Repositorio Digital Institucional
"José María Rosa"

Universidad Nacional de Lanús
Secretaría Académica
Dirección de Biblioteca y Servicios de Información Documental

Gerónimo Luciano Tondato

G-Matrix: propuesta de algoritmo para búsqueda de (sub) isomorfismos en grafos conexos. Estrategia de división del espacio de búsqueda y ejecución en paralelo

Trabajo Final Integrador presentado para la obtención del título de Licenciado en Sistemas

**Directores de Trabajo Final Integrador**

Darío Rodríguez, Hernán Amatriain

El presente documento integra el Repositorio Digital Institucional "José María Rosa" de la Biblioteca "Rodolfo Puiggrós" de la Universidad Nacional de Lanús (UNLa)
This document is part of the Institutional Digital Repository "José María Rosa"of the Library "Rodolfo Puiggrós" of the University National of Lanús (UNLa)

**Cita sugerida**

Tondato, Gerónimo Luciano. (2014). G-Matrix: propuesta de algoritmo para búsqueda de (sub) isomorfismos en grafos conexos. Estrategia de división del espacio de búsqueda y ejecución en paralelo [en Línea]. Universidad Nacional de Lanús. Departamento de Desarrollo Productivo y Tecnológico

Disponible en: http://www.repositoriojmr.unla.edu.ar/descarga/TFI/LicSis/035669_Tondato.pdf

**Condiciones de uso**

www.repositoriojmr.unla.edu.ar/condicionesdeuso

www.unla.edu.ar
www.repositoriojmr.unla.edu.ar
repositoriojmr@unla.edu.ar

# G-MATRIX: PROPUESTA DE ALGORITMO PARA BÚSQUEDA DE (SUB)ISOMORFISMOS EN GRAFOS CONEXOS

ESTRATEGIA DE DIVISIÓN DEL ESPACIO DE BÚSQUEDA
Y EJECUCIÓN EN PARALELO

Alumno
**A.P.U. Gerónimo Luciano Tondato**

Directores
**Mg. Darío Rodríguez e Ing. Hernán Amatriain**

Asesor Científico
**Dr. Ramón García-Martínez**

TRABAJO FINAL PRESENTADO PARA OBTENER EL GRADO
DE
LICENCIADO EN SISTEMAS

**DEPARTAMENTO**
**DE DESARROLLO PRODUCTIVO Y TECNOLÓGICO**
**UNIVERSIDAD NACIONAL DE LANUS**

**NOVIEMBRE, 2014**

Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa”
UNLa.

# RESUMEN

Durante las últimas décadas, los grafos han sido ampliamente estudiados. Recientemente con la explosión de internet y el advenimiento de las redes sociales, los mismos tomaron un rol fundamental. Un tema importante en el área, es el estudio del problema de búsqueda de (sub)isomorfismo en grafos, donde, dados un par de grafos A y B, consiste en determinar si A contiene un subgrafo que es isomorfo a B. Si bien esto parece simple de imaginar, el problema de búsqueda de (sub)isomorfismo está catalogado en la categoría de los NP-Completos. Debido a su gran complejidad, algoritmos capaces de dar soluciones rápidas son altamente deseables. En este trabajo presentamos un nuevo algoritmo de búsqueda de (sub)isomorfismo en grafos, al cual denominamos G-Matrix. Adicionalmente, hemos tenido en cuenta los nuevos paradigmas de hardware, y es por eso que también desarrollamos una metodología para dividir el espacio de búsqueda, que nos permite ejecutar nuestro algoritmo de forma paralela en computadores de múltiples núcleos o directamente sobre una red equipos informáticos.

# ABSTRACT

Over the past few decades, many graph related studies have been carried out. Most recently, with internet explosion and the proliferation of social networks, graphs have become mainstream again. One important topic in the field is the subgraph isomorphism problem, where given two graphs A and B as input, one must determine whether A contains a subgraph that is isomorphic to B. Though it seems easy to conceptualize, pure subgraph isomorphism problem is cataloged under NP-Complete category. Due its hardness, fast algorithms are highly desirable. In this work, we present a novel algorithm to address this problem, we named it: G-Matrix. In addition, we took into consideration new hardware architectures, that is why we also developed an efficient methodology aiming to search space division, which let us execute our algorithm in a parallel way on computers of multiple cores or directly over a network.

Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# DEDICATORIA

A mi papá Antonio, por haber estado ahí, siempre, siempre.

A mi mamá Juanita, por su afecto, su sonrisa, y su amor por cada ser vivo de este planeta.

A mi hermana Laura por su cariñosa amistad.

A mi hermana Ivana, por ser ejemplo de inquebrantable voluntad.

A mi abuela Fina y a mi abuela “del campo” por quererme más de la cuenta.

A mis amigos de la infancia: Braulio y Lucas, por ser espejo de mis penas y alegrías.

Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# AGRADECIMIENTOS

A la Universidad Nacional de Lanús por acogerme con generosidad de "alma máter".

Al Dr. Ramón García-Martínez, que a lo largo de los años logró responderme lo imposible:

*"¿Y esto para qué me sirve?"*.

Al Mg. Darío Rodríguez por su guía y su apoyo de amigo incondicional.

Al Ing. Herman Amatriain por la lectura detenida del trabajo, sus comentarios y sugerencias.

A mis profesores de cátedra que supieron alimentar mis ganas de aprender cada día más.

A mis compañeros de cursada, quienes hicieron de estos años de estudio un recuerdo para toda la vida.

Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# ÍNDICE

**1. INTRODUCCIÓN** **1**
1.1 MARCO DEL TRABAJO FINAL DE LICENCIATURA 1
1.2. DELIMITACIÓN DEL PROBLEMA 2
1.3. SOLUCIÓN PROPUESTA 2
1.4. VISIÓN GENERAL DEL TFL 3
**2. ESTADO DE LA CUESTIÓN** **5**
2.1. INTRODUCCIÓN: ¿QUÉ SON LOS GRAFOS? 5
2.2. CONCEPTOS NECESARIOS 6
2.2.1. Teoría de Grafos 6
2.2.2. Teoría de matrices booleanas 9
2.2.3. Formas de representación de grafos 10
2.3. EL PROBLEMA DE BÚSQUEDA DE (SUB) ISOMORFISMO EN GRAFOS 12
2.3.1. Clasificada en NP-Completo 13
2.3.2. Búsqueda en profundidad 13
2.3.3. Backtracking 14
2.3.4. Paralelización de algoritmos de búsqueda 16
2.4. ALGORITMO DE ULLMAN 17
2.4.1. Generalidades 17
2.4.2. Funcionamiento del Algoritmo de Ullman 17
**3. DESCRIPCIÓN DEL PROBLEMA** **21**
3.1. IDENTIFICACIÓN DEL PROBLEMA DE INVESTIGACIÓN 21
3.2. PROBLEMA ABIERTO 21
3.3. SUMARIO DE INVESTIGACIÓN 22
**4. SOLUCIÓN** **23**
4.1. INTRODUCCIÓN A LA SOLUCIÓN 23
4.2. ALGORITMO G-MATRIX: APORTACIÓN DE ESTE T.F.L 23
4.2.1. Lógica de G-Matrix: cómo funciona (comparado con Ullman) 24
4.2.2. Explicación detallada del funcionamiento de G-Matrix 28
4.2.2.1. Flujo de ejecución del algoritmo 28
4.2.2.2. Creación de la matriz inicial $M^t$ 30



Repositorio Digital Institucional "José María Rosa"
UNLa.

4.2.2.3. Creación de la lista que determina el orden de ejecución 32
4.2.2.4. Proceso de reducción de la matriz $M^t$ 34
4.2.3. Presentación del pseudocódigo del algoritmo G-Matrix 37
4.3. METODOLOGÍA DE PARALELIZACIÓN Y DIVISIÓN DEL ESPACIO DE BÚSQUEDA: APORTACIÓN DE ESTE T.F.L 40
4.3.1. ¿Por qué paralelizar? 40
4.3.2. División del espacio de búsqueda 41
4.3.2.1. Lógica de la división del espacio de búsqueda: cómo funciona 41
4.3.2.2. Explicación con ejemplo del proceso de división del espacio de búsqueda 42
4.3.2.3. Algoritmo para dividir el espacio de búsqueda 45
4.3.3. Estrategia de Ejecución en paralelo de los algoritmos 46
**5. EXPERIMENTACIÓN 49**
5.1. ENFOQUE DE LA EXPERIMENTACIÓN: GENERALIDADES 49
5.2. GENERADOR DE GRAFOS SINTÉTICOS 50
5.2.1. Generalidades 50
5.2.2. Algoritmo del generador de grafos sintéticos 50
5.2.3. Explicación del proceso de generación de grafos pasó a paso 52
5.3. CONFIGURACIÓN DE LA EXPERIMENTACIÓN, METODO DE MONTECARLO 55
5.3.1. Estructura de la tabla de parámetros de entrada siguiendo el método de Montecarlo 56
5.3.2. Fórmulas que limitan el rango de valores de los parámetros de entrada 58
5.3.3. Tabla de Montecarlo completa con los valores de los parámetros de entrada 61
5.4. RESULTADOS DE LA EJECUCIÓN DEL ALGORITMO DE ULLMAN 63
5.4.1. Tabla de resultados 64
5.4.2. Valoración del algoritmo, estadísticas, observaciones 66
5.5. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO DE ULLMAN APLICANDO PARALELISMO. 68
5.5.1. Tabla de resultados 68
5.5.2. Valoración del algoritmo, estadísticas, observaciones 71
5.6. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO G-MATRIX 73
5.6.1. Tabla de resultados 73



Repositorio Digital Institucional "José María Rosa"
UNLa.

5.6.2. Valoración del algoritmo, estadísticas, observaciones 76

5.7. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO G-MATRIX APLICANDO PARALELISMO. 77

5.7.1. Tabla de resultados 77

5.7.2. Valoración del algoritmo, estadísticas, observaciones 80

5.8. COMPARACIÓN Y EVALUACIÓN DE LOS RESULTADOS OBTENIDOS 82

5.8.1. Tabla comparativa de casos con resultado 83

5.8.2. Tests de Wilcoxon 83

5.8.2.1. Generalidades 83

5.8.2.2. Ullman Vs G-Matrix 86

5.8.2.3. Ullman vs Ullman paralelo 88

5.8.2.4. G-Matrix Vs G-Matrix paralelo 90

5.8.3. Apreciaciones finales 92

**6. CONCLUSIONES 93**

6.1. APORTES DEL TRABAJO FINAL DE LICENCIATURA 93

6.2. FUTURAS LÍNEAS DE INVESTIGACIÓN 94

**7. REFERENCIAS 95**

**VIII. ANEXO 99**

VIII.1 ALGORITMOS 99

VIII.1.1 Ullman 99

VIII.1.2. Ullman Paralelo 102

VIII.1.3 G-Matrix 107

VIII.1.4. G-Matrix Paralelo 111

VIII.1.5. Operaciones sobre Matrices 117

VIII.1.6. Recursos de Ejecución 121

VIII.2. CREACION DE LOS SET DE DATOS 122

VIII.2.1. Escribir resultados a disco 122

VIII.2.2. Excepción por definición de parámetros invalida 123

VIII.2.3. Generador de etiquetas 124

VIII.2.4. Grafo mayor (Contenedor) 124

VIII.2.5. Grafo menor (Objetivo) 127

VIII.2.6. Vértice 129

VIII.2.7. Generador de números aleatorios 130




Repositorio Digital Institucional "José María Rosa"
UNLa.

VIII.3. CLASES DE PRUEBA 131
VIII.3.1. Test de ejecución 131
VIII.3.2. Selector de parámetros 136
VIII.3.3. Señal para detener ejecución 138



Repositorio Digital Institucional "José María Rosa"
UNLa.

# ÍNDICE DE ALGORITMOS


| | | |
|---|---|---|
| Algoritmo 2.1. | Pseudocódigo del algoritmo de Ullman | 18 |
| Algoritmo 4.1. | G-Matrix | 37 |
| Algoritmo 4.2. | G-Matrix –Procedimiento: Crear matriz inicial M | 38 |
| Algoritmo 4.3.a. | G-Matrix –Procedimiento: Crear lista de ejecución | 38 |
| Algoritmo 4.3.b. | G-Matrix –Procedimiento: Crear lista de ejecución | 39 |
| Algoritmo 4.4.a. | G-Matrix –Procedimiento: Reducir M | 39 |
| Algoritmo 4.4.b. | G-Matrix –Procedimiento: Reducir M | 40 |
| Algoritmo 4.5.a. | Procedimiento para dividir el espacio de búsqueda. | 45 |
| Algoritmo 4.5.b. | Procedimiento para dividir el espacio de búsqueda. | 46 |
| Algoritmo 5.1. | Generador de grafos sintéticos. | 51 |




Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# ÍNDICE DE FIGURAS


Figura 2.1. Ejemplo de grafo con vértices y aristas señaladas 5

Figura 2.2. a) Red Social b) Componente químico. c) Diagrama de proceso d) Estructura de una proteína 6

Figura 2.3. a) Un grafo simple, b) Un subgrafo de a 7

Figura 2.4. Representación de isomorfismo, ambos grafos tienen la misma topología 8

Figura 2.5. Representación de sub-isomorfismo, tanto a como b son subgrafos de c 9

Figura 2.6. Representación de una matriz booleana 9

Figura 2.7. Grafo representado en una lista de adyacencia 10

Figura 2.8. Grafo representado en una lista de incidencia 10

Figura 2.9. Grafo representado en una matriz de adyacencia 11

Figura 2.10. Grafo representado en una matriz de incidencia 11

Figura 2.11. Búsqueda en profundidad, orden en el que se visitan los nodos 14

Figura 2.12. Repartición del árbol de búsqueda entre n procesadores 16

Figura 2.13. Esquema maestro/Esclavo 16

Figura 4.1. Ejemplo de una matriz M que codifica un sub-Isomorfismo 25

Figura 4.2. Construcción de Mt de atrás hacia adelante 27

Figura 4.3. Vista general de la lógica de G-Matrix explicada con matrices 27

Figura 4.4. Representación del árbol de búsqueda de Mt 28

Figura 4.5. Flujo de ejecución de G-Matrix 29

Figura 4.6. a) Grafo G. a.1) Matriz de adyacencia de G. b) Grafo G'. b.1) Matriz de adyacencia de G' 30

Figura 4.7. Ej. de primera instancia de M, sin ningún filtro 31

Figura 4.8. Ej. de segunda instancia de M, con filtro de primer nivel 31

Figura 4.9. Ej. de M inicial terminada, con filtro de segundo nivel 32

Figura 4.10. Ej. de Mt inicial 32

Figura 4.11. a) Matriz de adyacencia de G. b) Lista sin orden de G 33

Figura 4.12. Matriz Mt señalando la columna con menor cantidad de filas activas 33

Figura 4.13. Lista ordenada de G 34

Figura 4.14. Diagrama de flujo del proceso de reducción de M t 35

Figura 4.15. Vista del proceso de reducción de Mt explicado con matrices. 36




Repositorio Digital Institucional "José María Rosa"
UNLa.

