidos la densidad se puede obtener mediante la siguiente fórmula

$$D = \frac{2|E|}{|V|(|V|-1)}$$

Por lo tanto un grafo completo tendrá densidad $D = 1$ y un grafo desconectado tendrá densidad $D = 0$.

Para términos de esta investigación diremos que un grafo es denso si $D \geq 0.5$ en caso contrario diremos que es un grafo esparcido.

Valencia o grado

La valencia o grado de un nodo es la cantidad de aristas que inciden en él, se representa como $\delta(x)$. El grado máximo y mínimo de un grafo se representan respectivamente como $\Delta(G)$ y $\delta(G)$.

Adyacencia

La adyacencia se refiere a la vecindad de unos nodos con otros. Un par de nodos serán adyacentes si existe una arista que los una.

Grafo complemento

Sea G un grafo cualquiera, su complemento $\bar{G}$ será aquel que tenga el mismo conjunto de vértices que G pero presente solo las aristas que no están en el grafo original.

Formalmente: El complemento $\bar{G}$ de G es el grafo en V con conjunto de aristas $[V]^2 \setminus E$.

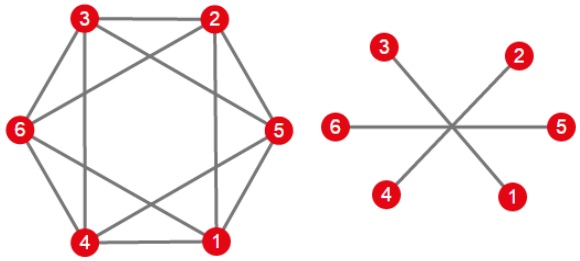


Fig. 1.5 Complemento de un grafo, en el complemento solo están las aristas que no existen en el otro.

1.2 Los conjuntos independientes

Los conjuntos independientes de un grafo son todos aquellos conjuntos de nodos que forman un grafo desconectado. Los conjuntos pueden tener cualquier tamaño dependiendo del grafo. Para el conteo se incluye el conjunto vacío, los conjuntos en donde se agrupan cada uno de los nodos individuales, los cuales son de tamaño uno, y el resto de conjuntos en donde se incluyen a todos los nodos que no tienen aristas en común.

Algunas definiciones se presentan a continuación:

1. Un conjunto independiente de G es un subconjunto $S \subseteq V$ de vértices tal que ningún par de vértices en S están conectados por una arista de G.
2. Dado un grafo G, un conjunto independiente es un subconjunto S de vertices en G tal que no hay dos vertices en S que sean adyacentes (conectados por una arista).

En partes posteriores del presente documento se hace la abreviación “CI” para indicar que se está hablando de conjuntos independientes.

Otro concepto que se estudia a la par de los conjuntos independientes de un grafo son los conjuntos independientes maximales.

Los maximales son un subconjunto de los conjuntos independientes de un grafo donde, no se puede añadir ningún otro nodo al conjunto para que este siga siendo independiente. Los conjuntos independientes maximales pueden tener el tamaño de cualquier otro conjunto independiente. Por ejemplo si el nodo de un grafo estuviera conectado a todos los otros nodos, un conjunto maximal del grafo sería aquel que solo incluyera a dicho nodo, ya que no se podría agregar a ningún otro y que el conjunto conserve su propiedad de independencia. De igual forma pueden existir varios conjuntos maximales del mismo tamaño.

Algunas definiciones de los conjuntos independientes maximales son:

1. Un conjunto independiente se llama maximal si no existe otro conjunto independiente en el que esté contenido propiamente. Esto es, un conjunto independiente I es maximal si para todo conjunto H tal que $I \subset H$ con $I \neq H$, se tiene que H no es independiente.
2. Un conjunto independiente maximal de un grafo es un conjunto independiente que no puede ser expandido a otro conjunto independiente por adición de algún otro vértice en el grafo.

El tamaño máximo de los conjuntos independientes de un grafo se ve limitado por la cantidad de nodos del mismo grafo, así como la cantidad de aristas, aunque no existe ninguna regla matemática que obtenga el tamaño máximo de los conjuntos independientes de un grafo. Esto se conoce como el conjunto máximo de un grafo o el conjunto independiente máximo y puede existir más de un conjunto con el tamaño máximo para un grafo dado.

Formalmente:

- Un conjunto independiente máximo es un conjunto independiente, el cual contiene la mayor cantidad de nodos para un grafo dado, por lo tanto, no existe otro CI que contenga más nodos.

El problema de encontrar el conjunto independiente máximo ha sido extensamente estudiado por algoritmos exactos y es un problema NP-Difícil.

1.2.1 Clique

Los cliques son los opuestos a los conjuntos independientes, son conjuntos de nodos que están unidos entre sí, o bien en otras palabras, son los subgrafos completos.

Al igual que en los conjuntos independientes, se presentan los cliques maximales y los máximos, los cuales tienen una definición similar que en los conjuntos independientes.

- Un clique maximal es un clique que no puede ser extendido por adición de otro nodo adyacente, por lo tanto este no sería un subconjunto del máximo clique.
- Un clique máximo del grafo G es un clique del tamaño máximo para G.

La importancia de los cliques para la obtención de los conjuntos independientes de un grafo es debido a que un conjunto independiente de G, será un clique del grafo complemento de G.

La búsqueda del clique máximo pertenece a la misma clase de complejidad NP-Difícil al igual que, la búsqueda del conjunto independiente máximo. Sin embargo son problemas que se pueden atacar por diferentes frentes, así mientras en un grafo será más rápido hallar los conjuntos independientes si dicho grafo es denso, para grafos esparcidos será más rápida la búsqueda de los cliques.

1.3 Complejidad de los algoritmos

La complejidad de los algoritmos ayuda a observar la eficiencia de un conjunto de instrucciones, frente a algunos datos de entrada. Sirve para evaluar si el algoritmo que se estudia es útil y también como medida de comparación frente a otros algoritmos.

Al estudiar la complejidad de un algoritmo se hace énfasis en dos puntos principales, los cuales son: el tiempo y el espacio (o la cantidad de recursos que se usan).

Algunas veces se centra el estudio más en el tiempo que en el espacio debido a que actualmente se pueden almacenar grandes cantidades de datos, pero es necesario que estos se procesen dentro de un intervalo de tiempo "aceptable". El tiempo puede representarse por funciones que indican cómo aumentará la tardanza del procesamiento en función de los datos de entrada, estas funciones pueden ser obtenidas estudiando la cantidad de ciclos que existen, el uso de métodos recursivos y las anidaciones de estas principalmente.

Es claro que con datos específicos de entrada, el comportamiento de los algoritmos puede variar, para ello hay que identificar tres casos para la evaluación de su rendimiento:

- El mejor caso, donde se pasará un mínimo de veces por los ciclos y se descartarán los bloques de mayor código en las condiciones.
- El peor caso, en este se considera que se repitan los ciclos un máximo de ocasiones y que las condiciones lleven a la mayor ejecución de instrucciones posible.
- El caso medio, es una posibilidad entre el mejor y el peor caso donde las instrucciones se ejecutarán de una forma intermedia, por ejemplo para un ciclo que va desde 1 a N, en el caso medio dicho ciclo se ejecutaría $N/2$ veces.

Cada uno de los tres casos generará una función de complejidad $T(n)$ que indica el tiempo en relación a los datos de entrada. De las funciones resultantes, la que mas interesa es la función $T(n)$ para el peor caso, debido a que es con ella con la que se clasifica su complejidad y se puede comparar con otros algoritmos.

Los casos mejor y medio a pesar de generar funciones que pueden servir igualmente para la clasificación de los algoritmos, se descartan en esta tesis debido a que por un lado el mejor caso solo servirá para comparar a partir de qué punto, con un conjunto de datos de entrada, el algoritmo pasa a tener una mejor

ejecución, ya que usualmente con pocos datos se tienen que ejecutar obligatoriamente una cantidad de instrucciones, en donde no se podría evaluar realmente su comportamiento. El caso medio aunque parecería ser una buena forma de evaluar las complejidades, tampoco se considera fiable debido a que en un algoritmo no todas las posibilidades son equiprobables, y por tanto la medida que se genera puede resultar errónea.

Algunas de las funciones de complejidad más comunes se presentan a continuación:

$O(k)$: Complejidad constante, este se da cuando bajo cualquier tipo o cantidad de datos de entrada a un algoritmo, siempre se ejecutan la misma cantidad de instrucciones.

$O(\log n)$: Complejidad logarítmica.

$O(n)$: Complejidad lineal, es una complejidad común y al igual que la complejidad logarítmica es considerada como buena.

$O(n \log n)$: Complejidad lineal logarítmica.

$O(n^2)$: Complejidad cuadrática.

$O(n^3)$: Complejidad cúbica.

$O(n^k)$: Complejidad polinómica, a partir de esta complejidad el tiempo de procesamiento se eleva bastante.

$O(2^n)$: Complejidad exponencial, aunque existen aún mayores, un algoritmo con este nivel de complejidad se considera ineficiente.

En base a estas cotas se clasifican las funciones de algoritmos en buenas o malas. Las funciones lineales, logarítmicas y cúbicas al estar acotadas superiormente por n^k para un valor k fijo, se le llama funciones polinómicas y se les atribuye ser eficiente o razonables, por otra parte funciones exponenciales o superiores a las que se les denomina superpolinómicas son ineficientes o no razonables.

Al igual que se pueden clasificar los algoritmos en razonables y no razonables, se pueden clasificar a los problemas que éstos presentan en tratables e intratables.

Las respectivas definiciones como se presentan en [12] son:

Problemas tratables: son aquellos que tienen ambos límites superior e inferior que caracteriza a N solo como un factor polinomial en la notación $O()$.

Problemas intratables: son aquellos que tienen ambos límites superior e inferior que caracteriza a N solo como un factor exponencial en la notación $O()$.

Así se les considera a los problemas tratables como computacionalmente solubles. Los problemas intratables, a pesar de que no se consideran solubles por medio de la computación, no quieren decir que no se puedan resolver. El objetivo para problemas intratables es encontrar algoritmos razonables que aproximen su solución.

1.3.1 Las clases de complejidad

Las clases de complejidad computacional son conjuntos de problemas que tienen una complejidad similar. Estas se definen a partir del tipo de problema de donde surgen y las cotas que lo delimitan. Algunas de las clases de complejidad son subconjuntos de otras clases o bien, tienen problemas en común con otras clases.

Se define una clase de complejidad como:

- Un conjunto de funciones que pueden calcularse dentro de un mismo rango de recursos de cómputo (ya sea tiempo o espacio de cómputo).

Los de problemas que se resuelven con algoritmos eficientes pertenecen a la clase de complejidad polinomial o clase P, los cuales usan un tiempo polinomial de cómputo para resolver cualquier problema de esta clase.

1.3.2 Determinismo y no determinismo

Un algoritmo será determinista si para un conjunto especifico de datos de entrada, siempre se presentará la misma salida. Una forma de que un algoritmo deje de ser determinista es que se incluyan en él decisiones no deterministas, como bien puede ser variables aleatorias o alguna interacción externa.

1.3.3 Las clases NP y #P

La clase NP engloba problemas de decisión (problemas cuyo resultado buscado es un “si” o un “no”), para los cuales existen algoritmos no deterministas que tienen un tiempo de ejecución polinómico, sin embargo no existen algoritmos deterministas que tengan un tiempo de ejecución razonable. Esta clase tiene un gran interés debido a que es la frontera entre problemas bien resueltos y los que no se sabe si pueden resolverse eficientemente.

Esta clase engloba a problemas NP-Completos, en los cuales se encuentran los anteriormente mencionados, el problema del máximo clique y del máximo conjunto independiente. Estos forman parte de los problemas más difíciles de la clase NP.

En el teorema de Cook se da una forma de determinar si un problema es NP-Completo. Con este mismo teorema se demuestra, que si se llega a averiguar un algoritmo determinista que solucione el problema en tiempo polinomial, todos los problemas NP se podrán resolver también polinomialmente y por tanto la clase NP sería equivalente a la clase P.

La clase #P presenta conjuntos de problemas NP, vistos como problemas de conteo en lugar de problemas de decisión. El algoritmo que se presenta en la investigación pertenece a esta categoría, ya que lo que se busca como resultado es la cantidad de todos los conjuntos independientes.

Capítulo 2. Diseño del algoritmo de CI

Para el diseño del algoritmo primero fue necesario entender cómo es que cotidianamente se cuentan los CI de un grafo. Esto se hace incrementalmente, primero contando todos los CI de tamaño = 1, después los de tamaño = 2 y así consecutivamente hasta terminar.

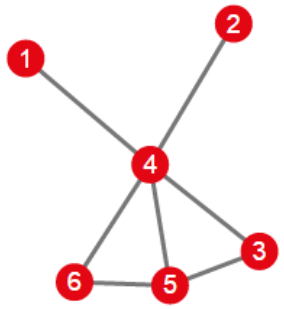


Fig. 2.1 Grafo de 6 nodos y 7 aristas

Un ejemplo de los conjuntos independientes, se puede mostrar usando el grafo de la figura 2.1. Con el grafo presentado se pueden obtener los conjuntos independientes por tamaño o por número de elementos como se muestra:

Tamaño 1 = {1}, {2}, {3}, {4}, {5}, {6} cantidad = 6

Tamaño 2 = {1,2}, {1,3}, {1,5}, {1,6}, {2,3}, {2,5}, {2,6}, {3,6} cantidad = 8

Tamaño 3 = {1,2,3}, {1,2,5}, {1,2,6}, {1,3,6}, {2,3,6} cantidad = 5

Tamaño 4 = {1,2,3,6} cantidad 1

En total se obtienen 20 + conjunto vacío se tienen en total 21 conjuntos independientes.

2.1 Algoritmo de conjuntos independientes

Como se sabe los grafos pueden ser representados de diferentes formas, dos de ellas son usando cierto tipo de listas ligadas, donde cada nodo apuntará a todos los nodos adyacentes a él, sin embargo de esta forma no se puede tener un control total de los elementos de un grafo los cuales son nodos y aristas. En esta forma de representación, las aristas irían implícitas, por lo que otra forma de representar un grafo son con pares de listas, una que contenga los nodos y otra que contenga las aristas, esta forma es más útil en especial para el dibujo de los grafos, debido a que se puede tener un control sobre la posición de ambos conjuntos de elementos que conforman a la estructura en general. En esta segunda forma de representación las aristas tienen dos apuntadores, uno a cada nodo a los que está conectada. Los nodos, siendo que pueden tener cero o más aristas incidentes, tendrán una lista de apuntadores a todas estas aristas.

Para encontrar todos los conjuntos independientes de un grafo, es necesario hacer un análisis exhaustivo sobre cada nodo, verificando si alguno del resto de nodos es adyacente, en caso de no serlo se puede continuar la búsqueda de algún nodo que no sea adyacente, ni a el primer nodo, ni al segundo, y así continuar en busca de conjuntos de mayor tamaño hasta encontrar un maximal, un conjunto independiente cuya adición de cualquier otro nodo, hará que este deje de ser independiente.

Usando la forma de representación de grafos, de la que se habló con anterioridad, se pueden hacer estas búsquedas, sin embargo, termina teniendo cierta ineficacia debido a la naturaleza de las listas ligadas que tienen que indexar en diferentes partes de memoria para acceder a sus elementos, y que también para ello es necesario recorrer los puestos anteriores.

Para este problema, se optó mejor por el uso de otra forma de representación de los grafos, que es usando una tabla de $n\,x\,n$ (siendo n igual a la cantidad de nodos en el grafo) donde se enlistan a lo alto y a lo ancho todos los nodos, y en las intersecciones en la tabla de un nodo N1 con un nodo N2, se colocará un 1 o true si este par de nodos son adyacentes entre si, y un 0 o false en caso contrario.

Esta forma a pesar de que tampoco puede tener un gran control sobre, ni las aristas, ni los nodos, resulta más beneficiosa para la búsqueda de conjuntos independientes, debido a que todo se hace en un área de la memoria y que esta búsqueda se basa en encontrar las adyacencias de los nodos.

Con esta última forma de representación se ha trabajado el algoritmo que se presenta a continuación.

Una vez se haya leído un grafo y se tenga en la tabla se pueden encontrar los primeros conjuntos independientes, los cuales no necesitan ser procesados dentro del algoritmo.

El vacío es el primer conjunto en contarse, debido a que es un conjunto independiente sin importar el tamaño del grafo, ni la cantidad de conjuntos que se puedan encontrar. El vacío es un conjunto con cero elementos, y se clasifica como un conjunto de tamaño igual a cero.

A continuación, también sin necesidad de entrar a los ciclos del algoritmo, se pueden encontrar los conjuntos de tamaño igual a uno. Estos conjuntos serán cada uno de los nodos tomados en cuenta individualmente $\{1\}, \{2\}, \{3\}, \dots \{n\}$. Para este punto hay que aclarar que solo se toman en cuenta grafos simples (que no tienen dos aristas conectadas al mismo par de nodos, ni aristas conectadas al mismo nodo en ambos extremos). En caso de que se aceptaran usar grafos que contengan bucles, se tendrían que eliminar del conteo todos los conjuntos que contengan nodos con bucles debido a que estos nodos no son independientes de sí mismos y por tanto, tampoco podrían pertenecer a conjuntos de mayor tamaño. En caso de aristas múltiples (si se tomarán en cuenta), bastaría tomar una sola arista para cada par de nodos adyacentes, ya que no habría diferencia si un par de nodos está conectado por una o varias aristas, dicho par no será independiente ni será un subconjunto de un CI (conjunto independiente).

Otro tipo de grafos no simples, que usualmente no se toman en cuenta dentro del estudio de los conjuntos independientes, son los grafos dirigidos. Estos tipos de grafos se caracterizan por tener un orden, por tanto el conjunto $\{a, b\} \neq \{b, a\}$ por lo que un par de nodos $a \rightarrow b$ se podría decir que $\{b, a\}$ es un conjunto independiente debido a que b no tiene adyacencia con a, sin embargo el caso contrario $\{a, b\}$ no sería independiente ya que a es dependiente de b.

Con lógica similar a estos casos foráneos al trabajo, se pueden estudiar los conjuntos independientes para grafos no simples.

Continuando con el algoritmo, siendo que los grafos simples no contienen bucles, la cantidad de conjuntos independientes con tamaño igual a uno será la misma cantidad de nodos que existan en el grafo.

Hasta este punto el conteo de conjuntos no ha entrado en ningún ciclo. Para el resto de CI ya es necesario recorrer los nodos buscando las no adyacencias. Para

explicar más sencillamente el funcionamiento del algoritmo se usará un grafo de ejemplo.

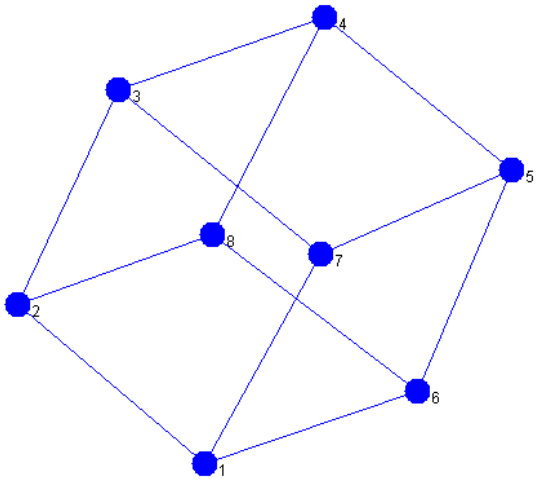


Fig. 2.2 Representación de un cubo en forma de grafo.

Este grafo será representado por la siguiente tabla:

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	1	0
2	1	0	1	0	0	0	0	1
3	0	1	0	1	0	0	1	0
4	0	0	1	0	1	0	0	1
5	0	0	0	1	0	1	1	0
6	1	0	0	0	1	0	0	1
7	1	0	1	0	1	0	0	0
8	0	1	0	1	0	1	0	0

Como se puede ver la tabla representativa es simétrica, esto ocurre para cualquier tipo de grafo simple, por lo que se puede solo recorrer la mitad y así reducir el procesamiento. También es innecesario tomar en cuenta a la diagonal debido a que no habrá conexión alguna, y como se mencionó, ya sabemos que los conjuntos con tamaño igual a uno será la cantidad de nodos que tenga el grafo.

La búsqueda de los CI empieza entonces de la parte superior de la tabla:

	1	2	3	4	5	6	7	8
1	0	1	0	0	0	1	1	0
2	1	0	1	0	0	0	0	1
3	0	1	0	1	0	0	1	0
4	0	0	1	0	1	0	0	1
5	0	0	0	1	0	1	1	0
6	1	0	0	0	1	0	0	1
7	1	0	1	0	1	0	0	0
8	0	1	0	1	0	1	0	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
2		1	0	0	0	0	1
3			1	0	0	1	0
4				1	0	0	1
5					1	1	0
6						0	1
7							0

La búsqueda se ejecuta recorriendo fila por fila en busca de las no adyacencias, las cuales están representadas por ceros. Hasta este punto el conteo es de 9 CI (8 con tamaño 1 más el vacío). Cuando encuentra un cero, se suma el contador en uno y salta a la fila del nodo donde se encontró el cero.

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
3			1	0	0	1	0

En este segundo nivel, se recorre igualmente en busca de ceros, sin embargo una vez se encuentre una no adyacencia en este nivel, no quiere decir que junto con 1 y 3 el nodo encontrado pueda ser independiente. Entonces una vez que se encuentre un cero en la fila del segundo nivel, pasará a verificar si es independiente de los nodos de los niveles anteriores.

Para verificar que el nodo en cuestión puede formar un CI con los nodos de ambos niveles de la tabla, se realiza una suma de los elementos de la columna de este nodo. Si el resultado de esta suma es igual a cero, entonces es un conjunto independiente y salta a la fila que representa a este nuevo nodo. El proceso se repite buscando nodos no adyacentes y verificando que estos sean independientes de los niveles anteriores

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
3			1	0	0	1	0
5					1	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
3			1	0	0	1	0
5					1	1	0

Una vez en algún nivel haya alcanzado el final de la fila, regresará a el nivel anterior, en la posición donde se había quedado y continuará la búsqueda de no adyacencias para el resto de nodos. En este caso, una vez que se encontró el conjunto $\{1,3,5,8\}$, se retornará al nivel tres y continuará la búsqueda hasta encontrar otro nodo no adyacente, también se verificará, como anteriormente se mencionó, de que el último elemento sea independiente con los elementos tomados en los niveles previos.

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
3			1	0	0	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
3			1	0	0	1	0

El proceso continuará de esta forma agregando y quitando niveles de la tabla y verificando cuando exista un posible conjunto independiente para algún nivel.

A continuación, se muestra parte del resto del procesamiento de la búsqueda:

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
4				1	0	0	1

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
4				1	0	0	1

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0
5					1	1	0

	2	3	4	5	6	7	8
1	1	0	0	0	1	1	0

Lo que se presenta hasta aquí da como resultado todos los conjuntos independientes que contengan al nodo 1, posteriormente se continuará con la siguiente fila y se realizará el mismo procedimiento de búsqueda hasta que se llegue a la última fila.