Figura 4.16. M de dimensiones |a| x |b| con cada i,j = 1 42
Figura 4.17. M con su fila más activa señalada 43
Figura 4.18. División entre la cantidad de 1 en la máxima fila de M y el número de procesos. 43
Figura 4.19. Creación de la lista D 44
Figura 4.20. Fraccionamiento de M 44
Figura 4.21. Diagrama de ejecución en paralelo 47
Figura 5.1. Paso 1 Vértices aislados. 53
Figura 5.2. Paso 2 Vértices conectados 53
Figura 5.3. Paso 3 Grafo objetivo completado. 53
Figura 5.4. Paso 4 Grafo con vértices aislado. 53
Figura 5.5. Paso 5 Grafo conexo. 54
Figura 5.6. Paso 6 Grafo conexo completado. 54
Figura 5.7. Paso 7 Grafo terminado con los id's permutados. 55
Figura 5.8. a) Grafo objetivo. b) Grafo contenedor con subisomorfismo señalado. 55
Figura 5.9. Diagrama de conjuntos. La parte sombreada representa la máxima cantidad de aristas de G' 60
Figura 5.10. Grafico circular - Ullman. Muestra los porcentajes de casos resueltos y no resueltos. 67
Figura 5.11. Gráfico de dispersión – Ullman. Muestra los casos no resueltos en función de las densidades de G y G'. 67
Figura 5.12. Gráfico de dispersión - Ullman. Muestra los casos resueltos en función de las densidades de G y G'. 68
Figura 5.13. Grafico circular. Muestra los porcentajes de casos resueltos y no resueltos. 71
Figura 5.14. Gráfico de dispersión – Ullman paralelizado. Muestra los casos no resueltos en función de las densidades de G y G'. 72
Figura 5.15. Gráfico de dispersión – Ullman Paralelizado. Muestra los casos resueltos en función de las densidades de G y G'. 72
Figura 5.16. Grafico circular – G-Matrix. Muestra los porcentajes de casos resueltos y no resueltos. 76
Figura 5.17. Gráfico de dispersión – G-Matrix. Muestra los casos no resueltos en función de las densidades de G y G'. 76
Figura 5.18. Gráfico de dispersión – G-Matrix. Muestra los casos resueltos en función de las densidades de G y G'. 77



Repositorio Digital Institucional "José María Rosa"
UNLa.

Figura 5.19. Gráfico circular – G-Matrix paralelizado. Muestra los porcentajes de casos resueltos y no resueltos. 81

Figura 5.20. Gráfico de dispersión – G-Matrix paralelizado. Muestra los casos resueltos en función de las densidades de G y G'. 81

Figura 5.21. Gráfico de dispersión – G-Matrix paralelizado. Muestra los casos resueltos en función de las densidades de G y G'. 82



Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# ÍNDICE DE TABLAS


Tabla 2.1. Compara aspectos de eficiencia de las listas de adyacencia e incidencia y de las matrices de adyacencia e incidencia. 12

Tabla 4.1. Similitudes y diferencias entre Ullman y G-Matrix 26

Tabla 5.1. Resumen de la Tabla de Montecarlo. Configuración de la tabla de parámetros de entrada 57

Tabla 5.2.a. Tabla de Montecarlo. Tabla con los valores de parámetros de entrada calculados. 61

Tabla 5.2.b. Tabla de Montecarlo. Tabla con los valores de parámetros de entrada calculados. 62

Tabla 5.2.c. Tabla de Montecarlo. Tabla con los valores de parámetros de entrada calculados. 63

Tabla 5.3.a. Resultados de la ejecución del algoritmo de Ullman 64

Tabla 5.3.b. Resultados de la ejecución del algoritmo de Ullman 65

Tabla 5.3.c. Resultados de la ejecución del algoritmo de Ullman 66

Tabla 5.4.a. Resultados de la ejecución del algoritmo de Ullman paralelizado 69

Tabla 5.4.b. Resultados de la ejecución del algoritmo de Ullman paralelizado 70

Tabla 5.4.c. Resultados de la ejecución del algoritmo de Ullman paralelizado 71

Tabla 5.5.a. Resultados de la ejecución del algoritmo G-Matrix 73

Tabla 5.5.b. Resultados de la ejecución del algoritmo G-Matrix 74

Tabla 5.5.c. Resultados de la ejecución del algoritmo G-Matrix 75

Tabla 5.6.a. Resultados de la ejecución del algoritmo G-Matrix paralelizado 78

Tabla 5.6.b. Resultados de la ejecución del algoritmo G-Matrix paralelizado 79

Tabla 5.6.c. Resultados de la ejecución del algoritmo G-Matrix paralelizado 80

Tabla 5.7. Algoritmo – Porcentaje. Muestra el porcentaje de casos con solución para cada algoritmo 83

Tabla 5.8. Estructura de una tabla para Test de Wilcoxon 84

Tabla 5.9. Valores críticos de Z 85

Tabla 5.10.a. Wilcoxon, Ullman vs. G-Matrix 86

Tabla 5.10.b. Wilcoxon, Ullman vs. G-Matrix 87

Tabla 5.11. Wilcoxon, Ullman lineal vs. Ullman paralelo 89

Tabla 5.12.a. Wilcoxon, G-Matrix lineal vs. G-Matrix paralelo 90

Tabla 5.12.b. Wilcoxon, G-Matrix lineal vs. G-Matrix paralelo 91




Repositorio Digital Institucional "José María Rosa" UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# 1. INTRODUCCIÓN

Este capítulo sirve como introducción a las temáticas tratadas en este Trabajo Final de Licenciatura. En la sección 1.1 se confecciona un marco teórico. En la sección 1.2 se delimitan los alcances del problema que será tratado. En la sección 1.3 se introducen brevemente las soluciones propuestas. Finalmente, en la sección 1.4 se realiza una visión general de este trabajo.

## 1.1. MARCO DEL TRABAJO FINAL DE LICENCIATURA

Los grafos son herramientas con una gran potencialidad. Desde su aparición en 1736 a manos del renombrado matemático suizo Leonhard Euler, quien los utilizó para resolver el problema de los puentes de Königsberg, sus usos y aplicaciones no han hecho más que incrementarse. La capacidad desestructurada de los grafos que permite modelar desde los problemas más sencillos y triviales hasta aquellos de gran complejidad, no pasó desapercibida por los científicos de la computación, quienes por décadas han estado inmersos en un intento por perfeccionar las técnicas que hacen posible explotar el potencial de esta herramienta [Conte et al., 2004]. Muchos investigadores se han enfocado en el desarrollo de metodologías que permiten tanto el almacenamiento como el análisis de datos sobre grafos. Eventualmente surge la necesidad de poder extraer la información que estas estructuras contienen, la creación de procesos eficientes para hacer consultas sobre grafos ha pasado a ser un tema presente en la mente de muchos investigares debido a sus múltiples aplicaciones [Ferro et al., 2007; Przulj et al., 2006; Ohlrich et al., 1993].

Los procesos de consultas varían según el tipo de base de datos de grafos al cual se estén aplicando, las existen principalmente de dos tipos [Lee et al., 2012]. Por un lado se tienen bases de datos que consisten en una multitud de pequeños o medianos grafos puestos en conjunto, donde el estilo de consulta que se realiza sobre ellos, se basa en determinar cuántos de estos módicos grafos son similares o contienen subgrafos de estructura similar al que se plantea en la consulta. Por otro lado se tienen bases de datos compuestas por un único grafo conexo de gran tamaño y el objetivo del proceso de consulta consiste en encontrar subgrafos dentro de esta estructura que sean similares a un grafo dado.

Existen generalmente dos grandes categorías de algoritmos a la hora de comparar grafos [Wang, 2010]. Aquellos que pretenden una comparación exacta, donde el grafo hallado debe coincidir plenamente con el de la consulta, a estos se los denomina de *búsqueda exacta*. Y otros que permiten cierto grado de distancia entre lo que se busca y lo que se encuentra, normalmente usan algún tipo



Repositorio Digital Institucional "José María Rosa"
UNLa.

de función que posibilita medir la magnitud de la diferencia que existe entre ambos grafos. Estos últimos se denominan de *búsqueda inexacta*.
No obstante, todos los acercamientos mencionados anteriormente se reducen al problema de búsqueda de (sub)isomorfismo en grafos, cuya complejidad se sabe que se encuentra dentro de la categoría de los NP-Completos [Conte et al., 2004], lo que significa que dar con resultados satisfactorios pasa a ser notablemente difícil a medida que crece el tamaño de los grafos.
En casi tres siglos, el estudio de teoría de grafos se ha topado con la complejidad extrema, con problemas irresolubles, o cuya solución abarca tiempos inaceptables. En este Trabajo Final de Licenciatura se da una iteración más sobre esta creciente área de estudio.

## 1.2. DELIMITACIÓN DEL PROBLEMA

Este trabajo se enfoca en los problemas de búsqueda de (sub)isomorfismo sobre un único grafo conexo de gran magnitud, es decir, poder determinar si un grafo $\mathbf{G_1}$ se encuentra replicado dentro de la estructura de un grafo $\mathbf{G_2}$ de mayor tamaño. Para las comparaciones solo se tiene en cuenta las restricciones topológicas de los grafos, esto es, se ignoran etiquetas sobre vértices o aristas ponderadas. El tipo de coincidencia que se busca entre los grafos debe ser exacta, cualquier tipo de aproximación inexacta es descarta y no se enlista como posible solución. Se le presta especial atención al algoritmo de Ullman [1976], y se lo utiliza como modelo conceptual para la posterior investigación.

## 1.3. SOLUCIÓN PROPUESTA

Se confeccionan dos acercamientos de solución:

- Por un lado, se propone un nuevo algoritmo para realizar búsquedas exactas de (sub)isomorfismo sobre grafos conexos; una extensión de un algoritmo de búsqueda en profundidad, basado en técnicas de *backtracking*, que utiliza una función de recorte para podar el árbol de búsqueda en cada iteración, guiado por heurísticas simples pero efectivas.

- Por el otro, se propone una metodología para dividir el espacio de búsqueda sobre el que opera el algoritmo, en una cantidad de fragmentos parametrizable, que permite la paralelización del algoritmo y su ejecución más eficiente sobre arquitecturas de múltiples procesadores, así como también sobre una red de computadores.



Repositorio Digital Institucional "José María Rosa"
UNLa.

## 1.4. VISIÓN GENERAL DEL TFL

A partir de este punto, el trabajo aquí presentado se estructura de la siguiente manera:

En el capítulo II, *Estado de la Cuestión***:** Se explican conceptos afines a los grafos, se tratan temas como isomorfismo y (sub)isomorfismo, se presentas formas de representación de grafos en listas y matrices, se presentan conceptos utilizados por los algoritmos de (sub)isomorfismo, búsquedas en profundidad, espacio de búsqueda, *backtracking*, se explica su clasificación dentro de la categoría de los NP-Completos, finalmente se describe al algoritmo de Ullman.

En el capítulo III, *Problema:* Se brindan nociones que permiten entender cómo funciona el crecimiento exponencial del espacio de búsqueda de este tipo de problemas a medida que crece el tamaño de la entrada, lo que constituye la identificación del problema de investigación. Se presentan los esfuerzos que otros autores están haciendo entorno al problema. Y se concluye con un sumario de investigación.

En el capítulo IV, *Solución:* Se aborda el problema con el desarrollo de un nuevo algoritmo de búsqueda de (sub)isomorfismo, nombrado G-Matrix, se explica su concepción y lógica de funcionamiento, la heurística aplicada, y su función de poda. Por otro lado, también se desarrolla una metodología para dividir el espacio de búsqueda en función de poder paralelizar el algoritmo.

En el capítulo V, *Experimentación:* Se explica el proceso de generación de pares de grafos sintéticos. Se ponen a prueba las propuestas siguiendo el método de Montecarlo. Para medir la eficiencia de G-Matrix se lo compara contra el algoritmo de Ullman. Y para testear la metodología de división de espacio de búsqueda, se implementan versiones paralelizadas de G-Matrix y Ullman y se las compara contra sus versiones lineales. Los resultados son evaluados siguiendo el test de rangos con signos de Wilcoxon [Vassarstats, 2014].

En el capítulo VI, *Conclusiones:* Se resaltan las aportaciones logradas en este Trabajo Final de Licenciatura y se confecciona una lista de futuras líneas de investigación.

En el capítulo VII, *Referencias:* Se lista la bibliografía consultada para el desarrollo de este trabajo.

En el capítulo VIII, *Anexo:* Se pone a disposición todo el trabajo de programación realizado para este T.F.L.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# 2. ESTADO DE LA CUESTIÓN

En este capítulo se presenta el estado de la cuestión. En la sección 2.1 se hace una breve introducción a los grafos y se señala su presencia en campos de estudio que los utilizan en la actualidad. En la sección 2.2 se detallan conceptos necesarios sobre teoría de grafos y matrices booleanas. En la sección 2.3 se introduce el problema de búsqueda de (sub)Isomorfismo en grafos, se lo clasifica, y se explican conceptos relevantes sobre búsqueda en profundidad, *backtracking* y paralelización. Finalmente en la sección 2.4 se presenta y describe el funcionamiento del algoritmo de Ullman para búsqueda (sub)grafos isomorfos.

## 2.1. INTRODUCCIÓN: ¿QUÉ SON LOS GRAFOS?

Las calzadas romanas, la red eléctrica y, recientemente, Internet. Todas las redes que el ser humano ha diseñado para su comunicación y progreso tienen algo en común: se pueden representar mediante grafos que nos ayudan a modelar y comprender mejor cómo funcionan. Conceptualmente hablando un grafo es una herramienta que permite modelar una estructura compuesta de entidades y relaciones. Donde las entidades vienen a estar representadas por algo que se llama *vértice* y a las relaciones que conectan dichas entidades se llaman aristas *aristas (ver figura 2.1).* Si se toma por ejemplo a Facebook, se puede decir que cada persona que forma parte de esa red social constituye un vértice, y toda vez que dos personas se aceptan mutuamente como amigos se genera un arco o arista que los une representando dicha relación. Luego Facebook pasa a ser modelado como un grafo de inmensa magnitud. En el caso particular mencionado anteriormente todas las relaciones son bidireccionales, A es amigo de B entonces B es amigo de A, esto vendría a representar un grafo no-dirigido (en los que se basa este trabajo) pero también los hay con relaciones unidireccionales, con relajaciones reflexivas de un vértice en sí mismo, con etiquetas que le ponen nombre a los vértices y ponderan las aristas, etc. Más conceptos sobre grafos se introducen en la sección 2.2.1.