Al terminar toda la tabla con este ejemplo deberían dar 35 conjuntos independientes distribuidos de la siguiente forma:

Tamaño 0=1
Tamaño 1=8
Tamaño 2=16
Tamaño 3=8
Tamaño 4=2

2.2 Pseudocódigo

```
//Para grafo de n nodos
Array matriz[n][n];
int numeroCI = 0;

//Aquí se almacena (en forma de pila/stack) el índice de las filas que están procesándose
Lista enProceso;

//Se almacena el resultado en na lista de strings
Lista(string) resultado;

//Se almacena la cantidad de CI encontrados para cada tamaño de conjunto
Lista distribucion;

//Se inicializan los valores
resultado.agregar("");
distribucion.agregar(0);

Metodo independiente(){

    //Iniciamos con cero que corresponde a la primera fila de la matriz
    enProceso.agregar(0);
    for(i=0; i<n; i++){

        //Se concatena los resultados
        resultado[0] += "{"+(i+1)+"}";

        //Se suma la cantidad de CI para el primer tamaño
        distribucion[0]++;

        for(j=i+1; j<n; j++){
            if(matriz[i][j]==0){
                numeroCI++;
```

```
                //Se agrega una nueva fila al proceso
                enProceso.agregar(j);

            //Si ya están en proceso 2 filas entonces en el resultado, si
            no existe el segundo nivel, se agrega...
                if(resultado.tamaño()<enProceso.tamaño()){
                    resultado.agregar("{"+(i+1)+","+(j+1)+"}");
                    distribucion.agregar(1);
                }

                //...y si ya existe se concatenan los resultados
                else{
                    resultado.[1] += "{"+(i+1)+","+(j+1)+"}";
                    distribucion[1]++;
                }

                //Se inicia el método recursivo
                independiente2(j);

                enProceso.remover();
            }
        }
        //Cambiamos el valor para continuar con la siguiente fila
        enProceso[0]=i+1;
    }
}

Metodo independiente2(int j){
  j++;
  //Se recorre la fila actual
  for(;j<n;j++){
    int suma = 0;
    for(k=0;k<enProceso.tamaño(); k++){

      //Se hace la suma de los elementos de una columna sobre las   filas
      en proceso
      suma+=tabla[enProceso[k]][j];
    }

    //Si el resultado es cero entonces los respectivos nodos forma un CI
    y pueden ser un subCI de otro mayor
    if(suma==0){
      numeroCI++;

      //Aquí se añadirá consecutivamente filas a la lista con cada
      iteración del método recursivo
      enProceso.agregar(j);
```

```
      //Si el tamaño de la lista resultado es menos que las filas en
      proceso, se agregaran elementos a esta lista, en caso contrario se
      trabajará sobre el correspondiente nivel
      if(resultado.tamaño()<enProceso.tamaño()){


        //Aquí se concatena el correspondiente conjunto encontrado nodo
        por nodo
        resultado.agregar("{"+(enProceso[0]+1));
        distribucion.agregar(0);
        for(int k=1;k<enProceso.tamaño();k++){
          resultado[enProceso.tamaño()-1] += ","+(enProceso[k]+1);
        }
        resultado[enProceso.tamaño()-1] += "}";
        distribucion.[enProceso.tamaño()-1]++;
      }else{
        resultado[enProceso.tamaño()-1] += "{"+(enProceso[0]+1);
        for(int k=1;k<enProceso.tamaño();k++){
         resultado[enProceso.tamaño()-1] += ","+(enProceso[k]+1);
        }
        resultado[enProceso.tamaño()-1] += "}";
        distribucion.[enProceso.tamaño()-1]++;
      }

      //Se repite el proceso en busca de CI más grandes
      independiente2(j);
      enProceso.remover();
    }
  }
}
```

2.3 Pseudocódigo versión ligera

```
//Para grafo de n nodos
Array matriz[n][n];
int numeroCI = 0;
//Aquí se almacena (en forma de pila/stack) el índice de las filas que
están procesándose
Lista enProceso;
Método independiente(){
      //Iniciamos con cero que corresponde a la primera fila de la matriz
      enProceso.agregar(0);
      for(i=0; i<n; i++){
            for(j=i+1; j<n; j++){
                  if(matriz[i][j]==0){
                        numeroCI++;
                        //Se agrega una nueva fila al proceso
                        enProceso.agregar(j);
                        //Se inicia el método recursivo
```

```
                        independiente2(j);
                        enProceso.remover();
                  }
            }
            //Cambiamos el valor para continuar con la siguiente fila
            enProceso[0]=i+1;
      }
}

Método independiente2(int j){
      j++;
      //Se recorre la fila actual
      for(;j<n;j++){
            int suma = 0;
            for(k=0;k<enProceso.tamaño(); k++){
                  //Se hace la suma de los elementos de una columna sobre
                  las filas en proceso
                  suma+=tabla[enProceso[k]][j];
            }
            //Si el resultado es cero entonces los respectivos nodos
            forma un CI y pueden ser un subCI de otro mayor
            if(suma==0){
                  numeroCI++;
                  //Aquí se añadirá consecutivamente filas a la lista con
                  cada iteración del método recursivo
                  enProceso.agregar(j);

                  //se repite el proceso en busca de CI más grandes
                  independiente2(j);
                  enProceso.remover();
            }
      }
}
```

2.4 La complejidad del algoritmo

Para el cálculo de la complejidad del algoritmo, hay que analizar su comportamiento, se sabe que se tienen dos ciclos anidados los cuales harán el recorrido de la matriz, uno para recorrer las filas y otro para las columnas o los elementos de la fila, buscando a la par las no adyacencias o los CI de tamaño 2, hasta aquí tenemos una complejidad de $n * n$.

Dentro de este par de ciclos anidados se hace la llamada al método recursivo, el cual en cada llamada recorrerá los elementos de su respectiva fila, llamándose a sí mismo si es necesario para recorrer otras filas. De esto podemos observar que la complejidad de este método estaría dada por n multiplicada por las siguientes llamadas al mismo método que también sería n, esto se repite recursivamente hasta terminar por encontrar todos los CI para un nodo y así, continuar con la búsqueda de los demás CI, omitiendo ya los encontrados anteriormente.

Dado que la ejecución del algoritmo usa dos ciclos anidados, más las múltiples repeticiones que genera el método recursivo, se obtiene que la complejidad es de $O(2^n)$. Esta complejidad se encuentra dentro de la clase exponencial como se esperaba. Es necesario destacar que a pesar de que se estén usando ciclos anidados, lo cual no siempre es lo ideal, estos ciclos no recorren en la mayoría de los casos a todos los elementos de matriz, incluso en el peor de los casos que son los grafos disconexos, en donde todos los conjuntos existentes serán a su vez conjuntos independientes, los ciclos principales recorren la matriz sobre la mitad. El hecho de que no se recorra la matriz completa en los ciclos, no basta para reducir su grado de complejidad, sin embargo es una técnica que agiliza notablemente el proceso.

De la misma forma, para ser un algoritmo de búsqueda exhaustiva, el grado de complejidad es aceptable para este problema y como ya se mencionó, este método es aplicable para cualquier tipo de grafo sin importar la cantidad de ciclos que este tenga, ni la forma en que estén conectados.

2.5 Uso del algoritmo para encontrar los cliques

Los cliques, como ya se vio anteriormente, son lo opuesto de los conjuntos independientes, por lo que si los CI son conjuntos de nodos en donde ninguno de estos tiene alguna adyacencia, los cliques son conjuntos en donde todos los nodos estarán conectados unos con otros, es decir, que cada clique de un grafo es en sí un subgrafo completo. Un punto importante a destacar, es que los conjuntos de cliques, en unión con los conjuntos independientes de un grafo, no necesariamente dan como resultado todos los conjuntos posibles de los nodos del grafo, existen conjuntos de los grafos que no serán ni independientes ni cliques.

Otra característica importante, es que en el grafo complemento de cualquier grafo, los cliques formaran los mismos conjuntos que los CI del grafo original y viceversa.

Con estos datos se puede observar que para la búsqueda de los cliques, se puede modificar el algoritmo de este trabajo con la diferencia, de que si para los CI se hace la búsqueda en la matriz de los ceros (no adyacencias), para los cliques se hará la búsqueda de unos o de las adyacencias de los nodos. O por otro lado, se puede también complementar (0 -> 1 y 1->0) los valores de la matriz de forma que represente al grafo complemento del original, y hacer la búsqueda de CI los cuales serán los cliques del grafo original, lo cual tiene como ventaja que se puede usar exactamente el mismo algoritmo para la búsqueda de este otro tipo de conjuntos.

Capítulo 3. Las aplicaciones

Uno de los objetivos en este trabajo, fue desarrollar aplicaciones que calculen la cantidad de conjuntos independientes de cualquier grafo, esto con el fin de facilitar el estudio en esta área. Las aplicaciones se hicieron para tres plataformas, igualmente para que se pueda acceder a ellas de forma más sencilla.

El programa con más funciones es el de escritorio, fue desarrollado en Java y además de calcular y mostrar los conjuntos independientes, también dibuja el grafo que se esté calculando, con la posibilidad de modificarlo agregando o quitando nodos o aristas y pasarlo a texto, una vez que se hayan hecho las alteraciones, se pueden abrir los grafos desde archivos de texto mientras tengan el formato que se especifica y de igual forma se puede guardar el grafo, además que se puede guardar también la imagen del grafo que se haya dibujado.

Cuenta también con un generador de grafos aleatorios que funciona indicando la cantidad exacta de nodos y aristas que se desean, con lo cual se puede estudiar el comportamiento de grafos de la misma densidad.

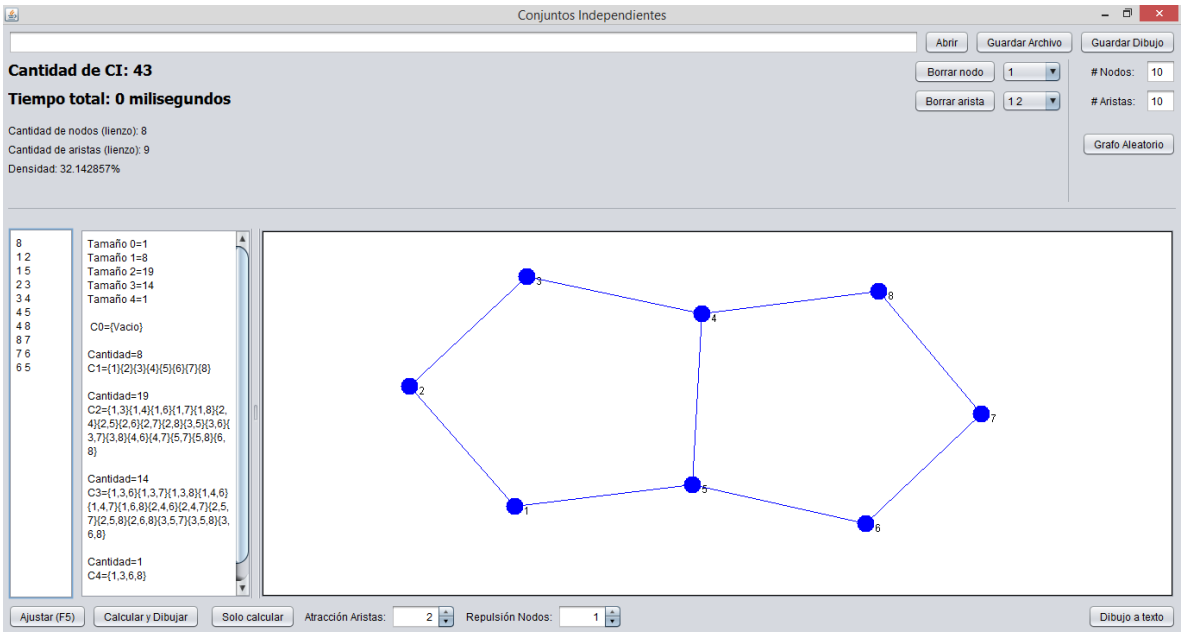


Fig. 3.1 Aplicación de escritorio con área de dibujo extendida.

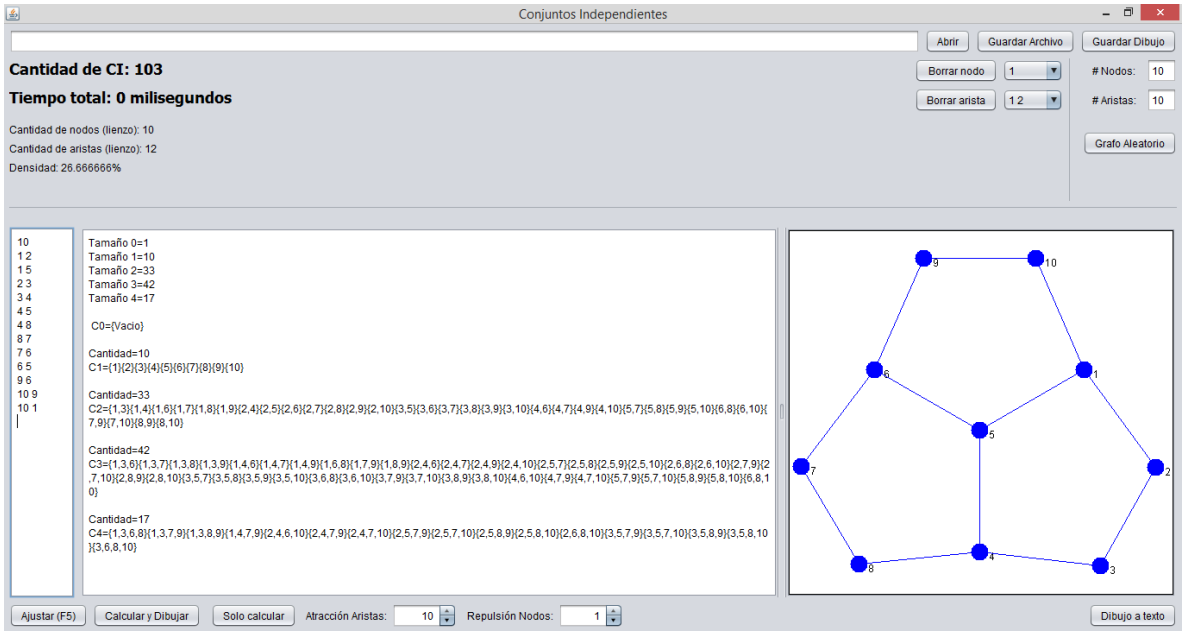


Fig. 3.2 Aplicación de escritorio con área de resultados extendida.

La siguiente aplicación, se desarrolló para páginas web en PHP en la cual, a pesar de no tener tantas funciones como la anterior, es capaz de calcular los CI mucho más rápido con una diferencia bastante amplia, como mínimo 70% más rápido (99% para el grafo doble estrella snark), en relación a la aplicación de escritorio como se demuestra en la sección de resultados.

Esta segunda aplicación se apoyó para la creación del dibujo de un algoritmo, que se puede insertar como código embebido mediante iframes, el cual funciona de manera similar al que tienen las otras dos aplicaciones. La fuente de donde se tomo es: http://g.ivank.net/

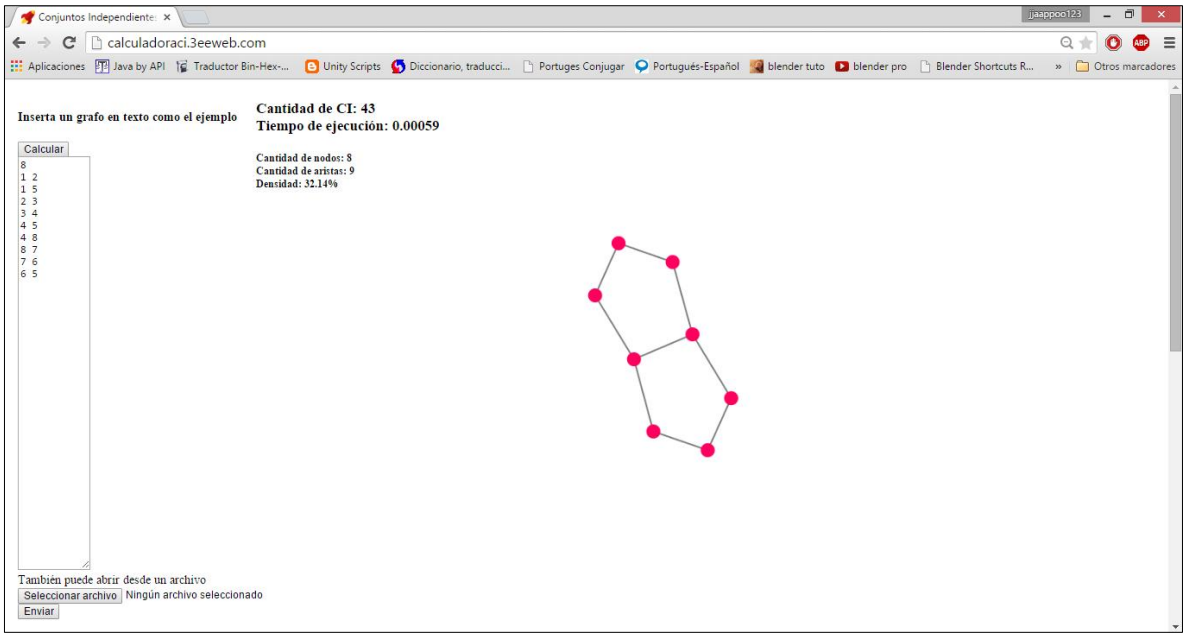


Fig. 3.3 Aplicación web pantalla principal.

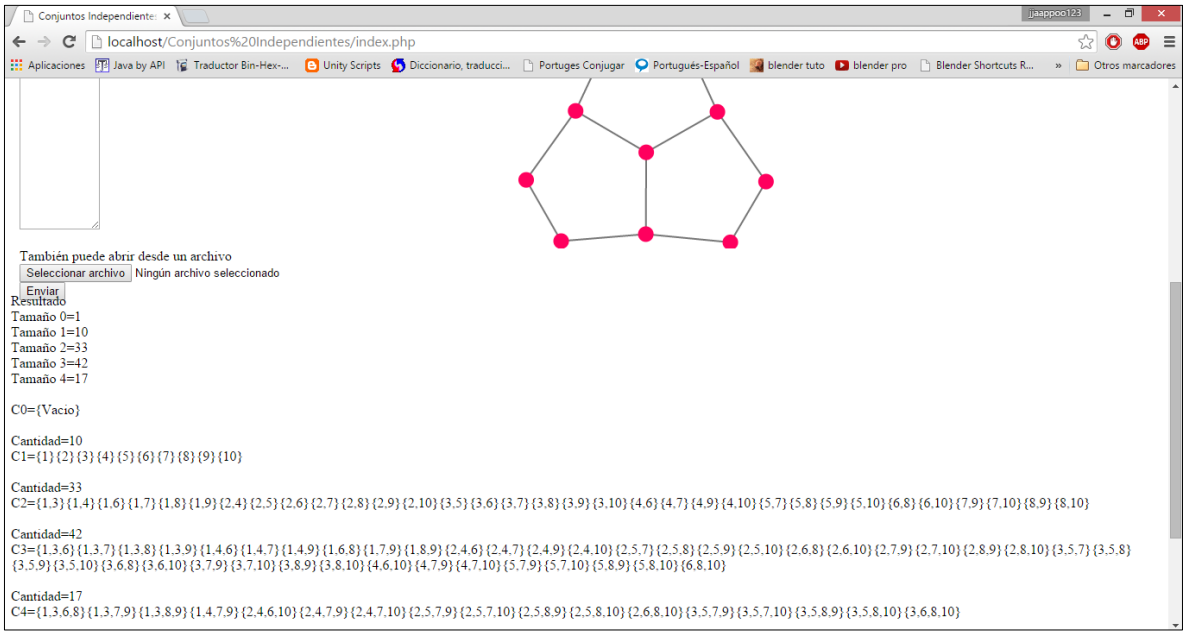


Fig. 3.4 Aplicación web Área de resultados.

La aplicación móvil se creó teniendo en cuenta las limitaciones de los procesadores y de la memoria de estos dispositivos, por lo que se consideró presidir de algunas funciones como mostrar el resultado de los CI encontrados, sin embargo se mantuvieron otras opciones importantes como el abrir los grafos desde archivos de texto y dibujarlos.

Fig. 3.5 (Izquierda) Aplicación móvil pantalla principal.
Fig. 3.6 (Derecha) Aplicación móvil pantalla de dibujo.

3.1 El algoritmo de dibujado.

Si bien, este algoritmo no es parte de los objetivos que se propusieron, no deja de ser una herramienta para representar a los grafos en formas bidimensionales. Este método no es nuevo, y es uno de varios algoritmos capaces de dibujar grafos, se escogió este algoritmo debido a que logra posicionar los nodos de una forma natural. La forma en que funciona, es simulando fuerzas de atracción y repulsión en cada nodo según convenga.

En el algoritmo que se desarrolló, se supuso que todos los nodos se repelerían entre si unos a otros, de una forma similar a la que funciona la gravedad donde entre más lejos los cuerpos menos interacción existirá, la fórmula para lograr esto se encuentra en la ley de Coulomb, en donde la fuerza de repulsión es inversamente proporcional al cuadrado de la distancia entre ambos nodos. Sin embargo, esto no es suficiente para que los nodos se coloquen en sus posiciones, debido a que si solo se deja esta fuerza repulsiva, los nodos se alejarán unos de otros indefinidamente. Dado este problema, se consideró usar la misma fuerza en las aristas, siendo que mientras que todos los nodos se repelen entre sí, al mismo tiempo se atraen con aquellos con los que sean adyacentes.

Al aplicar de esta forma, las fuerzas tanto de atracción como de repulsión según la fórmula de Coulomb, se logra que el grafo se expanda sin sobreponer muchas aristas pero se conserva el problema de que se expande más allá de lo deseado, por la razón que también se está considerando que la atracción disminuye con la lejanía y debido a esto, el movimiento de los nodos queda descrito por la fuerza dominante que es la repulsión.

Entonces, para poder encontrar un equilibro entre la repulsión y la atracción, se debe considerar otro tipo de fuerza que evite que se expanda indefinidamente y logre encontrar un equilibro. Para lograr esto, se consideró que las fuerzas que ejercen las aristas fueran como la de una liga, en donde por el contrario de la ley de coulomb, la fuerza que se ejerce entre dos puntos de ésta es directamente proporcional a la distancia, esto quiere decir que entre más lejos estén los nodos conectados, mayor será su fuerza de atracción. Al usar este otro tipo de fuerza de atracción se logra que los nodos se acomoden de forma que se pueda visualizar el grafo sin sobreponer tantas aristas, y que mientras dura este proceso al mismo tiempo encuentre un equilibro entre atracción y repulsión (Fig. 3.7).

Al implementar esté algoritmo, también hay que considerar variables comunes en la física como lo es la velocidad, tiempo y aceleración. Y otros métodos matemáticos, como es la suma entre vectores el cual se usa para saber la

dirección a donde se moverá alguno de los nodos respecto a los demás, según indiquen las fuerzas incidentes en este.

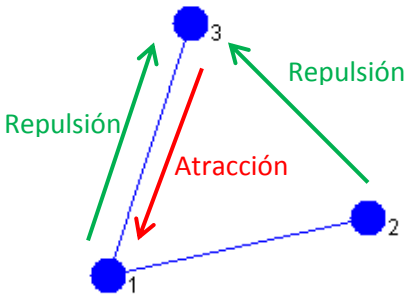


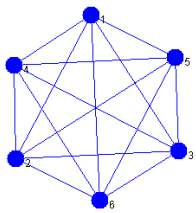
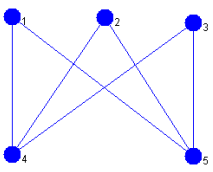
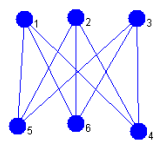
Fig. 3.7 Fuerzas de interacción en el nodo 3

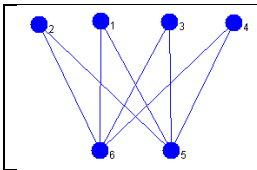
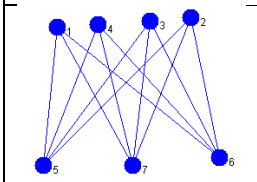
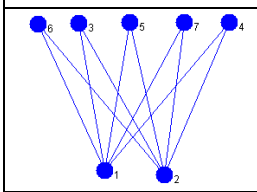
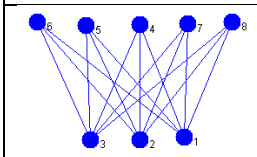
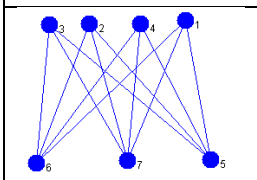
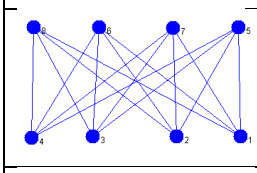
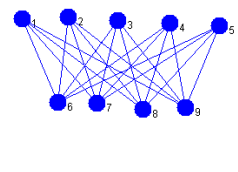
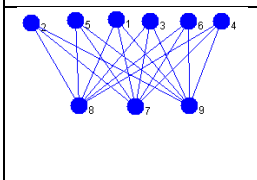
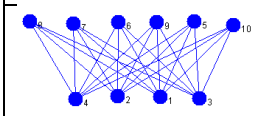
Capítulo 4. Resultados generales

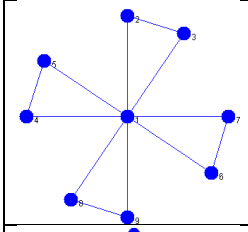
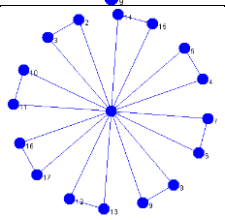
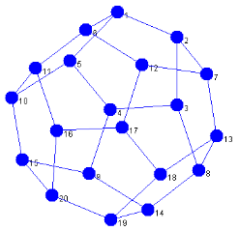
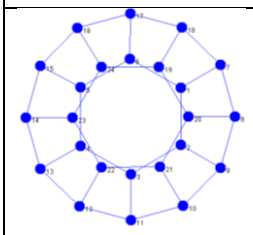
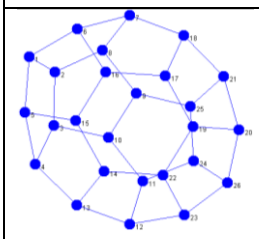
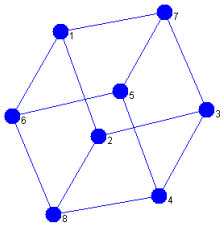
En cuanto a los resultados que presentaron las aplicaciones, se encontró una deferencia de velocidad notable entra la aplicación de escritorio con respecto a la aplicación web en donde para grafos grandes (entre 20 y 30 nodos), la aplicación de escritorio llegó a tardar varios minutos en comparación con la aplicación web, que para el mismo grafo solo tardaba algunos segundos.