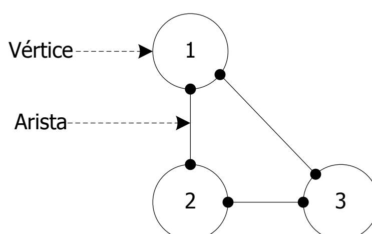



**Figura 2.1.** *Ejemplo de grafo con vértices y aristas señaladas.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Las *figuras 2.2 (a, b, c y d)* representan solo algunas de la gran cantidad de áreas en las que se pueden encontrar grafos. El carácter heterogéneo de la aplicación de grafos explica su creciente interés durante la última década

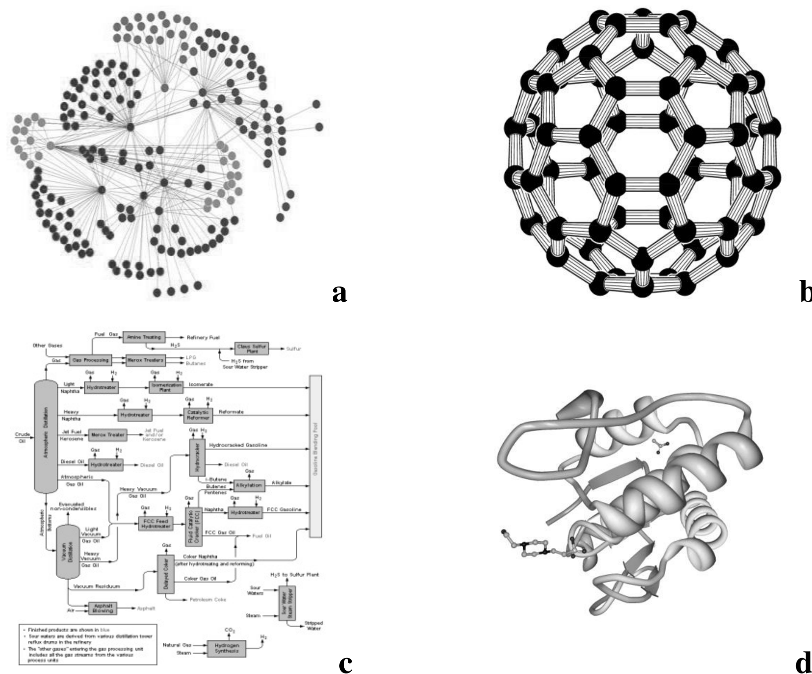



**Figura 2.2. a)** *Red Social* **b)** *Componente químico*. **c)** *Diagrama de proceso* **d)** *Estructura de una proteína*

# 2.2. CONCEPTOS NECESARIOS

En esta sección se presentan conceptos necesarios sobre teoría de grafos y matrices booleanas. En la sección 2.2.1 se aborda la teoría de grafos y en la sección 2.2.2 se aborda teoría sobre matrices booleanas.

## 2.2.1. Teoría de Grafos

A continuación se presentan una serie de conceptos y definiciones sobre teoría de grafos [Gross y Yellen, 2003; Biggs, 1993].

**Grafo:** Un grafo $G = (V,E)$ está compuesto de dos conjuntos, el conjunto de vértices $V$ y el conjunto de aristas $E$. Cada arista $e \in E$ está formada por un par no ordenado de vértices $uv$ donde $u,v \in V$ (ver *Figura. 2.3.a*).



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Subgrafo:** Un subgrafo de un grafo $G = (V(G),E(G))$ es un grafo $H = (V(H),E(H)) \ / \ V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$. Es decir, si $G = (V,E)$ es un grafo, entonces $G_1 = (V_1,E_1)$ es un subgrafo de $G$ si $V_1$ es distinto del conjunto vacío y $E_1$ es subconjunto de $E$, donde cada arista de $E_1$ es incidente con los vértices de $V_1$ (ver *Figura. 2.3.b*).

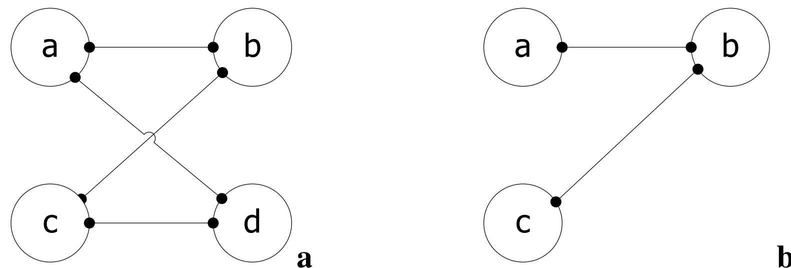



**Figura 2.3. a)** *Un grafo simple,* **b)** *Un subgrafo de a.*

**Adyacencia:** Dos aristas son adyacentes si tienen un vértice en común, y dos vértices son adyacentes si una arista los une.

**Incidencia:** Una arista es incidente a un vértice si esta lo une a otro.

**Ponderación**: Corresponde a una función que a cada arista le asocia un valor (costo, peso, longitud, etc.), para aumentar la expresividad del modelo. Esto se usa mucho para problemas de optimización, como el del vendedor viajero o del camino más corto.

**Etiquetado:** Distinción que se hace a los vértices y/o aristas mediante una marca que los hace unívocamente distinguibles del resto.

**Densidad:** Sea $G = (V,E)$ un grafo simple con $|V| = número\ de\ vértices, |E| = número\ de\ aristas.$ Se define la densidad del grafo como

$$d = \frac{2|E|}{|V|(|V|-1)} \tag{2.1}$$

Notar que $0 < d < 1$, donde $d = 0$ si todos los vértices son aislados y $d = 1$ si el grafo es completo. Si $d$ es cercano a 0 se dice que el grafo es disperso y si $d$ es cercano a 1 se dice que el grafo es denso.



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Árbol:** Un árbol es un grafo en el que cualesquiera dos vértices están conectados por exactamente un camino; es un grafo simple no dirigido $G$ que satisface:

- $G$ es conexo y no tiene ciclos.
- $G$ no tiene ciclos y, si se añade alguna arista se forma un ciclo.
- $G$ es conexo y si se le quita alguna arista deja de ser conexo.
- Dos vértices cualesquiera de $G$ están conectados por un único camino simple.

**Árbol de expansión T:** es un árbol compuesto por todos los vértices y algunas (quizá todas) de las aristas de $G$. Informalmente, un árbol de expansión de $G$ es una selección de aristas de $G$ que forman un árbol que cubre todos los vértices. Esto es, cada vértice está en el árbol, pero no hay ciclos. Por otro lado, todos los puentes de $G$ deben estar contenidos en $T$.
Un árbol de expansión de un grafo conexo $G$ puede ser también definido como el mayor conjunto de aristas de $G$ que no contiene ciclos, o como el mínimo conjunto de aristas que conecta todos los vértices.

**Isomorfismo:** Dos grafos $G_1 = (V_1,E_1)$ y $G_2 = (V_2,E_2)$ son isomorfos, expresado como **$G_1$~$G_2$**, si son topológicamente idénticos el uno con el otro. Es decir, que existe una función bijectiva $f: V_1 \rightarrow V_2$ $con\ e = xy \in E_1 \leftrightarrow f(x) f(y) \in E_2$ para cada arista $e \in E_1$ donde $x,y \in V_1$ (ver *figura 2.4*).

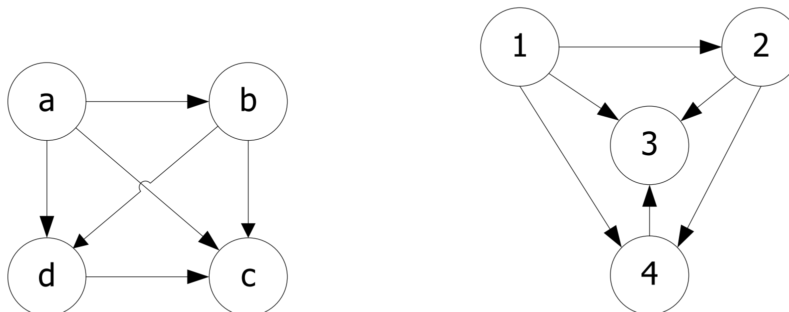



**Figura 2.4.** *Representación de isomorfismo, ambos grafos tienen la misma topología.*

**Sub-Isomorfismo:** Dados dos grafos $G_1 = (V_1,E_1)$ y $G_2 = (V_2,E_2)$, se puede describir al problema subgrafos isomorfos, como la búsqueda de isomorfismo entre **$G_2$** y un subgrafo de **$G_1$**. Es decir, determinar si $G_2$ se encuentra incluido en $G_1$ o no. Formalmente, dados que $G = (V,E), H = (V',E')$ sean grafos. ¿Existe un subgrafo $G_0 = (V_0,E_0): V_0 \subseteq V, E_0 = E \cap (V_0 \times V_0) \,/\, G_0 =\sim H$? Es decir, ¿Existe un $f: V_0 \longrightarrow V'$ tal que $(v_1,v_2) \in E_0 \leftrightarrow (f(v_1),f(v_2)) \in E'$? (ver *figura 2.5*).



Repositorio Digital Institucional "José María Rosa"
UNLa.

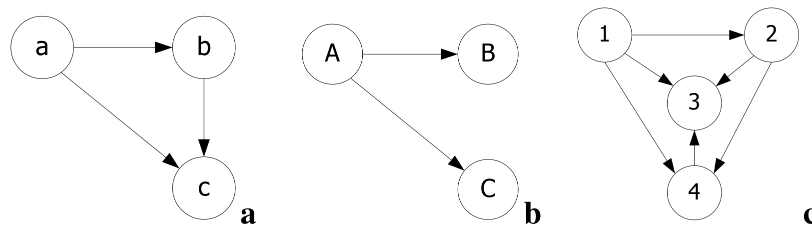



**Figura 2.5.** *Representación de sub-isomorfismo, tanto **a** como **b** son subgrafos de **c**.*

## 2.2.2. Teoría de matrices booleanas

A continuación se presentan conceptos y definiciones sobre teoría de matrices booleanas [Luce, 1952]

**Matriz booleana:** Una matriz booleana es una matriz de números cuyas componentes o entradas son exclusivamente ceros o unos. Las matrices booleanas son útiles porque pueden representar objetos abstractos como relaciones binarias o grafos. Una matriz booleana general de **|m| x |n|** elementos tiene la forma:

$A =$

| | 1 | 2 … | | n-1 | n |
|---|---|---|---|---|---|
| 1 | $a_{11}$ | $a_{12}$ | … | … | … |
| … | … | … | … | … | … |
| m-1 | … | … | … | … | … |
| m | … | … | … | … | $a_{mn}$ |

$Donde\ a_{i,j} = \{0|1\} \quad \forall\, i, j$

**Figura 2.6.** *Representación de una matriz booleana*

**Algunas Propiedades básicas de las operaciones con matrices booleanas:** Si **A**, **B** y **C** son matrices booleanas, entonces:

- A v B = B v A
- A^B = B^A
- A x (B x C) = (A x B) x C
- $(A^t)^t = A$
- $(A \times B)^t = B^t \times A^t$

**Multiplicación de matrices booleanas:** Dadas dos matrices $A_{n1,n2}$ y $B_{m1,m2}$ cuyo dominio está en *{0,1}* donde además $n2 = m1$**,** su multiplicación se define en la fórmula (2.2):

$$(AB)_{n1,m2}[i,j] = \bigvee_k (A(i,k) \wedge B(k,j)) \tag{2.2}$$



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 2.2.3. Formas de representación de grafos

En la práctica se usan muchas estructuras de datos diferentes para representar grafos:

**Lista de Adyacencia:** En esta estructura de datos se asocia a cada vértice *i* del grafo una lista que contenga todos aquellos vértices *j* que sean adyacentes a él. De esta forma sólo reservará memoria para los arcos adyacentes a *i* y no para todos los posibles arcos que pudieran tener como origen *i*. El grafo, por tanto, se representa por medio de un vector de *n* componentes donde cada componente va a ser una lista de adyacencia correspondiente a cada uno de los vértices del grafo. Cada elemento de la lista consta de un campo indicando el vértice adyacente (ver *figura 2.7*). En caso de que el grafo sea etiquetado, habrá que añadir un segundo campo para mostrar el valor de la etiqueta.

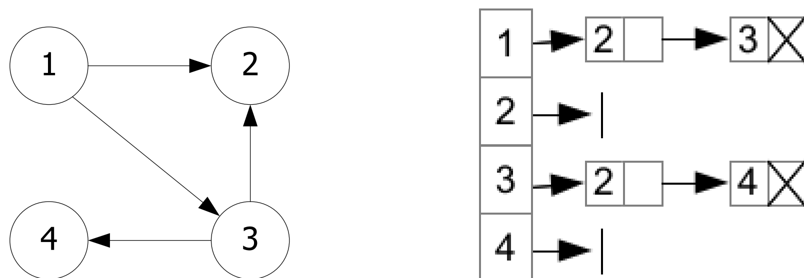



**Figura 2.7.** *Grafo representado en una lista de adyacencia.*

**Lista de Incidencia:** Las listas de incidencia son similares a las listas de adyacencia. Sin embargo, en lugar de relacionar vértices adyacentes, las listas de incidencia relacionan vértices con aristas (ver *figura 2.8*). Dado que estas proveen más rápido acceso a las aristas que las listas de adyacencia y las matrices, las listas de incidencia son una mejor opción a la hora de construir estructuras como circuitos de Euler, los cuales son circuitos cuyo propósito es visitar cada arista solamente una vez, y el punto de origen y fin es la misma arista.

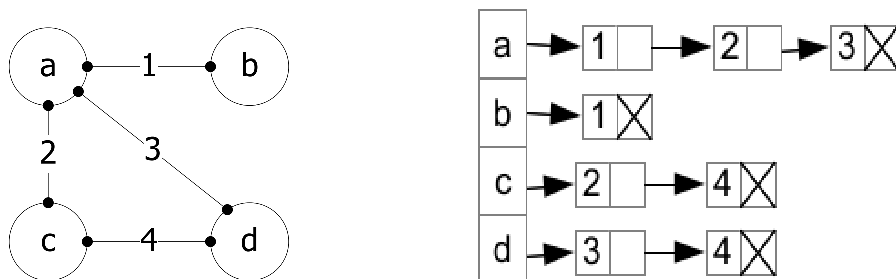


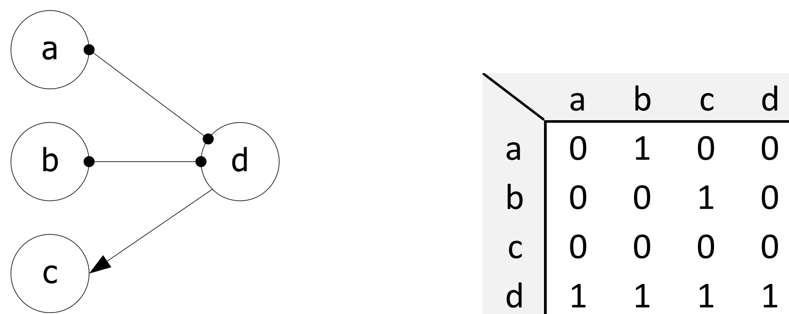

**Figura 2.8.** *Grafo representado en una lista de incidencia*.



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Matriz de Adyacencia:** La matriz de adyacencia es la forma más común de representación y la más directa. Consiste en una tabla de tamaño *|n| x |n|,* en que la que $a_{ij}$ tendrá como valor *1* si existe una arista del vértice *i* al vértice *j*. En caso contrario, el valor será *0*. Si el grafo es no dirigido hay que asegurarse de que se marca con un *1* tanto la entrada $a_{ij}$ como la entrada $a_{ji}$, puesto que se puede recorrer en ambos sentidos (ver *figura 2.9*).