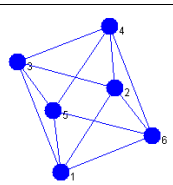
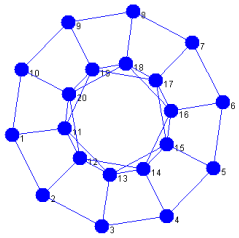
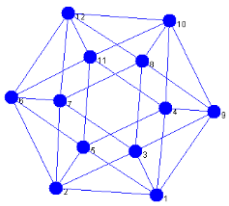
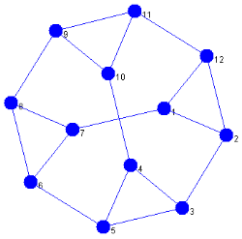
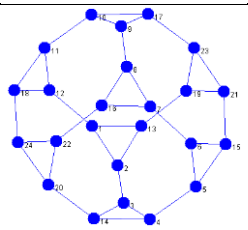
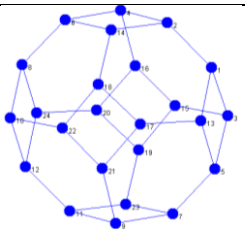
En cuanto a la aplicación móvil, se pensó en usar una versión más ligera del algoritmo debido a que este tipo de dispositivos presentan mucho menor rendimiento que el resto de computadoras. La modificación que se usa en la aplicación móvil, respecto a las otras dos aplicaciones, es que esta solo cuenta la cantidad de conjuntos independientes de un grafo, por lo tanto se omite el muestreo y el conteo por tamaños de los CI, con esto se reduce la velocidad en la ejecución, la cual logró superar a la versión web.

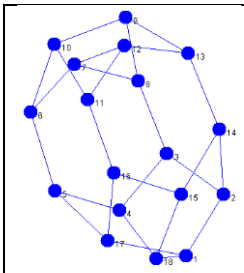
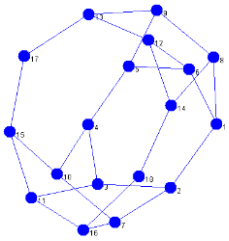
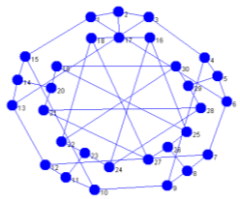
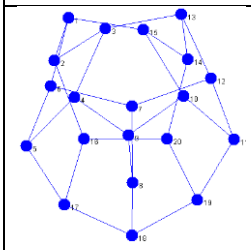
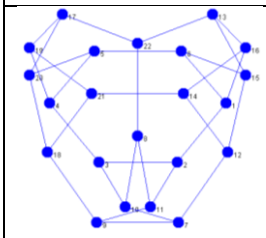
A continuación, se muestran algunos de los grafos que se probaron en la misma computadora con la aplicación de escritorio y la aplicación web, añadiendo también los resultados de tiempo de la versión móvil con su algoritmo ligero.

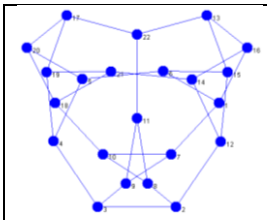
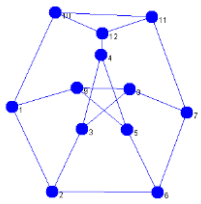
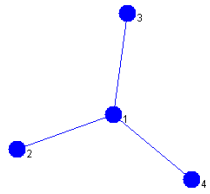
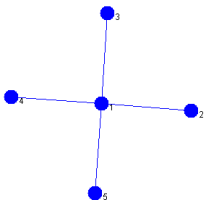
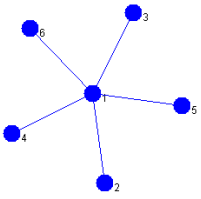
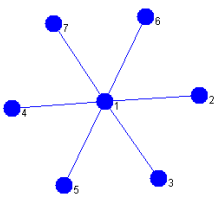
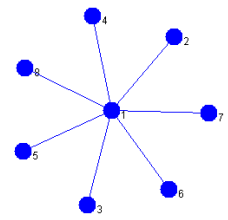
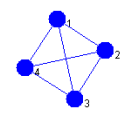
Dibujo	Nombre	Cantidad de CI	Densidad	Tiempo de ejecución aplicación de escritorio	Tiempo de ejecución web	Tiempo de ejecución móvil (versión ligera)
	K1-K25	2-26	100.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K2,3	11 Tamaño 0=1 Tamaño 1=5 Tamaño 2=4 Tamaño 3=1	60.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K3,3	15 Tamaño 0=1 Tamaño 1=6 Tamaño 2=6 Tamaño 3=2	60.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo

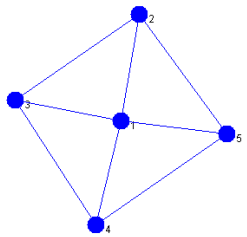
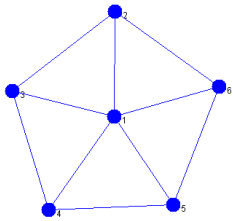
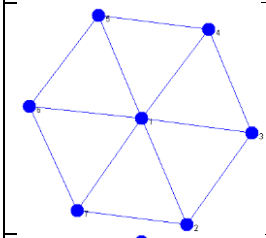
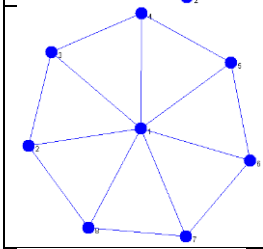
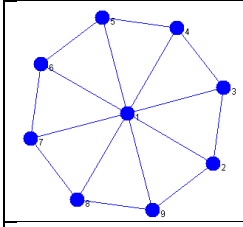
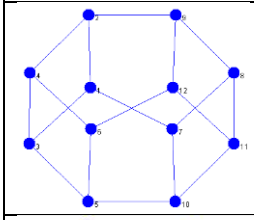
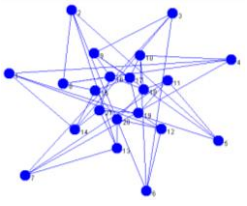
	K2,4	19 Tamaño 0=1 Tamaño 1=6 Tamaño 2=7 Tamaño 3=4 Tamaño 4=1	53.33%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K3,4	23 Tamaño 0=1 Tamaño 1=6 Tamaño 2=7 Tamaño 3=4 Tamaño 4=1	57.14%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K2,5	35 Tamaño 0=1 Tamaño 1=7 Tamaño 2=11 Tamaño 3=10 Tamaño 4=5 Tamaño 5=1	47.61%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K3,5	39 Tamaño 0=1 Tamaño 1=8 Tamaño 2=13 Tamaño 3=11 Tamaño 4=5 Tamaño 5=1	53.57%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K3,4	23 Tamaño 0=1 Tamaño 1=7 Tamaño 2=9 Tamaño 3=5 Tamaño 4=1	57.14%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K4,4	31 Tamaño 0=1 Tamaño 1=8 Tamaño 2=12 Tamaño 3=8 Tamaño 4=2	57.14%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K4,5	47 Tamaño 0=1 Tamaño 1=9 Tamaño 2=16 Tamaño 3=14 Tamaño 4=6 Tamaño 5=1	55.55%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K3,6	71 Tamaño 0=1 Tamaño 1=9 Tamaño 2=18 Tamaño 3=21 Tamaño 4=15 Tamaño 5=6 Tamaño 6=1	50.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	K4,6	79 Tamaño 0=1 Tamaño 1=10 Tamaño 2=21 Tamaño 3=24 Tamaño 4=16 Tamaño 5=6 Tamaño 6=1	53.33%	<1 milisegundo	<1 milisegundo	<1 milisegundo

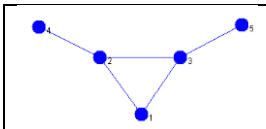
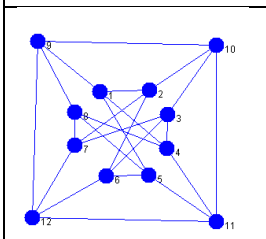
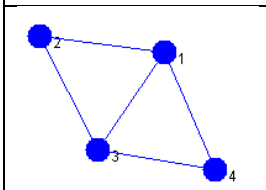
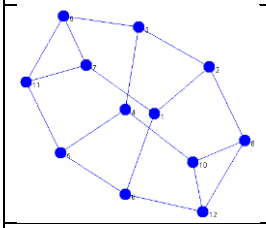
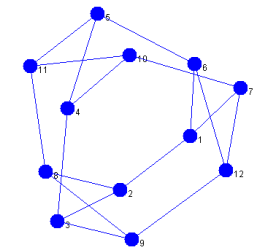
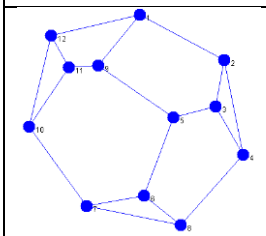
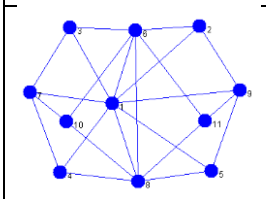
	F4	82 Tamaño 0=1 Tamaño 1=9 Tamaño 2=24 Tamaño 3=32 Tamaño 4=16	33.33%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	F8	6562 Tamaño 0=1 Tamaño 1=17 Tamaño 2=112 Tamaño 3=448 Tamaño 4=1120 Tamaño 5=1792 Tamaño 6=1792 Tamaño 7=1024 Tamaño 8=256	17.64%	1.11 segundos	0.18 segundos	0.8 segundos
	20-fullerene	5828 Tamaño 0=1 Tamaño 1=20 Tamaño 2=160 Tamaño 3=660 Tamaño 4=1510 Tamaño 5=1912 Tamaño 6=1240 Tamaño 7=320 Tamaño 8=5	15.78%	447 ms	182 ms	16 ms
	24-fullerene	33327 Tamaño 0=1 Tamaño 1=24 Tamaño 2=240 Tamaño 3=1304 Tamaño 4=4212 Tamaño 5=8316 Tamaño 6=9918 Tamaño 7=6756 Tamaño 8=2292 Tamaño 9=264	13.04%	18.31 segundos	1.18 segundos	66 ms
	26-fullerene	79698 Tamaño 0=1 Tamaño 1=26 Tamaño 2=286 Tamaño 3=1742 Tamaño 4=6461 Tamaño 5=15120 Tamaño 6=22361 Tamaño 7=20298 Tamaño 8=10523 Tamaño 9=2660 Tamaño 10=218 Tamaño 11=2	12.00%	107.85 segundos	3.12 segundos	95 ms
	Cubo	35 Tamaño 0=1 Tamaño 1=8 Tamaño 2=16 Tamaño 3=8 Tamaño 4=2	42.85%	<1 milisegundo	<1 milisegundo	<1 milisegundo