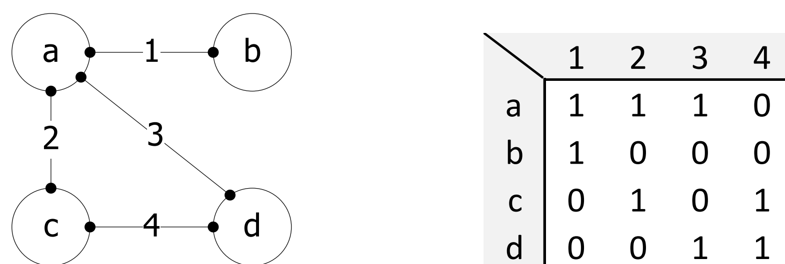
Como se puede apreciar, la matriz de adyacencia siempre ocupa un espacio de $n^2$, es decir, depende solamente del número de nodos y no del de aristas, por lo que será útil para representar grafos densos.




| | a | b | c | d |
|---|---|---|---|---|
| a | 0 | 1 | 0 | 0 |
| b | 0 | 0 | 1 | 0 |
| c | 0 | 0 | 0 | 0 |
| d | 1 | 1 | 1 | 1 |

**Figura 2.9.** *Grafo representado en una matriz de adyacencia.*

**Matriz de Incidencia:** Las matrices de incidencia son similares a las matrices de adyacencia, excepto que relacionan vértices y aristas como lo hacen las listas de incidencia. En las matrices de incidencia, las filas representan los vértices y las columnas representan las aristas (ver *figura 2.10*). Al igual que en las matrices de adyacencia, un *0* indica que un vértice no es incidente a una arista y un *1* indica que lo es.




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| a | 1 | 1 | 1 | 0 |
| b | 1 | 0 | 0 | 0 |
| c | 0 | 1 | 0 | 1 |
| d | 0 | 0 | 1 | 1 |

**Figura 2.10.** *Grafo representado en una matriz de incidencia.*

La *tabla 2.1* proporciona el costo computacional de algunas operaciones que se pueden realizar sobre grafos para las distintas representaciones vistas [Cormen et al., 2001].



Repositorio Digital Institucional "José María Rosa"
UNLa.

| | Lista de Adyacencia | Lista de Incidencia | Matriz de Adyacencia | Matriz de Incidencia |
|---|---|---|---|---|
| Almacenamiento | $O(\|V\|+\|E\|)$ | $O(\|V\|+\|E\|)$ | $O(\|V\|^2)$ | $O(\|V\|.\|E\|)$ |
| Agregar vértice | $O(1)$ | $O(1)$ | $O(\|V\|^2)$ | $O(\|V\|.\|E\|)$ |
| Agregar arista | $O(1)$ | $O(1)$ | $O(1)$ | $O(\|V\|.\|E\|)$ |
| Quitar vértice | $O(\|E\|)$ | $O(\|E\|)$ | $O(\|V\|^2)$ | $O(\|V\|.\|E\|)$ |
| Quitar arista | $O(\|E\|)$ | $O(\|E\|)$ | $O(1)$ | $O(\|V\|.\|E\|)$ |
| Consulta. Ej ¿Son los vértices $u,v$ adyacentes? Suponiendo que se conoce la ubicación de $u,v$ | $O(\|V\|)$ | $O(\|V\|)$ | $O(1)$ | $O(\|E\|)$ |
| Observaciones | Cuando necesita quitar algún vértice o arista, tiene que recorrer todos los vértices y aristas | | Es lento para agregar o quitar vértices porque la matriz tiene que ser redimensionada | Es lento para agregar o quitar vértices y aristas porque la matriz tiene que ser redimensionada |

**Tabla 2.1.** *Compara aspectos de eficiencia de las listas de adyacencia e incidencia y de las matrices de adyacencia e incidencia.*

Usualmente se prefiere a las listas de adyacencia debido a su eficiencia a la hora de representar grafos dispersos. Las matrices de adyacencia se prefieren en el caso de que el grafo sea muy denso, o en el caso que se requiera averiguar rápidamente si una arista conecta a dos vértices dados.

## 2.3. EL PROBLEMA DE BÚSQUEDA DE (SUB) ISOMORFISMO EN GRAFOS

En esta sección se presenta el problema de búsqueda de (sub)isomorfismo en grafos y se explican conceptos utilizados a la hora de encontrar soluciones. En la sección 2.3.1 se lo clasifica y se explica porque pertenece a la categoría de los NP-Completo. En la sección 2.3.2 se explica el concepto de búsqueda en profundidad y se añade un pseudocódigo. En la sección 2.3.3 se explica en concepto de *backtracking* y se añade un pseudocódigo. Finalmente, en la sección 2.3.4 se aborda sobre estrategias de paralelización y división del árbol de búsqueda.



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 2.3.1. Clasificada en NP-Completo

En teoría de la complejidad computacional, la clase de complejidad NP-completo [Cook, 1971] es el subconjunto de los problemas de decisión en NP tal que todo problema en NP se puede reducir en cada uno de los problemas de NP-completo. Se puede decir que los problemas de NP-completo son los problemas más difíciles de NP y probablemente no formen parte de la clase de complejidad P.

La prueba de que el problema de subgrafos isomorfos pertenece a la familia de los NP-Completos se basa en una reducción del problema de las cliques, un problema de decisión NP-Completo en el cual la entrada es un único grafo $G$ y un número $k$, donde además la interrogante que se plantea es si $G$ contiene un subgrafo completo de $k$ vértices. Para trasladar lo anterior al problema de subgrafos isomorfos, tómese a $H$ por un grafo completo $K_k$, luego la respuesta al problema de subgrafos isomorfos para $G$ y $H$ es igual a la respuesta del problema de la clique para $G$ y $k$. Dado que el problema de la clique es NP-Completo, esta reducción de muchos a uno en tiempo polinomial demuestra que el problema de subgrafos isomorfos es también NP-Completo [Weneger, 2005].

Una reducción alternativa que proviene del problema de los ciclos Hamiltonianos traslada un grafo $G$ al cual se le pone a prueba su Hamiltoneidad dentro de un par de grafos $G$ y $H$, donde $H$ es un ciclo que tiene el mismo número de vértices que $G$. Dado que el problema de los ciclos Hamiltonianos es NP-Completo incluso para grafos planos, esto demuestra que el problema de los subgrafos isomorfos se encuentra en NP-Completo aunque se tratase de un caso de grafos planos [De La Higera et al., 2013].

El problema de isomorfismo entre subgrafos es una generalización del problema de isomorfismo entre grafos, el cual se basa en determinar si un grafo $G$ es isomorfos a otro grafo $H$: esto será cierto si y solo si $G$ y $H$ tienen el mismo número de vértices y si el problema de isomorfismo entre subgrafos para $G$ y $H$ es también verdadero.

### 2.3.2. Búsqueda en profundidad

Una Búsqueda en profundidad es un algoritmo que permite recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

El análisis temporal y espacial de este tipo de algoritmos difiere según su área de aplicación. En ciencias de la computación, los algoritmos de búsqueda en profundidad son típicamente usados para



Repositorio Digital Institucional "José María Rosa"
UNLa.

recorrer grafos enteros, y tienen una complejidad de $O(|V| + |E|)$. En este tipo de aplicaciones, consumen un espacio $O(|V|)$ en el peor caso, para almacenar la pila de vértices de la búsqueda actual así como también el conjunto de vértices que ya han sido visitados.

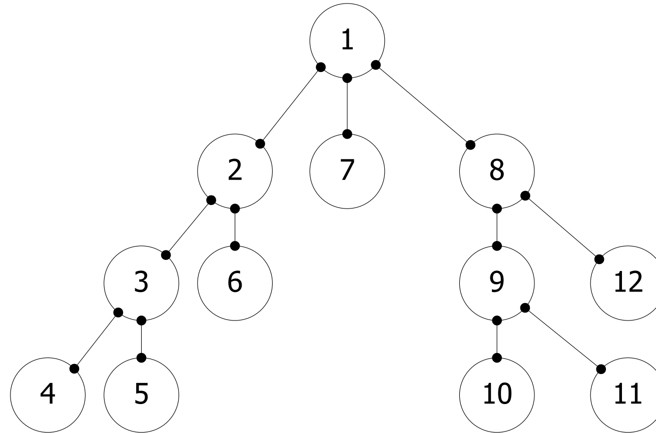



**Figura 2.11.** *Búsqueda en profundidad, orden en el que se visitan los nodos*

Se presenta el pseudocódigo del *algoritmo 2.2* de búsqueda en profundidad que utiliza un método recursivo [Goodrich y Tamassia, 2002]:

**Algoritmo**: Búsqueda en Profundidad

**Entrada**: G // Un grafo
v // un vértice de G

**Salida**: Todos los vértices alcanzables desde v

1: **Marcar** v como visto
2: **Para todas** las aristas desde v **hasta** w en G.aristasAdjacentes(v) **hacer**
3: **Si** vértice w no está marcado como visto **entonces**
4: Llamar recursivamente a Búsqueda_en_Profundidad(G,w)
5: **Fin si**
6: **Fin para**

**Algoritmo 2.1.** *Búsqueda en profundidad recursiva*

## 2.3.3. Backtracking

En su forma básica, la idea de *backtracking* se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente. El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el



Repositorio Digital Institucional "José María Rosa"
UNLa.

recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa.

El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución. En este caso el algoritmo puede, o bien detenerse (si lo único que se necesita es una solución del problema) o bien seguir buscando soluciones alternativas (si se desea examinarlas todas). Por otra parte, el recorrido no tiene éxito si en alguna etapa la solución parcial construida hasta el momento no se puede completar. En tal caso, el recorrido vuelve atrás exactamente igual que en un recorrido en profundidad, eliminando sobre la marcha los elementos que se hubieran añadido en cada fase. Cuando vuelve a un nodo que tiene uno o más vecinos sin explorar, prosigue el recorrido de una solución [Knuth, 1968].

**Algoritmo**: Backtracking

**Entrada**: X[1 . . . i ] // TSolución
Ascender: // Booleano

**Salida**: X[1….i]

```
1: L: ListaComponentes
2: Inicio
3:        Si EsSolución (X) Entonces
4:                Ascender ← Verdadero;
5:        Sino
6:                Ascender ← Falso;
7:                L ← Candidatos (X);
8:                Mientras ¬Ascender Y ¬Vacía (L) Hacer
9:                        X[i + 1] ← Cabeza (L);
10:                       L ← Resto (L);
11:                       Backtracking (X,Ascender);
12:               Fin Mientras
13:       Sin Si
14: Fin
```

**Algoritmo 2.2.** *Pseudocódigo backtracking*

Se puede visualizar el funcionamiento de una técnica de *backtracking* como la exploración en profundidad de un grafo.

Cada vértice del grafo es un posible estado de la solución del problema. Cada arco del grafo representa la transición entre dos estados de la solución (i.e., la toma de una decisión).

Típicamente el tamaño de este grafo será inmenso, por lo que no existirá de manera explícita. En cada momento sólo se tienen en una estructura los nodos que van desde el estado inicial al estado actual. Si cada secuencia de decisiones distinta da lugar a un estado diferente, el grafo es un árbol (el árbol de estados).



Repositorio Digital Institucional "José María Rosa"
UNLa.

## 2.3.4. Paralelización de algoritmos de búsqueda

Dado que en la actualidad la tendencia de los fabricantes es hacer microprocesadores que cada vez cuentan con más núcleos, la idea general será distribuir el espacio de búsqueda entre los distintos procesadores de los que se dispone, de forma que cada uno busque la solución del problema en un subespacio de soluciones distinto. De esta manera se explorarán varias ramas del árbol de soluciones al mismo tiempo en distintos procesadores. En problemas de búsqueda de solución aumentan las posibilidades de encontrar la solución en menos tiempo. La *figura 2.12* ejemplifica lo repartición del árbol de búsqueda entre n procesadores:

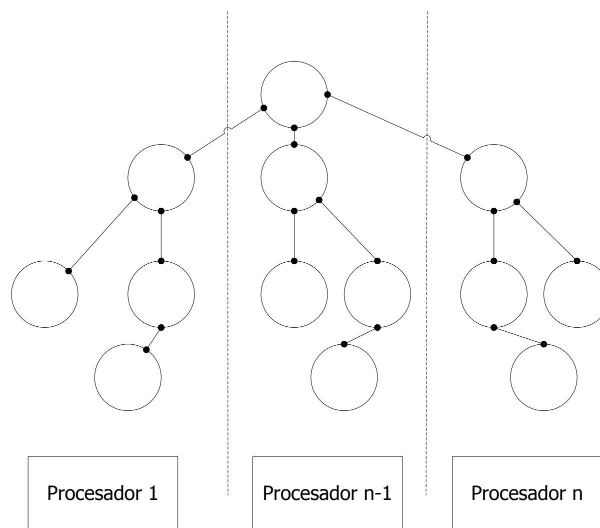



**Figura 2.12.** *Repartición del árbol de búsqueda entre n procesadores.*

Otro aspecto interesante es la comunicación entre los procesos para poder realizar el trabajo de forma independiente pero a la vez conjunta. Uno de los paradigmas de programación en paralelo más simples es el de maestro-esclavo. El proceso central (el maestro) genera varios subproblemas, los cuales son disparados para ser ejecutados por algún otro proceso (los esclavos).

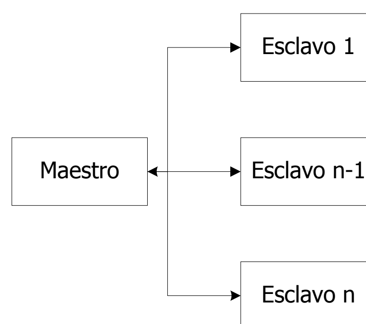



**Figura 2.13.** *Esquema maestro/Esclavo*



Repositorio Digital Institucional "José María Rosa"
UNLa.

La única interacción que existe entre el maestro y el esclavo es que el maestro inicia la ejecución del esclavo y luego el esclavo devuelve los resultados al maestro [Charm , 2014]

# 2.4. ALGORITMO DE ULLMAN

En esta sección se presenta y describe el algoritmo de Ullman para búsquedas de (sub)isomorfismo en grafos. En la sección 2.4.1 introducen generalidades. En la sección 2.4.2 se procede a explicar el funcionamiento del algoritmo más detalladamente.

## 2.4.1. Generalidades

Este trabajo se enfoca en los algoritmos de sub(isomorfismo) exactos (es decir aquellos que detectan exactamente la estructura buscada y no una aproximación) que no tienen ningún tipo de restricción topológica.

La solución al problema de sub(isomorfismo) en grafos a través de la fuerza bruta resulta en un algoritmo de búsqueda de árbol en profundidad. El algoritmo de Ullmann reduce el espacio de búsqueda aplicando un filtro de dos niveles, lo que ocasiona una reducción en el número de vértices posibles en los que hay que buscar. En el mejor de los casos Ullmann tiene una complejidad de $O(n^3)$ y en el peor de los casos de $O(n!.n^3)$ [Voss et al., 2009].

## 2.4.2. Funcionamiento del Algoritmo de Ullman

Suponiendo que se quiere saber si un grafo $P$ comparte un (sub)isomorfismo con otro grafo $T$, el algoritmo de Ullmann [1976] enuncia que es posible codificar un (sub)isomorfismo como una matriz $M$ de $|Vp| \times |Vt|$ en la cual cada fila contiene exactamente un $1$ y cada columna contiene como máximo un $1$.

La matriz inicial $M^0_{ij}$ se construye iniciando cada valor en $1$ y luego progresivamente ir transformando los $1$ en $0$ en aquellas posiciones donde se sabe que no puede haber (sub)isomorfismo, por ejemplo donde $grado(Tj) < grado(Pi)$, es decir:

$$M^0_{i,j} = \begin{Bmatrix} 1 \; si \; grado(Tj) \geq grado(Pi) \\ 0 \; en \; el \; resto \; de \; los \; casos \end{Bmatrix} \forall i,j$$

(2.3)



Repositorio Digital Institucional "José María Rosa"
UNLa.

Algoritmo: Ullmann

Entrada: Las matrices de adyacencia de P y T, y la matriz $M^0$

Salida: Todas las matrices M de $v_p$ x $v_t$ que representan sub(isomorfismos)

```
1: M ← M0 ; d ← 1; H1 ← 0; k ← 0; hacia_atras ← verdadero
2: Para i ∈ [1, vp ] Hacer
3:    Fi ← 0;
4: Fin para
5: M1 ← M 0
6: Mientras d = 0 Hacer
7:    hacia_atras← verdadero
8:    Si (∃j : j > k ∧ md,j = 1 ∧ Fj = 0) Entonces
9:       hacia_atras ← falso
10:      Repetir
11:          k ←k+1
12:      Hasta md,k = 1 ∨ Fk = 1
13:      ∀j = k : md,j ← 0
14:      Si [refinar (M, P, T ) ] Entonces
15:          Si d < vp Entonces
16:             Hd ← k; Fk ← 1; d ← d + 1; k ← 0; Md ← M
17:          Sino
18:             Si ci,j = pi,j ∀i, j ∈ [1, vp ] Entonces
19:                almacenar M
20:             Fin Si
21:             M ← Md
22:          Fin Si
23:      Sino
24:          M ← Md
25:      Fin Si
26:   Fin si
27:   Si hacia_atras Entonces
28:      Fk ← 0; d ← d – 1;
29:      Si d > 0 Entonces
30:         M ← Md ; k ← Hd;
31:      Fin Si
32:   Fin Si
33: Fin Mientras
```

**Algoritmo 2.1.** *Pseudocódigo del algoritmo de Ullman*

Adicionalmente el algoritmo de Ullmann aplica un filtro de dos niveles para recortar el espacio de búsqueda. Este se basa en la observación de que si se considera que un vértice $p \in P$ puede llegar a ser parte de un isomorfismo con un vértice $t \in T$ entonces también debería ser posible mapear a todos los vértices *p1...pn* adyacentes a *p* con vértices adyacentes a *t* (siempre comparando el grado de los vértices), si esto no llegase a ser posible entonces se puede decir que el supuesto isomorfismo entre los vértices *p* y *t* es erróneo y se puede cambiar el *1* que lo representa en *M* por un *0.* Formalmente:



Repositorio Digital Institucional "José María Rosa"
UNLa.

$$V_k(P_{i,j} = 1 \exists p\big(M_{k,p} T_{p,j} = 1\big))$$
$$Establecer\ M^0_{i,j} = 0\ donde\ la\ condición\ no\ se\ cumpla$$

(2.4)

La forma de comprobar si $M$ contiene un (sub)isomorfismo es generando una matriz $C$ tal que:

$$C = M(MB)^T$$

(2.5)

Donde además se cumple que:

$$P = 1\ \rightarrow C = 1\ \forall i, j$$

(2.6)

El algoritmo trabaja enumerando sistemáticamente todas las matrices $M$ posibles que pueden ser obtenidas removiendo todos excepto un *1* por cada fila y dejando como máximo un *1* en cada columna (si durante este proceso se llegasen a eliminar todos los *1* de una fila el proceso puede ser detenido, ya que no hay forma de que $M$ pueda contener un (sub)isomorfismo). Luego se comprueba si $M$ contienen un sub(isomorfismo) a través de (2.5) y (2.6).



Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# 3. DESCRIPCIÓN DEL PROBLEMA

En este capítulo se realiza la descripción del problema abordado en este Trabajo Final de Licenciatura. En la sección 3.1 se explica cuál fue el problema de investigación identificado. En la sección 3.2 se analizan los motivos que hacen de este un problema vigente. Finalmente, en la sección 3.3 se traza un sumario de investigación.

## 3.1. IDENTIFICACIÓN DEL PROBLEMA DE INVESTIGACIÓN

Los grafos son una herramienta ampliamente utilizada para el reconocimiento de patrones estructurales [Yan et al., 2005; Gallagher, 2006; Zhu et al., 2011]. Sin embargo, a pesar de su potencial, su uso se encuentra muchas veces limitado debido a la gran complejidad computacional que suponen los algoritmos que operan sobre ellos. Este trabajo en particular se enfoca en los algoritmos de búsqueda de (sub)isomorfismo en grafos, cuya complejidad computacional se sabe que es exponencial en el peor caso.

La forma más simple de encontrar subgrafos isomorfos se reduce a un algoritmo basado en la fuerza bruta que recorra el grafo en forma de árbol en profundidad. Este acercamiento puede llegar a resultar útil para grafos pequeños o de pocos vértices, pero a medida que el grafo crece en tamaño, el número de permutaciones sobre las que hay que buscar se vuelve inmanejable. Por ejemplo, encontrar un subgrafo isomorfo de 20 vértices en un grafo mayor de 30 vértices tiene $7.3 \times 10^{25}$ permutaciones, mientras que encontrar uno de 200 vértices en otro de 300 vértices, tiene alrededor de $3.2 \times 10^{456}$ permutaciones [Dahm et al., 2012]. Tiendo en cuenta estas cifras, es fácil pensar que se requieren técnicas más complejas que permitan reducir sustancialmente el campo de búsqueda en función de tener algún resultado favorable para grafos que superen un determinado tamaño considerable.

## 3.2. PROBLEMA ABIERTO

El interés por los problemas de búsqueda de subgrafos isomorfos ha venido incrementándose durante las últimas décadas. Técnicas de este tipo han sido aplicadas en el área de la quimioinformática para encontrar similitudes entre componentes químicos a partir de su fórmula estructural [Rahman et al., 2009; Ohlrich et al., 1993].



Repositorio Digital Institucional "José María Rosa"
UNLa.

Entre otros de sus usos destacados se sitúa el descubrimiento de patrones en bases de datos, que consiste en encontrar copias de un grafo $G_1$ en un grafo de mayor tamaño $G_2$ [Kuramochi y Karypis, 2001]. En el área de la bioinformática se utiliza para estudiar redes biológicas [Ferro et al., 2007] como las interacciones en redes proteína - proteína [Przulj et al., 2006].
Dado el carácter extremadamente flexible de los grafos para modelar estructuras complejas, y el creciente interés por las redes sociales, las técnicas de búsqueda de (sub)isomorfismo también son aplicadas frecuentemente para estudiar las relaciones en grupos humanos [Snijders et al., 2006].
En el campo de la electrónica y los micro-controladores, los grafos son utilizados para describir los circuitos electrónicos, los métodos de búsqueda de subgrafos ayudan a optimizar la estructura lógica de los mismos [Ogras y Marculescu, 2005; Ohlrich et al., 1993].
De lo expuesto anteriormente salta a la vista que encontrar formas más eficientes de explotar el potencial que tienen los grafos; permanece en la actualidad como un problema abierto.

## 3.3. SUMARIO DE INVESTIGACIÓN

De lo anteriormente expuesto surgen las siguientes preguntas de investigación:

Pregunta 1: ¿Es posible desarrollar algún nuevo algoritmo de búsqueda de (sub)grafos isomorfos que utilice heurísticas diferentes, permitiendo así reducir el factor de ramificación sobre el espacio de búsqueda y encontrar resultados más eficientemente?

Pregunta 2: ¿Es posible desarrollar un método que permita dividir el espacio de búsqueda por un factor elegido a discreción, de forma de posibilitar la paralelización de la búsqueda, asignando los fragmentos resultantes a procesos independientes para maximizar así el uso de los recursos de cómputo disponibles?

En los siguientes capítulos se proponen soluciones a los interrogantes planteados y sus correspondientes validaciones.



Repositorio Digital Institucional "José María Rosa"
UNLa.

# 4. SOLUCIÓN

En este capítulo se presentan las aportaciones realizadas en el marco de este Trabajo Final de Licenciatura. En la sección 4.1 se abordan cuestiones generales y se confecciona un primer acercamiento a las soluciones propuestas. En la sección 4.2 se presenta el nuevo algoritmo G-Matrix, se explica su lógica de funcionamiento y las partes que lo componen. En la sección 4.3 se presentan la Metodología de Paralelización y División del Espacio de Búsqueda sobre algoritmos de (sub)isomorfismo en grafos.

## 4.1. INTRODUCCIÓN A LA SOLUCIÓN

Con el objetivo de abordar el sumario de investigación en la sección 3.3 se confeccionan dos acercamientos de solución. Por un lado se desarrolla un nuevo algoritmo de búsqueda de (sub)isomorfismo en grafos, el cual fue denominado G-Matrix por el uso intensivo de matrices que hace. Y por el otro, se desarrolló una metodología para dividir el espacio de búsqueda sobre el cual operan los algoritmos, lo que se complementa con una estrategia para poner en marcha la ejecución en paralelo.

Los fundamentos de G-Matrix están fuertemente ligados a los del algoritmo de Ullman, y pone en práctica los mismos principios básicos que subyacen en este último. No obstante, G-Matrix es un algoritmo independiente que propone una nueva forma de buscar (sub)isomorfismo en grafos. Mientras Ullman depende solo de una función para recortar el espacio de búsqueda al inicio del proceso y luego aplicar fuerza bruta hasta dar con la solución, G-Matrix hace un acercamiento por *backtracking* aplicando heurísticas que permiten mejorar sustancialmente los tiempos de ejecución.

Por otro lado, como es habitual en este tipo de problemas, se estudian mecanismos de paralelización sobre los algoritmos mencionados y se propone una metodología para dividir el espacio de búsqueda que puede ser fácilmente aplicada tanto en Ullman como en G-Matrix.

## 4.2. ALGORITMO G-MATRIX: APORTACIÓN DE ESTE T.F.L

En esta sección se presenta el nuevo algoritmo G-Matrix para búsqueda de (sub)isomorfismo en grafos. En la sección 4.2.1 se explica las bases del algoritmo y la lógica de su funcionamiento en contraste con el algoritmo de Ullman. En la sección 4.2.2 se profundiza en las explicaciones de cada



Repositorio Digital Institucional "José María Rosa"
UNLa.

proceso de G-Matrix de forma detallada. Por último, en la sección 4.2.3 se presenta el algoritmo G-Matrix descripto en pseudocódigo.

### 4.2.1. Lógica de G-Matrix: cómo funciona (comparado con Ullman)

G-Matrix es un algoritmo de búsqueda de (sub)grafos isomorfos, es decir busca un grafo **G** dentro de otro **G’** de mayor tamaño en termino de vértices y aristas (ver sección 2.2.1).
La lógica que utiliza G-Matrix, es decir, los conceptos en los que éste se basa para funcionar guarda muchas similitudes con el algoritmo de Ullman. No obstante, la mecánica que le confiere a G-Matrix su capacidad para encontrar (sub)isomorfismos rápidamente dista mucho de la de este. Mientras Ullman aplica una metodología casi exclusivamente basada en el procesamiento por fuerza bruta, G-Matrix aprovecha más eficientemente la topología de los grafos (representados en matrices) y las variables que intervienen en el proceso para generar una estructura de apoyo que dota al algoritmo con la capacidad de recortar el espacio de búsqueda, permitiéndole así, dirigir los esfuerzos de ejecución hacia caminos que tengan más oportunidades de conformar (sub)Isomorfismos, a la vez que omite buscar por caminos que se sabe que no producirían ningún resultado satisfactorio.
Al igual que Ullman, G-Matrix empieza por representar los grafos de entrada (grafo “objetivo y grafo “contenedor) en matrices cuadradas. A partir de estos genera una matriz $M\ |V_G|\ x\ |V_{G'}|$ donde $G$ es igual al grafo “objetivo” y $G'$ es igual al grafo “contenedor” donde además todo $M_{i,j} = 1$.
Siguiendo con las similitudes, se inicia un proceso de refinamiento de $M$ de dos niveles. En el primer nivel se analiza el posible *matching* de cada vértice de $G$ con cada vértice de $G'$ solo teniendo en cuenta el grado de los vértices involucrados, nótese que si un vértice de $G$ tiene por ejemplo grado cuatro, solo será posible vincularlo con algún vértice de $G'$ que posea también grado cuatro o superior, si el grado del vértice de $G'$ es inferior al grado del vértice de $G$ entonces no es posible relacionar dichos vértices en un contexto donde estos conformen un isomorfismo, esta incompatibilidad se ve reflejada en $M$ colocando un $0$ en la relación correspondiente a los vértices evaluados. Formalmente:

$$M_{i,j} = \begin{Bmatrix} 1\ si\ grado(G'_j) \geq grado(G_i) \\ 0\ en\ el\ resto\ de\ los\ casos \end{Bmatrix} \forall i,j \tag{4.1}$$

En el segundo nivel, se observa un paso más adelante y también se compara el grado de los vértices adyacentes a aquellos que se busca *matchear.* Si se quiere *matchear* por ej: el vértice $i$ de $G$ con el



Repositorio Digital Institucional "José María Rosa" UNLa.

vértice *j* de $G'$ entonces los grados de los vértices adyacentes a *i* deben ser menores o iguales a los grados de los vértices adyacentes a *j*. Si esta condición no se cumple entonces se puede descartar con seguridad el posible *match* entre *i* y *j*.

$$\forall k(G_{i,k} = 1 => \exists p(M_{k,p} \wedge G'_{p,j} = 1)) \tag{4.2}$$

En este punto ya se tiene una matriz *M*, donde cada *i,j = 1* codifica un posible (pero no seguro) isomorfismo entre un vértice *i* de *G* y un vértice *j* de *G'*. El objetivo tanto de Ullman como de G-Matrix se centra en encontrar una matriz *M* ($M^t$ en G-Matrix, ver *fórmula 4.4* y *figura 4.2*) en la cual cada fila contiene exactamente un *1* y cada columna contiene como máximo un *1* (en el caso de G-Matrix la matriz *M* se trabaja de forma transpuesta, pero es en escancia la misma matriz).