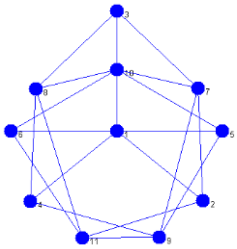
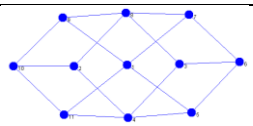
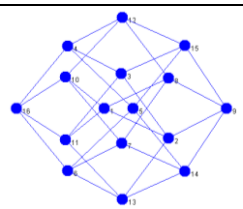
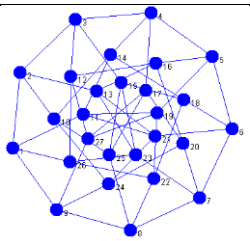
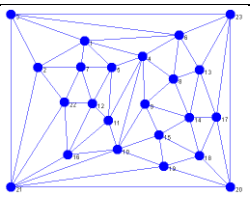
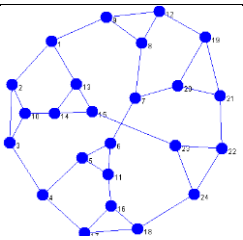
	Octaedro	10 Tamaño 0=1 Tamaño 1=6 Tamaño 2=3	80.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Dodecaedro	3063 Tamaño 0=1 Tamaño 1=20 Tamaño 2=150 Tamaño 3=540 Tamaño 4=995 Tamaño 5=922 Tamaño 6=385 Tamaño 7=50	21.05%	516 ms	104 ms	4 ms
	Icosaedro	69 Tamaño 0=1 Tamaño 1=12 Tamaño 2=36 Tamaño 3=20	45.45%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Tetraedro truncado	163 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=72 Tamaño 4=30	27.27%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Cubo truncado	26441 Tamaño 0=1 Tamaño 1=24 Tamaño 2=240 Tamaño 3=1296 Tamaño 4=4092 Tamaño 5=7632 Tamaño 6=8056 Tamaño 7=4272 Tamaño 8=828	13.04%	10.80 segundos	0.95 segundos	0.05 segundos
	Octaedro truncado	37076 Tamaño 0=1 Tamaño 1=24 Tamaño 2=240 Tamaño 3=1304 Tamaño 4=4218 Tamaño 5=8400 Tamaño 6=10356 Tamaño 7=7824 Tamaño 8=3555 Tamaño 9=960 Tamaño 10=168 Tamaño 11=24 Tamaño 12=2	13.04%	21.28 segundos	1.42 segundos	0.06 segundos

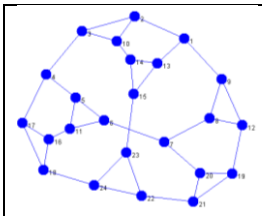
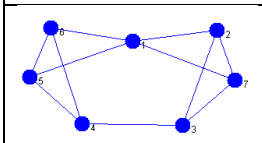
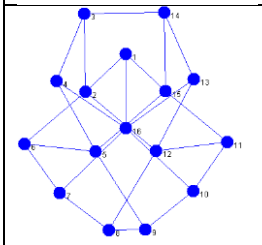
	Blanuša snark (primero)	2495 Tamaño 0=1 Tamaño 1=18 Tamaño 2=126 Tamaño 3=438 Tamaño 4=801 Tamaño 5=748 Tamaño 6=316 Tamaño 7=46 Tamaño 8=1	17.64%	408 ms	72 ms	4 ms
	Blanuša snark (segundo)	2766 Tamaño 0=1 Tamaño 1=18 Tamaño 2=127 Tamaño 3=450 Tamaño 4=853 Tamaño 5=848 Tamaño 6=399 Tamaño 7=68 Tamaño 8=2	16.99%	250 ms	74 ms	3 ms
	Doble estrella snark	473240 Tamaño 0=1 Tamaño 1=30 Tamaño 2=390 Tamaño 3=2890 Tamaño 4=13515 Tamaño 5=41736 Tamaño 6=86620 Tamaño 7=120790 Tamaño 8=111535 Tamaño 9=66240 Tamaño 10=24098 Tamaño 11=4920 Tamaño 12=465 Tamaño 13=10	10.34%	98 minutos	20.85 segundos	0.54 segundos
	Flor snark	6114 Tamaño 0=1 Tamaño 1=20 Tamaño 2=160 Tamaño 3=660 Tamaño 4=1510 Tamaño 5=1923 Tamaño 6=1310 Tamaño 7=450 Tamaño 8=75 Tamaño 9=5	15.78%	1.16 segundos	0.19 segundos	0.02 segundos
	Loupekine snark	14277 Tamaño 0=1 Tamaño 1=22 Tamaño 2=198 Tamaño 3=946 Tamaño 4=2607 Tamaño 5=4218 Tamaño 6=3901 Tamaño 7=1911 Tamaño 8=429 Tamaño 9=42 Tamaño 10=2	14.28%	4.95 segundos	0.50 segundos	0.01 segundos

	Loupekine snark (segundo)	14207 Tamaño 0=1 Tamaño 1=22 Tamaño 2=198 Tamaño 3=946 Tamaño 4=2607 Tamaño 5=4218 Tamaño 6=3899 Tamaño 7=1897 Tamaño 8=399 Tamaño 9=20	14.28%	4.34 segundos	0.50 segundos	0.02 segundos
	Grafo Tietze	178 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=75 Tamaño 4=39 Tamaño 5=3	27.27%	78 ms	4 ms	1 ms
	S3	9 Tamaño 0=1 Tamaño 1=4 Tamaño 2=3 Tamaño 3=1	50.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	S4	17 Tamaño 0=1 Tamaño 1=5 Tamaño 2=6 Tamaño 3=4 Tamaño 4=1	40.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	S5	33 Tamaño 0=1 Tamaño 1=6 Tamaño 2=10 Tamaño 3=10 Tamaño 4=5 Tamaño 5=1	33.33%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	S6	65 Tamaño 0=1 Tamaño 1=7 Tamaño 2=15 Tamaño 3=20 Tamaño 4=15 Tamaño 5=6 Tamaño 6=1	28.57%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	S7	129 Tamaño 0=1 Tamaño 1=8 Tamaño 2=21 Tamaño 3=35 Tamaño 4=35 Tamaño 5=21 Tamaño 6=7 Tamaño 7=1	25.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	W4	5 Tamaño 0=1 Tamaño 1=4	100.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo

	W5	8 Tamaño 0=1 Tamaño 1=5 Tamaño 2=2	80.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	W6	12 Tamaño 0=1 Tamaño 1=6 Tamaño 2=5	66.66%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	W7	19 Tamaño 0=1 Tamaño 1=7 Tamaño 2=9 Tamaño 3=2	57.14%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	W8	30 Tamaño 0=1 Tamaño 1=8 Tamaño 2=14 Tamaño 3=7	50.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	W9	48 Tamaño 0=1 Tamaño 1=9 Tamaño 2=20 Tamaño 3=16 Tamaño 4=2	44.44%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Cubo de Bidiakis	191 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=76 Tamaño 4=46 Tamaño 5=8	27.27%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Brinkmann	4363 Tamaño 0=1 Tamaño 1=21 Tamaño 2=168 Tamaño 3=658 Tamaño 4=1344 Tamaño 5=1400 Tamaño 6=665 Tamaño 7=106	20.00%	687 ms	143 ms	15 ms

	Grafo Toro	12 Tamaño 0=1 Tamaño 1=5 Tamaño 2=5 Tamaño 3=1	50.00%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Chvátal	127 Tamaño 0=1 Tamaño 1=12 Tamaño 2=42 Tamaño 3=52 Tamaño 4=20	36.36%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo Diamante	6 Tamaño 0=1 Tamaño 1=4 Tamaño 2=1	83.33%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Dürer	171 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=74 Tamaño 4=36	27.27%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Franklin	196 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=76 Tamaño 4=45 Tamaño 5=12 Tamaño 6=2	27.27%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Frucht	169 Tamaño 0=1 Tamaño 1=12 Tamaño 2=48 Tamaño 3=73 Tamaño 4=34 Tamaño 5=1	27.27%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Goldner–Harary	103 Tamaño 0=1 Tamaño 1=11 Tamaño 2=32 Tamaño 3=35 Tamaño 4=17 Tamaño 5=6 Tamaño 6=1	41.81%	<1 milisegundo	<1 milisegundo	<1 milisegundo

	Grafo de Grötzsch	103 Tamaño 0=1 Tamaño 1=11 Tamaño 2=35 Tamaño 3=40 Tamaño 4=15 Tamaño 5=1	36.36%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Herschel	124 Tamaño 0=1 Tamaño 1=11 Tamaño 2=37 Tamaño 3=45 Tamaño 4=22 Tamaño 5=7 Tamaño 6=1	32.72%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Hoffman	747 Tamaño 0=1 Tamaño 1=16 Tamaño 2=88 Tamaño 3=208 Tamaño 4=228 Tamaño 5=132 Tamaño 6=56 Tamaño 7=16 Tamaño 8=2	26.66%	64 ms	19 ms	2 ms
	Grafo de Holt	47860 Tamaño 0=1 Tamaño 1=27 Tamaño 2=297 Tamaño 3=1737 Tamaño 4=5913 Tamaño 5=12042 Tamaño 6=14472 Tamaño 7=9693 Tamaño 8=3213 Tamaño 9=438 Tamaño 10=27	15.38%	47.08 segundos	2.10 segundos	0.08 segundos
	Grafo de Kittell	2985 Tamaño 0=1 Tamaño 1=23 Tamaño 2=191 Tamaño 3=707 Tamaño 4=1167 Tamaño 5=757 Tamaño 6=137 Tamaño 7=2	24.50%	234 ms	97 ms	6 ms
	Grafo de Markström	27266 Tamaño 0=1 Tamaño 1=24 Tamaño 2=240 Tamaño 3=1297 Tamaño 4=4107 Tamaño 5=7716 Tamaño 6=8276 Tamaño 7=4554 Tamaño 8=1002 Tamaño 9=49	13.04%	11.21 segundos	0.94 segundos	0.03 segundos

	Grafo de McGee	34305 Tamaño 0=1 Tamaño 1=24 Tamaño 2=240 Tamaño 3=1304 Tamaño 4=4212 Tamaño 5=8328 Tamaño 6=10024 Tamaño 7=7072 Tamaño 8=2668 Tamaño 9=424 Tamaño 10=8	13.04%	17.10 segundos	0.94 segundos	0.03 segundos
	Moser spindle	18 Tamaño 0=1 Tamaño 1=7 Tamaño 2=10	52.38%	<1 milisegundo	<1 milisegundo	<1 milisegundo
	Grafo de Sousselier	888 Tamaño 0=1 Tamaño 1=16 Tamaño 2=93 Tamaño 3=249 Tamaño 4=319 Tamaño 5=179 Tamaño 6=31	22.50%	40 ms	27 ms	3 ms

4.1 Tendencia de los resultados a la campana gaussiana

En el resultado anterior, se puede ver como la cantidad de conjuntos independientes se eleva, y disminuye posteriormente con forme avanza el tamaño de estos. Esto pasa con todos los grafos, todos van elevando su cantidad hasta donde alcanzan un tamaño intermedio, y después disminuyen. Usualmente, la segunda mitad disminuye en menor cantidad de lo que la primera eleva sus resultados. Este tipo de resultados tiene cierta similitud con la campana de gauss.

La campana de gauss es una función que se define de la siguiente forma

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

Donde a, b, c son constantes cualesquiera.

La constante a define qué altura alcanzará la función o cuál va a ser su punto máximo.

La constante b mueve la función ya sea a la izquierda o a la derecha según su valor.

La constante c modifica la curvatura de la gráfica haciéndola más redonda con forme aumenta su valor.

El hecho de estudiar esta función, dentro del cálculo de los conjuntos independientes, es debido a que esta puede aproximar el resultado. Dentro de las pruebas que se hicieron se puede modificar la función de otra manera:

$$f(x) = ae^{-\frac{(x-b)^2}{b}}$$

Donde a es la cantidad máxima de CI para algún tamaño y b es el tamaño en donde se encontraron más conjuntos independientes.

Al integrar esta función desde 0 hasta $2b$ se puede aproximar la cantidad total de conjuntos independientes que se puedan encontrar. El resultado de integrar este valor usualmente está por arriba de la cantidad total de CI, sin embargo es una buena forma de aproximar el resultado si se conocen los valores de a y b.