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 |

**Figura 4.1.** *Ejemplo de una matriz M que codifica un sub-Isomorfismo*

Es hasta este punto donde llegan las similitudes entre G-Matrix y Ullman. A partir de aquí Ullman comienza un proceso de fuerza bruta, probando cada combinación posible de *M* que cumpla las restricciones antes mencionadas, luego de encontrar una combinación comprueba que está conforme un isomorfismo verificando que se cumpla:

$$C_{i,j} = 1 \rightarrow G_{i,j} = 1, \forall\ i,j.\ Donde\ C = M(MG')^t \tag{4.3}$$

Mientras tanto G-Matrix realiza un paso previo a comenzar con la búsqueda, que consisten en crear una lista de pares *{i,j}* a partir de *G* con el fin de ordenar el flujo de la ejecución, determinando el camino de búsqueda que debe seguir el algoritmo (ver 4.2.2.2).

A partir de aquí G-Matrix también inicia su proceso más importante, pero no lo hace aplicando puramente fuerza bruta sino más bien intenta construir $M^t$ basándose en un proceso inverso que utiliza las otras matrices *G* y $G'^t$ (ver *fórmula 4.4* y *figura 4.2*) como apoyo, esto permite recortar el



Repositorio Digital Institucional "José María Rosa"
UNLa.

espacio de búsqueda y reducir notablemente los tiempos de ejecución justamente sobre el bucle principal, y hace de G-Matrix una variante más eficiente que su antecesor. En la *tabla 4.1* se describen las similitudes y diferencias entre estos dos algoritmos:

| Ullman | G-Matrix |
|---|---|
| Construye matrices cuadradas de los grafos $G$ y $G'$ | Construye matrices cuadradas de los grafos $G$ y $G'$ |
| Genera una matriz $M$ de $\|V_G\|$ $x$ $\|V_{G'}\|$ | Genera una matriz $M$ de $\|V_G\|$ $x$ $\|V_{G'}\|$ |
| Filtra $M$ por grado de vértices en primer y segundo nivel | Filtra $M$ por grado de vértices en primer y segundo nivel |
| No aplica | Genera una lista de pares ordenados *{i,j}* a partir de $G$, la cual determina el flujo de ejecución del algoritmo |
| Busca (sub)isomorfismo en $M$ por fuerza bruta | **Busca (sub)isomorfismo en $M^t$ aplicando un proceso inverso que se apoya en los grafos $G$ y $G'^t$** |
| Verifica cada combinación de $M$ encontrada en busca de la presencia de (sub)Isomorfismo | No es necesario, si encuentra un $M^t$ siempre contiene un (sub)Isomorfismo |

**Tabla 4.1.** *Similitudes y diferencias entre Ullman y G-Matrix*

Como se observa en la *tabla 4.1* el valor agregado que ofrece G-Matrix radica principalmente en la metodología usada para encontrar (sub)isomorfismo en $M^t$, la misma funciona de la siguiente manera:

Se sabe que existe (sub)isomorfismo entre un grafo $G$ y un grafo $G'$ si se cumple (4.3). G-Matrix se basa en la búsqueda inteligente de $M^t$ para ello se comienza reformulando $C$ aplicando las propiedades algebraicas de las matrices, se llega a la expresión equivalente:

$$M(MG')^t = MG'^tM^t \;\rightarrow\; C = MG'^tM^t \tag{4.4}$$

De esa manera se consigue eliminar el paréntesis $(MG')^t$ y nos permite trabajar cada matriz de forma separada. Note que $C$ y $G$ tienen las mismas dimensiones. Como se busca un



Repositorio Digital Institucional "José María Rosa"
UNLa.

(sub)Isomorfismo, se presupone que *C* contiene al menos todos los $i,j = 1$ que contiene *G*. Esto nos posibilita construir $M^t$ de atrás hacia adelante empezando por *G*:

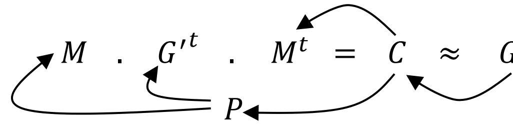



**Figura 4.2.** *Construcción de $M^t$ de atrás hacia adelante*

Las flechas en la *figura 4.2* indican proveniencia, por ejemplo *C* proviene de la multiplicación de *P* por $M^t$, y *P* proviene de la multiplicación de *M* por $G'^t$.

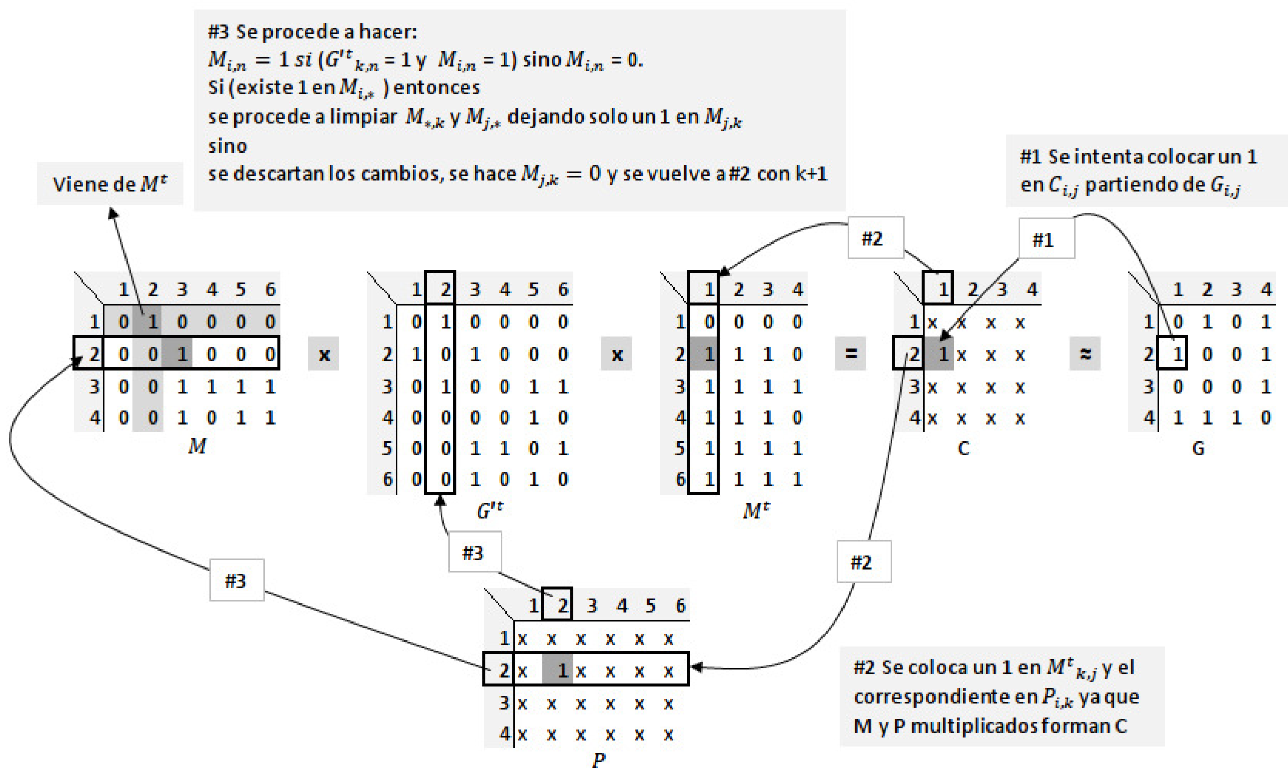



**Figura 4.3.** *Vista general de la lógica de G-Matrix explicada con matrices*

Se determina cual $G_{i,j} = 1$ se quiere colocar primero en *C* y a partir de este se comienzan a construir las demás matrices de forma tal que los valores colocados en estas validen la igualdad (4.4). Si bien la *figura 4.2* da entender un proceso que parece *straight-forward* la realidad es que no se puede saber si un valor colocado en una determinada matriz es correcto (correcto en el sentido de que forme parte de un (sub)isomorfismo) hasta que no se concluya con todo el circuito, por ende en cada salto hay varios valores posibles para colocar en la matriz siguiente, esto establece una



Repositorio Digital Institucional "José María Rosa"
UNLa.

estructura arbolea y G-Matrix aplica una estrategia de búsqueda en profundidad para dar con la combinación correcta. La *figura 4.3* otorga una vista más amplia de este proceso:
Observe que en estos ejemplos se está incluyendo una matriz $P$ en el flujo del algoritmo, esto se hace únicamente con fines didácticos y funciona como paso intermedio para lograr una mejor compresión de la mecánica del algoritmo. En la práctica, el uso de la matriz $P$ es opcional.

## 4.2.2. Explicación detallada del funcionamiento de G-Matrix

En esta sección se procese a hacer una explicación más específica de cómo funciona G-Matrix, su estructura principal y las partes que lo componen. En la sección 4.2.2.1 se detalla el flujo de ejecución de G-Matrix y se explican las variables en las que este se apoya. En la sección 4.2.2.2 se profundiza sobre la metodología para generar la matriz $M^t$ con la que arranca el proceso. En la sección 4.2.2.3 se explica cómo se crea una lista de ejecución cuyo objetivo es ordenar el proceso de reducción de $M^t$. Finalmente, en la sección 4.2.2.4 se describe el proceso por cual se logra reducir $M^t$ en cada iteración hasta hallar un (sub)isomorfismo (si lo hubiera).

### 4.2.2.1. Flujo de ejecución del algoritmo

El flujo principal de G-Matrix, donde se determina qué caminos seguir para buscar (sub)Isomorfismos (dentro de la matriz $M^t$) y qué caminos obviar, funciona como una extensión de un algoritmo de búsqueda en profundidad. El orden de su ejecución se apoya en una lista que se generara a partir de la matriz $G$ (ver 4.2.2.3). En cada paso el algoritmo guarda o restaura el estado de las variables según se esté descendiendo o ascendiendo en el árbol de búsqueda.

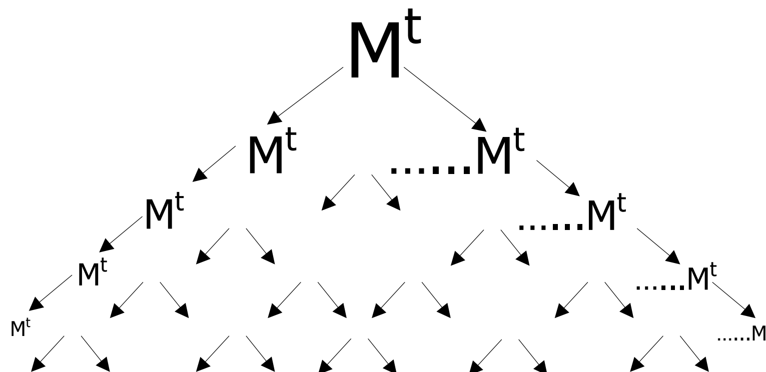



**Figura 4.4.** *Representación del árbol de búsqueda de $M^t$. Nótese que no tiene que ser necesariamente binario.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

En la *figura 4.4* se puede ver la estructura del árbol de búsqueda de $M^t$, en esencia se van realizando operaciones sobre la matriz $M^t$ hasta encontrar aquella variación que satisfaga todos los criterios buscados.

Al igual que otros algoritmos de búsqueda en profundidad, el flujo de G-Matrix se encarga de dar respuesta a tres preguntas principales que se resumen en: 1) Si se debe ascender o descender; 2) Si ya se encontró el elemento buscado; y 3) Si el elemento buscado efectivamente no existe y se procede a terminar la ejecución.

La *figura 4.5* muestra un diagrama del flujo de ejecución de G-Matrix en más detalle, la explicación de los procesos individuales que aparecen nombrados en dicho diagrama se pueden encontrar en las secciones 4.2.2.2, 4.2.2.3 y 4.2.2.4 respectivamente.

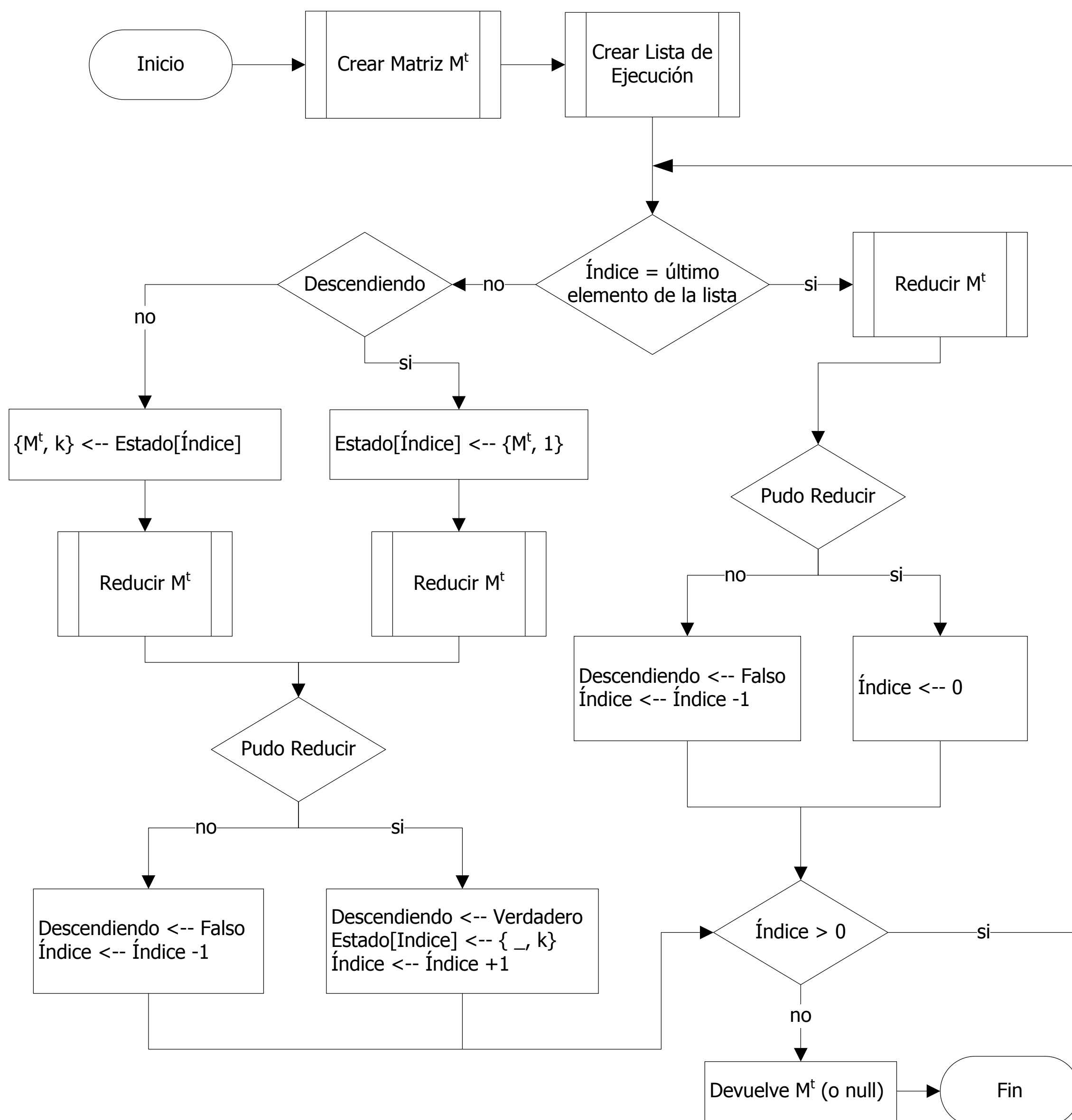



**Figura 4.5.** *Flujo de ejecución de G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Para el mejor entendimiento de la *figura 4.5* es oportuno mencionar que:

*Índice:* representa la posición en la lista de ejecución sobre la que está actualmente trabajado el algoritmo. Si índice alcanza el valor cero significa que debe terminar la ejecución.