Sin embargo, la forma en que funciona el algoritmo presentado, no será la ideal para aproximar mediante la integral de esta función, debido a que no cuenta secuencialmente los CI por tamaño, sino que durante el proceso de búsqueda

encuentran CI de diferentes tamaños por lo que la aproximación sería variable e iría en aumento hasta que haya terminado la búsqueda, lo cual solo agregaría cálculo extra a el algoritmo y aportaría datos no muy útiles durante el proceso.

Por otro lado, esta forma de aproximar los resultados, a pesar de no ser la más adecuada para el algoritmo presentado, puede ser útil en otros casos donde se pueda resolver en algún punto los dos datos que representan las constante a y b de la función.

Estas constantes podrían ser encontradas a la mitad del procesamiento de un algoritmo que busque secuencialmente por tamaños los CI, y de esta forma tener una aproximación del resultado antes de finalizar o continuar la búsqueda.

Otra forma que se intentó para la aproximación, fue con diferentes valores de c en la función original de gauss. Se probó con $c = 1$ cuya integral igualmente de 0 a $2b$ resulta en la mayoría de los casos por debajo del total de conjuntos encontrados.

También se hicieron pruebas integrando la primera mitad (de 0 a b) con la función $f(x) = ae^{-\frac{(x-b)^2}{2(1)^2}}$ y la segunda mitad (de b a $2b$) con $f(x) = ae^{-\frac{(x-b)^2}{b}}$.

El resultado de esto queda igualmente por debajo de la mayoría de conjuntos de los grafos.

La suma de las integrales de estas dos funciones se consideró, debido a que una vez que se alcanza para algún tamaño de conjuntos la máxima cantidad, los tamaños siguientes siempre son mayores o iguales a los anteriores.

La relación de los resultados con la campana de gauss, abre puertas para la aplicación o uso de los conjuntos independientes sobre campos como la estadística, en donde suele ser más común aunque también se puede encontrar tanto a esta función como a los grafos en química y en geometría y varías otras áreas de matemática.

4.2 Clases de grafos y tendencia de aumento de CI

Existen algunos grafos que por su naturaleza, es posible generar reglas matemáticas, ya sea para saber tanto la cantidad de conjuntos independientes como la cantidad de éstos para todos los tamaños. A continuación se muestran algunas clases de grafos que tienen alguna tendencia para poder predecir su cantidad de conjuntos independientes:

4.2.1 Grafos Kx

Iniciando por el mejor de los casos, los grafos completos son los más sencillos de predecir ya que estos solo tendrían $x + 1$ conjuntos independientes, x conjuntos de tamaño 1 más el vacío. Esto es debido a que por estar todos los nodos conectados entre si, no se pueden generar CI de mayor tamaño.

4.2.2 Grafos disconexos

Es poco usual encontrar estos tipos de grafos en problemas debido a que resultan poco útiles al no tener aristas, sin embargo, al calcular su cantidad de conjuntos se pueden averiguar ciertas reglas.

Una de estas reglas es:

- Un grafo disconexo con n nodos tendrá un total de 2^n conjuntos independientes.

De igual forma, la manera en que están distribuidos los tamaños de los conjuntos independientes de estos grafos se puede encontrar en el nivel n de la pirámide de pascal.

Por ejemplo un grafo no conexo con 10 nodos tiene los conjuntos distribuidos de la siguiente forma:

Tamaño 0=1
Tamaño 1=10
Tamaño 2=45
Tamaño 3=120
Tamaño 4=210
Tamaño 5=252
Tamaño 6=210
Tamaño 7=120
Tamaño 8=45
Tamaño 9=10
Tamaño 10=1

Esto corresponde con el nivel decimo de la pirámide:

0											1									
1										1		1								
2									1		2		1							
3								1		3		3		1						
4							1		4		6		4		1					
5						1		5		10		10		5		1				
6					1		6		15		20		15		6		1			
7				1		7		21		35		35		21		7		1		
8			1		8		28		56		70		56		28		8		1	
9		1		9		36		84		126		126		84		36		9		1
10	1		10		45		120		210		252		210		120		45		10		1

Con el algoritmo, encontrar estos valores para grafos no conexos, con número de nodos superior a diez, empieza a tardar de forma notoria, sin embargo para encontrar estos valores mediante la pirámide es mucho menos tardado, e incluso la complejidad de encontrar los valores para algún nivel de la pirámide es menor al que presenta el algoritmo, por otro lado este método solo se reduce a grafos disconexos lo que no deja mucha amplitud para sus uso.

4.2.3 Grafos Kx,x

Los grafos Kx,y se caracterizan por ser bipartitos, es decir que se pueden dividir en dos conjuntos independientes. De estos grafos para aquellos que se pueden dividir en dos conjuntos del mismo tamaño los nombraremos grafos Kx,x y estos presentan características que pueden llevar a calcular con mayor sencillez la cantidad de conjuntos independientes, por tamaño que existen en cualquiera de estos grafos.

Los grafos Kx,x tienen un cálculo muy similar a los grafos disconexos. Por ejemplo para calcular la cantidad total de CI para alguno de estos grafos la fórmula es:

$$2^{\frac{n}{2}+1} - 1$$

Y para encontrar su distribución de cantidad respecto al tamaño de estas, se puede encontrar también usando la pirámide de pascal pero usando 2 en lugar de 1 en las diagonales como se muestra a continuación:

Distribución para grafos: K5,5 K6,6 K7,7

K5,5	K6,6	K7,7
Cantidad de CI: 63 Tamaño 0=1 Tamaño 1=10 Tamaño 2=20 Tamaño 3=20 Tamaño 4=10 Tamaño 5=2	Cantidad de CI: 127 Tamaño 0=1 Tamaño 1=12 Tamaño 2=30 Tamaño 3=40 Tamaño 4=30 Tamaño 5=12 Tamaño 6=2	Cantidad de CI: 255 Tamaño 0=1 Tamaño 1=14 Tamaño 2=42 Tamaño 3=70 Tamaño 4=70 Tamaño 5=42 Tamaño 6=14 Tamaño 7=2

Pirámide de pascal con diagonales de 2:

0								2												
1							2		2											
2						2		4		2										
3					2		6		6		2									
4				2		8		12		8		2								
5			2		10		20		20		10		2							
6		2		12		30		40		30		12		2						
7	2		14		42		70		70		42		14		2					
8		2		16		56		112		140		112		56		16		2		
9		2		18		72		168		252		252		168		72		18		2
10	2		20		90		240		420		504		420		240		90		20		2

La distribución respecto a los tamaños de los conjuntos, solo varia en el primer valor que es uno en lugar de dos y esto corresponde al conjunto vacío.

4.2.4 Grafos Kx, y

Otro tipo de grafos bipartitos Kx, y presentan un mayor problema para su cálculo. Para $K2, y$ se puede calcular la cantidad total de conjuntos con la siguiente formula:

$$\#CI = 3 + 2^{n-2}$$

Y consecutivamente se puede calcular la cantidad de conjuntos para $K3, y$ con la formula:

$$\#CI = 7 + 2^{n-3}$$

Con estas dos fórmulas iniciales podemos deducir que para cualquier grafo del tipo Kx, y se puede calcular la cantidad total de conjuntos independientes con:

$$\#CI = (2^x - 1) + 2^{n-x}$$

Debido a que $y = n - x$ para grafos bipartitos Kx, y, la formula se puede reducir a:

$$\#CI = 2^x + 2^y - 1$$

De forma similar, para encontrar la distribución entre sus conjuntos se puede usar como herramienta la pirámide de pascal. Para cada grafo Kx,y con valores diferentes de x y y necesitará crear su propia pirámide de distribución, la cual se puede formar mediante sumas de los valores. Estas pirámides tomarán como referencia el nivel x de la pirámide de los grafos Kx,x y para encontrar los niveles inferiores, se sumarán los valores de la pirámide original a partir también del nivel x. A continuación se muestra el ejemplo para los grafos $K2,y$:

El nivel x para $K2,y$ es:

1 4 2

Para calcular el siguiente nivel se agregan los valores del nivel x de la pirámide original:

1	1	4	2	2	1

De esto se puede calcular el siguiente nivel:

2		1	1	4	2	2	1	
3	1		5		4		1	

Los valores del nuevo nivel se calculan con un valor de la pirámide original y con otro valor del nivel superior. El siguiente nivel de la tabla se muestra a continuación:

2			1	1	4	2	2	1		
3		1	1	5	3	4	3	1	1	
4	1		6		7		4		1	

Los valores sin color son la pirámide que se genera, los valores azules corresponden a la pirámide original, el morado, rojo y verde corresponden a la suma que se hace para encontrar cada nuevo valor de la pirámide.

El ejemplo anterior representa la pirámide formada para un grafo $K2,y$, para otro grafo como $K4,y$ se deberán tomar los niveles 4 de las pirámides para generar la suma $\{1,8,12,8,2\}$ y $\{1,4,6,4,1\}$. Esta pirámide quedaría de la siguiente forma:

4							1	1	8	4	12	6	8	4	2	1							
5						1	1	9	5	16	10	14	10	6	5	1	1						
6					1	1	10	6	21	15	24	20	16	15	6	6	1	1					
7				1	1	11	7	27	21	39	35	36	35	21	21	7	7	1	1				
8			1	1	12	8	34	28	60	56	71	70	56	56	28	28	8	8	1	1			
9		1	1	13	9	42	36	88	84	127	126	126	126	84	84	36	36	9	9	1	1		
10	1	1	14	10	51	45	124	120	211	210	252	252	210	210	120	120	45	45	10	10	1	1	

Por lo que para un grafo $K4{,}10$ la distribución de las cantidades de conjuntos por tamaño corresponde al nivel 10 de la tabla de $K4, y$ la cual es:

Tamaño 0=1
Tamaño 1=14
Tamaño 2=51
Tamaño 3=124
Tamaño 4=211
Tamaño 5=252
Tamaño 6=210
Tamaño 7=120
Tamaño 8=45
Tamaño 9=10
Tamaño 10=1

4.2.5 Grafos Fx

También conocidos como grafos de la amistad, se caracterizan por la unión de grafos $K3$ sobre un solo vértice común. Estos grafos constan de $2x + 1$ vértices y $3x$ aristas. Al igual que los anteriores grafos presentados, se puede calcular su distribución y tamaño mediante fórmulas, o con ayuda de alguna modificación del triángulo de pascal. La distribución que siguen estos grafos está dada de la siguiente forma.

Inicialmente el triángulo se iniciará con 1 y 3 que corresponden a la distribución del grafo $F1$ o $K3$. El segundo nivel se calcula multiplicando por dos el número izquierdo y sumándolo con el derecho. Con esto basta para formar el triángulo de distribución de los grafos Fx, sin embargo hay una excepción para el lugar que corresponde a los conjuntos con tamaño tres en donde solo a los que

corresponden a este tamaño se les restará 2 al resultado de la anterior fórmula. Con esto la tabla de distribución de los grafos Fx es la siguiente:

2	1	3										
3	1	5	4									
4	1	7	12	8								
5	1	9	24	32	16							
6	1	11	40	80	80	32						
7	1	13	60	160	240	192	64					
8	1	15	84	280	560	672	448	128				
9	1	17	112	448	1120	1792	1792	1024	256			
10	1	19	144	672	2016	4032	5376	4608	2304	512		
11	1	21	180	960	3360	8064	13440	15360	11520	5120	1024	
12	1	23	220	1320	5280	14784	29568	42240	42240	28160	11264	2048

Como se puede ver cada nivel del triángulo sencillamente calculable tomando el doble del término superior izquierdo más el término superior.

$32 * 2$	16	$64 + 16 = 80$
	80	

4.2.6 Grafos Sx

Estos al igual que los grafos no conexos, son fáciles de calcular. Estos grafos se caracterizan por que existe un solo nodo al que todos están conectados, el resto de nodos no tendrán adyacencia entre sí. Por esto para el cálculo de los CI basta con modificar la ecuación de los grafos no conexos.

La fórmula para calcular los CI de este tipo de grafos es:

$$2^x - 1$$

Y para su distribución basta tomar el nivel x del triángulo de pascal con la modificación de que al segundo término (el que corresponde a los CI de tamaño igual a uno) se le sume uno. Esta adicción no se toma en cuenta para el cálculo de los niveles posteriores como se hizo con los grafos Fx.

Conclusiones

Se han presentado actualmente trabajos que intentan resolver el problema del conjunto independiente maximal, muchos de ellos se enfocan a cierta clase de grafos, con lo cual se pueden encontrar tiempos de ejecución bajos, al igual que complejidades aceptables, dado que en general se trata de un problema NP. No obstante al hablar de algoritmos que se enfocan a solo un tipo o clase de grafos es de suponer que solo se puede trabajar sobre áreas reducidas en el campo.

La propuesta de crear un algoritmo que sea capaz de encontrar todos los conjuntos independientes de un grafo, sin importar su estructura, se da con la razón de poder explorar en toda su extensión en el campo.

Este algoritmo presenta tanto ventajas como desventajas frente a los demás. La ventaja más clara es la ya mencionada de funcionar con cualquier tipo de grafo, pero además de esta se agregan otras como, que es capaz de hacer el muestreo de todos los CI de forma ordenada y por tamaños, todo dentro de una sola ejecución, lo cual muchos otros métodos por su naturaleza no son capaces de lograr.

Como desventajas del algoritmo se tienen que frente a los demás es tardado y tiene una complejidad alta, cosa que es de esperar para los algoritmos de búsquedas exhaustivas que suelen presentar complejidades exponenciales.

Durante el desarrollo se observó que el trabajo se puede extender más allá del problema principal, y resolver también otro problema que se ha presentado comúnmente en las clases NP. Se encontró entonces, que con ligeras modificaciones al momento de leer o de ejecutar el método principal, se pueden encontrar los cliques igualmente de cualquier grafo.

Una característica, entre los grafos más comunes que se han estudiado es que presentan densidades bajas, la mayoría de los grafos con densidades arriba de 0.5 suelen ser muy pequeños. Esto da como consecuencia de que desventajosamente, la búsqueda de los conjuntos independientes sea más larga a partir de algunos grafos. Arriba de 15 nodos empieza a presentar lentitud notable. Por otro lado, y debido a la estrecha relación entre cliques y CI, se pueden calcular con lo anterior mencionado, los cliques de varios grafos conocidos bastante extensos sin, de esta forma, tardar demasiado en la ejecución. El factor de tener esta otra alternativa, dependerá de la situación a la cual se requiere aplicar dichas búsquedas.

Las aplicaciones se desarrollaron para tres plataformas bajo dos lenguajes distintos, la aplicación de escritorio y la móvil se desarrolló en JAVA y la aplicación web en PHP. La aplicación de escritorio presentó tiempos según lo esperado bajo distintos procesadores, sin embargo la aplicación web desarrollada en PHP presentó tiempos muy por debajo de lo esperado, dando como mayor diferencia un tiempo de ejecución de aproximadamente 27 segundos en comparación a 98 minutos de la aplicación de escritorio para el grafo doble estrella snark (30 nodos, densidad 0.1), que es el grafo más grande y computacionalmente difícil que se probó.

Esta prueba, siendo que se generó bajo la misma computadora, abre una puerta para hacer un estudio sobre la velocidad de los procesos en los lenguajes, debido a que a pesar de que la máquina virtual de java genera retrasos en comparación a otros lenguajes más nativos, el tiempo de diferencia es bastante y esto se puede deber ya sea operaciones básicas como sumas o restas, o más probablemente a concatenaciones de Strings y listas, y accesos a memoria que son usados varias veces en el proceso.

La aplicación móvil, al estar usando una versión ligera del algoritmo, en la cual no incluye el muestreo de los resultados, también resultó bastante favorable superando por mucho los tiempos del algoritmo original.

En las aplicaciones, también se destacó el uso de un método para dibujar los grafos, el cual se basa en leyes físicas de atracción y repulsión. Este método no fue planteado en los objetivos principales, pero es una herramienta tan útil como la calculadora debido que da una imagen natural de cómo es el grafo que se está representando, el mismo método puede usarse en muchas otras aplicaciones que estudien grafos para así poder visualizarlo. Según lo investigado [30] esté método presenta una complejidad $O(n^2)$ debido a que se tiene que calcular la dinámica de cada nodo en cada iteración. Esté método, al ser usado en dispositivos móviles, que son los dispositivos con menor rendimiento, logró tener un desempeño aceptable incluso con los grafos más grandes del repositorio.

Teniendo ya las aplicaciones, se procedió a crear un repositorio el cual fuera un complemento del trabajo para servir de comparación en cuanto a tiempo y resultados de los actuales y futuros algoritmos que se desarrollen en el área. Este cuenta con ochenta grafos de diferentes tipos y clases en donde se destaca la densidad de éstos, el tiempo de ejecución en la aplicación de escritorio y la web así como también la distribución que éstos tienen en sus conjuntos independientes.

Al obtener los resultados de varios tipos de grafos, se observaron ciertas tendencias que podrían servir en trabajos posteriores, como una forma de aproximar la cantidad de conjuntos independientes mediante fórmulas de la campana gaussiana. Estas fórmulas no son del todo fiables debido a que solo aproximan la cantidad de conjuntos independientes, sin embargo también se propusieron otras formas de encontrar la cantidad de conjuntos independientes y además su distribución mediante el triángulo de pascal, en donde usando algunas modificaciones con base en el método de crecimiento de la pirámide, es posible calcular para algunas clases de grafos, con lo cual se puede decir que cualquier clase de grafos en donde crezcan en tamaño de una forma simétrica constante o periódica, es posible encontrar una regla que sea capaz de resolver el problema de los conjuntos independientes para solo esa clase. Las resoluciones mediante el triángulo de pascal son bastante sencillas; una vez se conoce la regla a seguir. Estos métodos puedes ser útiles en topologías de redes entre otras cosas aplicables.

El trabajo realizado durante toda la investigación logró abarcar más allá de los objetivos propuestos, dando como resultado aplicaciones con varias herramientas no solo para los el área de los conjuntos independientes, sino también para otras áreas de grafos, de igual forma se lograron encontrar otros métodos capaces de resolver este problema, una vez que ya se habían desarrollado algunas de estas aplicaciones, con lo cual da a notar la importancia de trabajos de este tipo al facilitar a los investigadores, herramientas que encapsulen la información para de esta manera dar pasos más grandes en el descubrimiento y la innovación de procesos.

Perspectivas

Como trabajo futuro sobre la investigación realizada se puede proceder sobre el algoritmo agregando el reconocimiento de los conjuntos independientes maximales. De igual forma, se puede modificar el algoritmo propuesto para hacer la búsqueda de los cliques de un grafo. Como se menciona anteriormente, para esto es necesario invertir la búsqueda para que lo que se obtenga como resultado sean subconjuntos de nodos que sean grafos completos.

De igual forma, en continuidad a lo avanzado con respecto a las clases de grafos y la pirámide de pascal, aún se pueden encontrar fórmulas útiles para otras clases de grafos. La complejidad de encontrar una fórmula para alguna clase va de la mano en la forma en que crecen los grafos de dicha clase, es decir cuántas aristas y cuantos nodos se agregan para el siguiente elemento y en qué posiciones. Es posible que al igual que los grafos Kx,y se tengan que usar más de una pirámide para encontrar el resultado.

Bibliografía

[1] Sabino Miranda Jiménez, 2013. Modelo para la generación automática de resúmenes abstractos basado en grafos conceptuales. Tesis doctoral. México D.F. Instituto Politécnico Nacional. 121 p.

[2] Ke XU. Benchmarks with Hidden Optimum Solutions for Graph Problems [en línea]. Beijing, China. [Consulta 23 de enero de 2015]. Disponible en: http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

[3] Apuntes de Grafos [en línea]. Valladolid, España. [Consulta 24 de febrero de 2015]. Disponible en: http://www.infor.uva.es/~cvaca/asigs/estr0506apg.pdf

[4] Optimización con recocido simulado para el problema de conjunto independiente. [En línea]. Junio 1998 [Consulta: 10 de enero de 2015]. Disponible en: http://www.azc.uam.mx/publicaciones/enlinea2/3.htm

[5] Graph Theory, MathWorld [En línea]. [Consulta: 16 de enero de 2015]. Disponible en: http://mathworld.wolfram.com/topics/GraphTheory.html

[6] Rosa Guerequeta; Antonio Vallecillo, *Técnicas de Diseño de Algoritmos*. 2da edición. Servicio de Publicaciones de la Universidad de Málaga. 1998. ISBN: 84-7496-666-3.

[7] Reinaldo Giudici Espinoza, Ángeles Bris Lluch. *Introducción a la teoría de grafos*. Ediciones de la Universidad de Simón Bolívar, 1997. ISBN: 980-237-154-8.

[8] Juan Bernardo Vázquez Gómez. *Análisis y diseño de algoritmos*. ISBN: 978-607-733-053-0

[9] Reinhard Diestel. *Graph Theory*. 4ª edición. Julio 2010. ISBN: 978-3-642-14278-9.

[10] Ivan Kuckir, Graph drawer, Imágenes de grafos tomadas de http://g.ivank.net/.

[11] Javier Galve et. al. *Algorítmica diseño y análisis de algoritmos funcionales e imperativos*. Addison-Wesley Iberoamérica, S. A. 1993. ISBN: 0-201-62350-1.

[12] Russell L. Shackelford. *Computing and algorithms*. Addison-Wesley. 1998. ISBN: 0-201-31451-7.

[13] Baase, Sara; Van Gelder, Allen. *Algoritmos computacionales*. 3ra edición. Addison-Wesley. 2002. ISBN: 970-26-0142-8.

[14] Guillermo De Ita, Pedro Bello, Diseño de Algoritmos Eficientes para el Conteo de Conjuntos Independientes en una Red, Benemérita Universidad Autónoma de Puebla, Facultad de Computación.

[15] Rafael López Bracho, María Paula Ortuño Sánchez, Un algoritmo paralelo para el problema del conjunto independiente, Revista de Matemática: Teoría y Aplicaciones 2000 7(1-2) : 125–134.

[16] Mingyu Xiao, Hiroshi Nagamochi, An exact algorithm for maximum independent set in degree-5 graphs, Discrete Applied Mathematics, 2014.

[17] Nicholas Nash, Sylvain Lelait, David Gregg, Efficiently Implementing Maximum Independent Set Algorithms on Circle Graphs, ACM Journal of Experimental Algorithmics, Vol. 13, Article 1.9, December 2008

[18] Alexander Chatzigeorgiou, Nikolaos Tsantalis, George Stephanides, Application of Graph Theory to OO Software Engineering, Department of Applied Informatics, University of Macedonia.
[19] M. E. Stickney, An Application of Graph Theory to Software test Data Selection, Huges Aircraft Company.
[20] Henry K. DeWitt, Applications of the theory of random graphs to average algorithm performance analysis, ACM ’76 Proceedings of the 1979 annual conference, Pages 251-258.
[21] Alexandru T. Balaban, Applications of Graph Theory in Chemistry, Department of Organic Chemistry, Polytefhnic Institute. 76206 Bucharest, Roumania.
[22] John W. Essam, Michael E. Fisher, Some Basic Definition in Graph Theory, Reviews of Modern Physics Volume 42, April 1970.
[23] Greenhill Catherine, The Complexity of counting colorings and independent sets in sparse graphs and hypergraphs”, Computational Complexity, 1999.
[24] Vadhan Salil P., The complexity of Counting in Sparse, Regular, and Planar Graphs, SIAM Journal on Computing, Vol. 31, No. 2, (2001), 398-427.
[25] Miguel Jiménez M., Grafos para la Física Cuántica, Departamento de matemática aplicada. Sevilla, Diciembre 2010.
[26] Jerrold R. Griggs, Chales M. Grinstead, The number of maximal independent sets in a connected graph, Discrete Mathematics, Elsevier, 1988.
[27] White, D. R., & Reitz, K. P. (1983). Graph and semigroup homomorphisms on networks of relations. Social Networks, 5(2), 193-234.
[28] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" (PDF). Numerische Mathematik 1: 269–271.
[29] Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". Proceedings of the American Mathematical Society 7: 48–50.
[30] Kobourov, Stephen G. (2012), Spring Embedders and Force-Directed Graph Drawing Algorithms.

Agradecimientos

Agradezco el apoyo económico brindado para el desarrollo de esta tesis al cuerpo académico: Algoritmos Combinatorios y Aprendizaje (Facultad de Ciencias de la Computación - Benemérita Universidad Autónoma de Puebla)