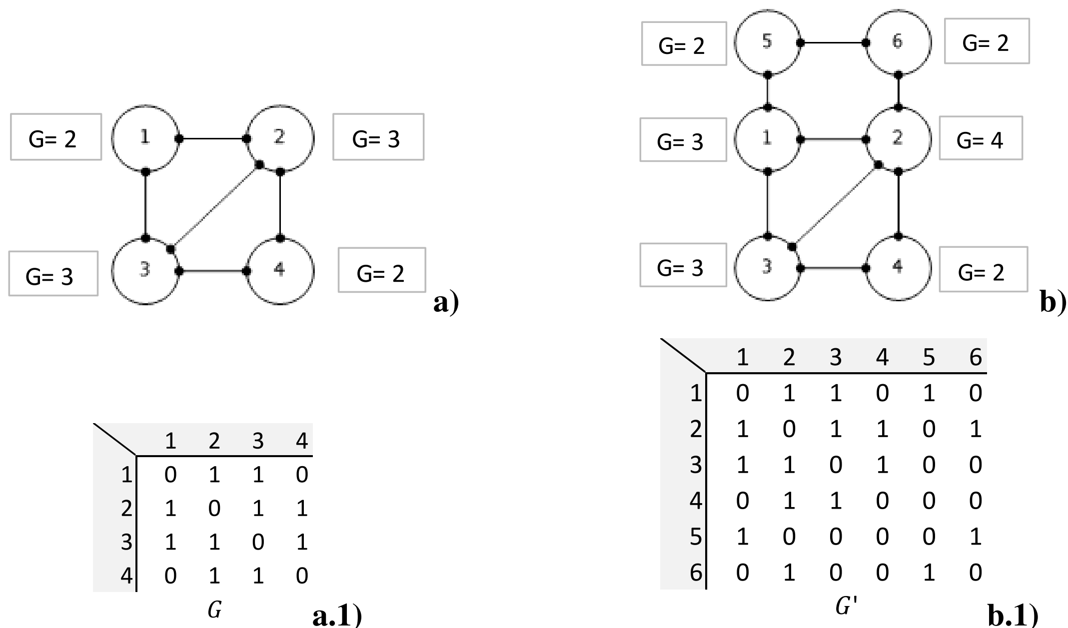
*Estado*: es un arreglo que guarda el valor las variables $M^t$ y $k$ en cada nivel de ejecución para luego estos poder ser retomados en caso de que se esté ascendiendo en el árbol de búsqueda y poder continuar exactamente en el punto donde se abandonó la última vez. De lo anterior se desprende que la cantidad máxima de elemento que puede contener *Estado* es igual a la cantidad de elementos en la lista de ejecución.

*k:* indica la posición exacta donde se deja la ejecución del nivel actual al momento de pasar al nivel siguiente, de forma tal que en caso de ascender se pueda retomar exactamente desde donde se dejó.

Se encuentra disponible una representación en pseudocódigo de este proceso en la sección 4.2.3 *Algoritmo 4.1.*

#### 4.2.2.2. Creación de la matriz inicial $M^t$

El primer paso antes de crear la matriz $M^t$ es generar las matrices de adyacencia de los grafos $G$ y $G'$. Para esta explicación se utilizaran un par de grafos de ejemplo.




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 0 | 0 | 0 | 1 |
| 6 | 0 | 1 | 0 | 0 | 1 | 0 |

**Figura 4.6.** ***a)*** *Grafo G.* ***a.1)*** *Matriz de adyacencia de G.* ***b)*** *Grafo G'.* ***b.1)*** *Matriz de adyacencia de G'*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Al costado de cada vértice se encuentra una etiqueta del tipo *"$G = x$"* que representa el grado de cada vértice en cuestión, esto será útil posteriormente en el proceso de creación de $M^t$.

Los grafos utilizados en este ejemplo y sus respectivas representaciones en matrices de adyacencia se pueden ver *figura 4.6.*

Si bien en la práctica este proceso se puede hacer en un solo bloque de código para este ejemplo se van a ver versiones sucesivas de $M$ hasta llegar a su versión final $M^t$.

Las dimensiones de la matriz $M$ corresponden al número de filas de $G$ por el número de columnas de $G'$. Y como aún no se comienza con el filtrado, en cada $M_{i,j}$ se coloca el valor $1$ que en principio representa que puede existir una correlación entre el vértice $i$ de $G$ y el vértice $j$ de $G'$. La *figura 4.7* describe lo anterior:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figura 4.7.** *Ej de primera instancia de M, sin ningún filtro*

A continuación se procede a hacer un primer refinamiento de $M$. En esta instancia se compara el grado de cada vértice $i$ de $G$ contra el grado de cada vértice $j$ de $G'$. Si $grado(G_i) > grado(G'_j)$ entonces se coloca un $0$ en $M_{i,j}$. Por ejemplo si se miran los grafos de las *figuras 4.6.a* y *4.6.b* se ve que $G_2$ tiene grado 3 mientras que $G'_4$ tiene grado 2 lo que nos permite colocar un $0$ en $M_{2,4}$ este proceso se repite hasta completar $M$. En la *figura 4.8* se ve cómo quedaría $M$ después de este primer refinamiento:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figura 4.8.** Ej *de segunda instancia de M, con filtro de primer nivel*

Ahora se procede a hacer un segundo refinamiento pero esta vez mirando el grado de los vértices adyacentes a aquellos que se quiere comparar. Para cada vértice $i$ de $G$ el grado de los vértices adyacentes a $i$ deben ser mayores o iguales al grados de los vértices adyacentes del vértice $j$ de $G'$. Por ejemplo si se miran los grafos $a$ y $b$ de la *figura 4.6* se ve que el vértice 1 de $G$ tiene dos vértices



Repositorio Digital Institucional "José María Rosa"
UNLa.

adyacentes de grado 3 cada uno mientras que en el vértice 5 de $G'$ tiene dos vértices pero uno es de grado 2 y el otro de grado 3, esto nos permite asegurar que una no hay una relación de (sub)isomorfismo entre 1 de $G$ y 5 de $G'$ por lo que se puede colocar un *0* en $M_{1,5}$ con total seguridad. Este proceso se repite hasta completar $M$. La *figura 4.9* muestra cómo queda $M$ después de este proceso:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 1 | 0 | 0 |

$M$

**Figura 4.9.** Ej *de M inicial terminada, con filtro de segundo nivel*

Finalmente se transpone $M$ para su posterior procesamiento en el algoritmo:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |

$M^t$

**Figura 4.10.** *Ej de* $M^t$ *inicial*

Se encuentra disponible una representación en pseudocódigo de este proceso en la sección 4.2.3 *Algoritmo 4.2.*

### 4.2.2.3. Creación de la lista que determina el orden de ejecución

La ejecución de G-Matrix sigue un camino que se determina a través de una lista de ejecución, la misma sirve para evitar que G-Matrix busque por caminos que generen demasiadas bifurcaciones y en consecuencia la eficiencia del algoritmo se vea afectada. En otros términos podría decirse que esta lista determina una estrategia de búsqueda de alto nivel. La lista de ejecución guarda elementos del tipo *{i,j}* donde $i$ y $j$ representan las coordenadas de cada valor *1* dentro de la matriz $G$. El proceso se divide en dos partes, primero se codifica $G$ en forma de lista siguiendo un orden secuencial, y en una segunda instancia se ordenan los elementos de la lista para que respondan a la



Repositorio Digital Institucional "José María Rosa"
UNLa.

estrategia de ejecución planeada. En la *figura 4.11* se observa una matriz de $G$ y una primera lista obtenida a partir de este.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 |

$G$ **a)**

$$Lista = \{1,2\}\{1,3\}\{2,1\}\{2,3\}\{2,4\}\{3,1\}\{3,2\}\{3,4\}\{4,2\}\{4,3\}$$ **b)**

**Figura 4.11.** ***a)*** *Matriz de adyacencia de G.* ***b)*** *Lista sin orden de G*

El primer paso antes de ordenar la lista es determinar cuál será su primer elemento. El primer elemento de la lista debe seguir la forma *{x,j}* donde $x$ puede ser cualquier valor, y $j$ debe corresponderse con el N° de la columna de $M^t$ que tenga la menor cantidad de filas activas (es decir filas con valor *1*). En la *figura 4.12* se muestra un ejemplo de una matriz $M^t$ con su columna minina señalada:

*N° de columna con la mínima cantidad de filas activas*

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 |
| | = | = | = | = |
| *Total:* | 4 | 3 | 3 | 4 |

$M^t$

**Figura 4.12.** Matriz $M^t$ señalando la columna con menor cantidad de filas activas

Hacer esto es importante porque así se asegura que el algoritmo comienza a ejecutar sobre la variante que genera menor cantidad de bifurcaciones.

Finalmente se procede a ordenar la lista, para esta tarea se deben tener en cuenta los siguientes criterios:

El primer elemento debe tener en $j$ el valor obtenido en el paso anterior, por ejemplo en la *figura 4.12* se ve que el número de la columna mínima es igual a 2 por ende nuestro primer elemento de la



Repositorio Digital Institucional "José María Rosa"
UNLa.

lista debe ser del tipo *{i,2}.* También se debe respetar que el *j* de cada elemento siguiente sea igual al *i* del elemento anterior formando una cadena. Formalmente:

$$j = i \ / \ (j \in E_{n+1}) \wedge (i \in E_n) \tag{5.5}$$

En la *figura 4.13* se puede ver cómo quedaría la lista ordenada:

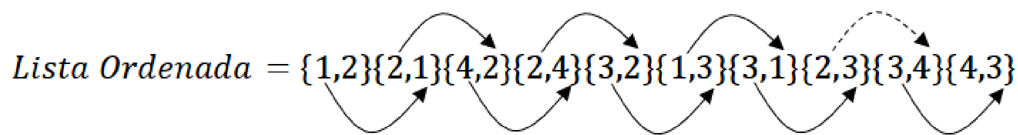



**Figura 4.13.** Lista ordenada de G

Si en algún punto del ordenamiento la condición (5.5) no puede ser satisfecha, pero aún quedan elementos por ordenar simplemente se rompe la cadena, se coloca el primer elemento los que queden por ordenar y se continúa con la misma metodología que se venía utilizando. Por ejemplo en la *figura 4.13* se ve como en el elemento N° 8 la cadena se rompe y simplemente se colocan los elementos restantes empezando una nueva cadena.

Se encuentra disponible una representación en pseudocódigo de este proceso en la sección 4.2.3 *Algoritmo 4.3.*

#### 4.2.2.4. Proceso de reducción de la matriz $M^t$

El proceso de reducción de $M^t$ es el más importante dentro de G-Matrix y es donde transcurre la vasta mayoría del tiempo de ejecución de este algoritmo. Casi toda la lógica de G-Matrix converge en este procedimiento.

Para trabajar, el mismo recibe a modo de parámetros de entrada la matriz $M^t$, la matriz $G^t$, un par de valores *{i, j}* que le dicen sobre que columnas de $M^t$ trabajar (aclaración: *i* y *j* en este caso representan columnas) y un valor *k* que indica desde que fila de $M^t$ se debe comenzar a buscar.

Este procedimiento tiene por objetivo determinar si el camino que se viene siguiendo en el flujo de G-Matrix (ver 4.2.2.1 ) tiene posibilidades de continuar, en cuyo caso se encarga de proponer los ramales por los que se puede seguir a la vez que reduce $M^t$ estrechando el árbol de búsqueda; o si definitivamente ya no quedan variantes posibles, en cuyo caso el proceso se encarga de notificar a



Repositorio Digital Institucional "José María Rosa"
UNLa.

G-Matrix que la rama está agotada, para que este otro suba un nivel y continúe la búsqueda por algún camino adyacente o termine la ejecución.

Para lograr lo anterior el proceso se vale de una metodología de selección y reducción. En primera instancia comienza recorriendo la columna $j$ de $M^t$ a partir del valor que viene dado por la variable $k$ (note que si es la primera vez que se recorre una rama determinada, $k$ será igual a $1$, pero si se está retrocediendo en el árbol de búsqueda, $k$ indicará la posición en la que se abandonó el nivel la vez anterior). Cuando encuentra un $M^t_{k,j} = 1$ realiza un "Y lógico" entre la columna $k$ de $G'^t$ y la columna $i$ de $M^t$ poniendo los resultados en un vector $V$ (note que independientemente de los valores, la posición $k$ de $V$ siempre será $0$ ya que por construcción solo puede haber un $1$ en cada fila de $M^t$). Sino encontrase ningún $M^t_{k,j} = 1$ devuelve *falso* y termina el proceso.

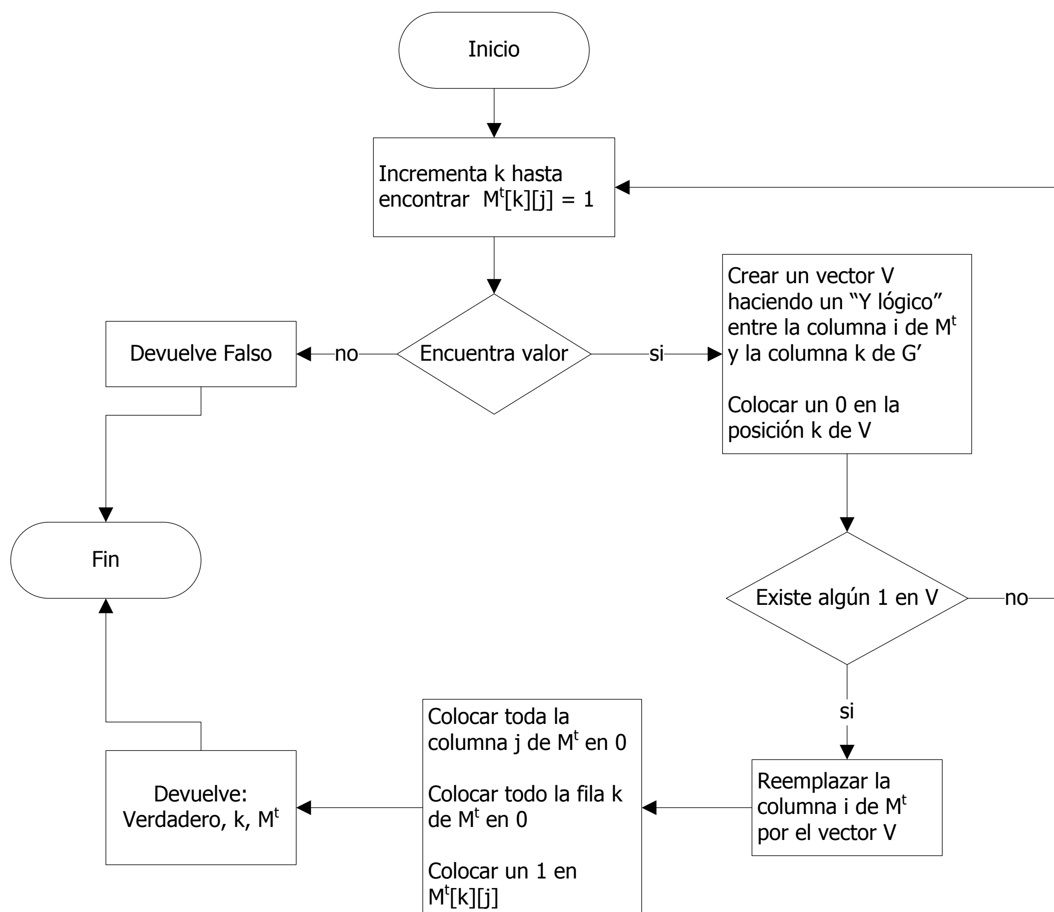



**Figura 4.14.** *Diagrama de flujo del proceso de reducción de $M^t$*

Una vez que se tiene el vector $V$ se debe verificar que este contenga al menos un valor *seteado* en $1$ para poder continuar (de lo contrario se estaría confirmando que el valor $M^t_{k,j}$ elegido no puede formar parte de un (sub)Isomorfismo y se procede a buscar otro $M^t_{k,j} = 1$). Si $V$ cumple la condición



Repositorio Digital Institucional "José María Rosa"
UNLa.

anterior se procede a reemplazar la columna *i* de $M^t$ por el vector *V*, y se reduce $M^t$ *seteando* en *0* todos los valores de la columna *j* de $M^t$ y todos los valores de la fila *k* de $M^t$, solo dejando un *1* en $M^t_{k,j}$. Finalmente se devuelve *verdadero*, *k* y $M^t$.

A en la *figura 4.14* puede se observa la explicación anterior representada en un diagrama de flujo. Para su mejor visualización, en la *figura 4.15* se muestra cómo trabaja el proceso de reducción de $M^t$ puesto en marcha sobre unas matrices ilustrativas.

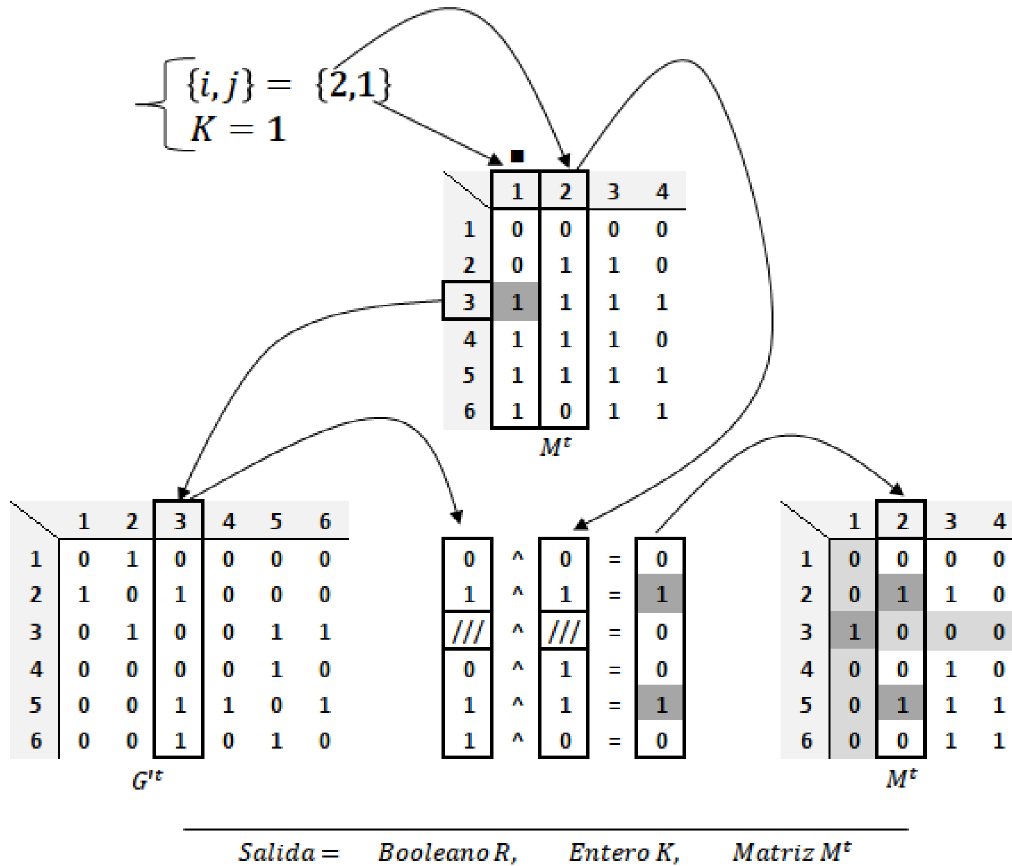



**Figura 4.15.** *Vista del proceso de reducción de* $M^t$ *explicado con matrices. (Los valores de las matrices no responden a un caso real, solo son de carácter ilustrativo)*

El ejemplo que se muestra en la *figura 4.15* comienza buscando en $M^t_{1,1}$, encuentra un valor en $M^t_{1,3}$, toma la columna 3 de $G'^t$ y la columna 2 de $M^t$ y ejecuta un "Y lógico" sobre ellas (note que la fila 3 de ambas columnas están desactivadas, ya que los valores que allí aparezcan son irrelevantes, se cocola un *0* por construcción). Como en el vector resultante quedan dos valores en *1*, se procede a reemplazarlo por la columna 2 de $M^t$ y se *setean* en *0* la columna 1 de $M^t$ y la fila 3 de $M^t$ solo dejando un uno en $M^t_{1,3}$. El proceso termina devolviendo *verdadero*, el valor 3, y la matriz $M^t$.

Se encuentra disponible una representación en pseudocódigo de este proceso en la sección 4.2.3 *Algoritmo 4.4*.



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 4.2.3. Presentación del pseudocódigo del algoritmo G-Matrix

En esta sección se brinda una representación en pseudocódigo del algoritmo G-Matrix con sus respectivos procedimientos.

Se presenta el *algoritmo 4.1* que describe el flujo de ejecución de G-Matrix. Puede encontrar una explicación del *algoritmo 4.1* en lenguaje coloquial en la sección 4.2.2.1.

```
Algoritmo: G-Matrix
Entrada: Grafo_G,   //Grafo
         Grafo_G'   //Grafo
Salida:  M^t o Null // Matriz
1: G[][],G'^t[][],M^t[][] ← Crear_Matriz_Inicial_M^t(Grafo_G,Grafo_G');
2: L[{}] ← Crear_Lista_De_Ejecución(G, M^t);
3: Estado[{}] ← Ø; Indice_L ← 1; K ← 1; Redujo ← Falso; Descendiendo ← Verdadero;
4: Mientras (Indice_L > 0)
5:      Si: (Indice_L < [N° De Elementos En L])
6:          Si (Descendiendo)
7:              Estado[Indice_L] ← {M^t, 1};
8:              {Redujo,M^t,K,} ← Reducir_M^t(M^t, G'^t, L[Indice_L], 1);
9:          Sino
10:             {M^t,K} ← Estado[Indice_L];
11:             {Redujo,M^t,K} ← Reducir_M^t(M^t, G'^t, L[Indice_L], K);
12:         Fin Si
13:         Si (Redujo)
14:             Descendiendo ← Verdadero;
15:             Estado[Indice_L] ← { _, K};
16:             Indice_L++;
17:         Sino
18:             Indice_L--;
19:             Descendiendo ← Falso;
20:         Fin Si
21:     Sino
22:         {Redujo, M^t, _} ← Reducir_M^t(M^t, G'^t, L[Indice_L], 1);
23:         Si (Redujo)
24:             Indice_L ← 0;
25:         Sino
26:             Indice_L--;
27:             Descendiendo ← Falso;
28:         Fin Si
29:     Fin si
30: Fin Mientras
32: Si (Redujo) Retornar M^t; Sino Retornar Ø; Fin Si;
```

**Algoritmo 4.1.** *G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Se presenta el *algoritmo 4.2* que describe el proceso de creación de la matriz $M^t$ inicial. Puede encontrar una explicación del *algoritmo 4.2* en lenguaje coloquial en la sección 4.2.2.2.

**Procedimiento:** Crear_Matriz_Inicial_$M^t$

**Entrada:** Grafo_G, *//Grafo*
Grafo_G' *//Grafo*

**Salida:** G, *//Matriz*
$G'^t$, *//Matriz*
$M^t$ *//Matriz*

```
1: M[N° De Vértices De Grafo_G][N° De Vértices De Grafo_G'];
2: Para Cada Vértice I En Grafo_G
3:      Para Cada Vértice J En Grafo_G'
4:           Si (Grado(I) <= Grado(J))
5:                Para Cada Vértice X Adyacente A I
6:                     Para Cada Vértice Y Adyacente A J
7:                          Si(Grado(X)<= Grado(Y))
8:                               M[I][J] ← Verdadero;
9:                          Sino
10:                              M[I][J] ← Falso;
11:                         Fin Si
12:                    Fin Para
13:               Fin Para
14:          Sino
15:               M[I][J] ← Falso;
16:          Fin Si
17:     Fin Para
18: Fin Para
19: Mt ← Transponer(M);
20: G[][]← Matriz_De_Adyacencia(Grafo_G);
21: G'[][]← Matriz_De Adyacencia(Grafo_G');
22: G'ᵗ ← Transponer(G')
23: Retornar G, G'ᵗ, Mᵗ;
```

**Algoritmo 4.2.** *G-Matrix –Procedimiento: Crear matriz inicial M*

Se presenta el *algoritmo 4.3* que describe el proceso de creación de la lista de ejecución. Puede encontrar una explicación del *algoritmo 4.3* en lenguaje coloquial en la sección 4.2.2.3.

**Procedimiento:** Crear_Lista_De_Ejecución

**Entrada:** G, *//Matriz*
$M^t$, *//Matriz*

**Salida:** L[{}] *//Lista de elementos {i,j}*

**Algoritmo 4.3.a** *G-Matrix –Procedimiento: Crear lista de ejecución*



Repositorio Digital Institucional "José María Rosa"
UNLa.

1: Lista[]←Ø;
2: Lista_Ordenada[] ← Ø;
3: **Para** I ← 1 **Hasta** [N° De Filas De G] **Incremento** 1
4: **Para** J ← 1 **Hasta** [N° De Columnas De G] **Incremento** 1
5: **Si** (G[I][J])
6: Lista[]← {I,J};
7: **Fin Si**
8: **Fin Para**
9: **Fin Para**
10: H ← [N° De La Columna De $M^t$ Con Menor Cantidad De Verdaderos]
11: **Mientras** Lista <> Ø
12: Encuentra ← **Falso**;
13: **Para Cada** {I,J} **En** Lista
14: **Si** (J = H)
15: Lista_Ordenada[] ← {I,J};
16: H = I;
17: **Remover** {I,J} **De** Lista;
18: Encuentra ← **Verdadero**;
19: **Romper Para**
20: **Fin Para**
21: **Si** (^Encuentra)
22: H++;
23: **Fin Si**
24: Encuentra ← **Falso**;
25: **Mientras**
26: **Retornar** Lista_Ordenada;

**Algoritmo 4.3.b** *G-Matrix –Procedimiento: Crear lista de ejecución*

Se presenta el *algoritmo 4.4* que describe el proceso de reducción de la matriz $M^t$. Puede encontrar una explicación del *algoritmo 4.4* en lenguaje coloquial en la sección 4.2.2.4.

**Procedimiento:** Reducir_$M^t$

**Entrada:** $M^t$, *//Matriz*
$G'^t$, *//Matriz*
{I,J}, *//Conjunto de enteros*
K *//Entero*

**Salida:** Redujo, *//Booleano*
$M^t$, *//Matriz*
K *//Entero*

1: Redujo ← **Falso**;
2: **Mientras**( K <= [N° De Filas De $M^t$] **Y** Redujo = **Falso**)
3: **Si**(M[K][J])
4: Aux ← $M^t$[K][I];
5: $M^t$[K][I] ← **Falso**;
6: **Para** N ← 1 **Hasta** [N° De Filas De $M^t$] **Incremento** 1

**Algoritmo 4.4.a** *G-Matrix –Procedimiento: Reducir M*



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
7:                      Si (Mt[N][I] Y Gt[N][K])
8:                           Mt[N][I] ← Verdadero;
9:                           Redujo ← Verdadero;
10:                     Sino
11:                          Mt[N][I] ← Falso;
12:                     Fin Si
13:                Fin Para
14:                Si (Redujo)
15:                     Mt[K][*] ← Falso;
16:                     Mt[*][J] ← Falso;
17:                     Mt[K][J] ← Verdadero;
18:                Sino
19:                     Mt[K][I] ← Aux;
20:                Fin Si
21:           Fin Si
22:     K++;
23:     Fin Mientras
24: Retornar Redujo, Mt, K;
```

**Algoritmo 4.4.b** *G-Matrix –Procedimiento: Reducir M*

# 4.3. METODOLOGÍA DE PARALELIZACIÓN Y DIVISIÓN DEL ESPACIO DE BÚSQUEDA: APORTACIÓN DE ESTE T.F.L

En este capítulo se presentan aportaciones hechas en este Trabajo Final de Licenciatura. En la sección 4.3.1 se explica porqué se toma la decisión de paralelizar los algoritmos vistos. En la sección 4.3.2 se presenta la metodología utilizada para dividir el espacio de búsqueda, pseudocódigo incluido. Finalmente, en la sección 4.3.3 se describe la estrategia puesta en marcha para abordar el problema completo.

## 4.3.1. ¿Por qué paralelizar?

Durante la última década se ha visto como el mercado de microprocesadores ha abandonado el patrón mono-núcleo para dar paso a microprocesadores que contienen múltiples núcleos en su interior, con tendencia a incrementarse a medida que el tiempo transcurre. A la vista de este escenario, no es ilógico pensar en formas de adaptar los algoritmos hacia un modelo que utilice las bondades que este nuevo paradigma tiene para ofrecer. Tradicionalmente los algoritmos de búsqueda de (sub)grafos isomorfos están pensados para trabajar directamente sobre un solo hilo. Pero, si se analiza el espacio de búsqueda sobre el que trabajan dichos algoritmos (ver *figura 4.4*) se puede apreciar que no es necesario que el algoritmo conozca la totalidad del dominio para realizar



Repositorio Digital Institucional "José María Rosa" UNLa.

operaciones sobre un fragmento de este. Como el espacio de búsqueda puede ser conceptualizado en forma de árbol, no resulta difícil intuir una estrategia que divida el árbol en cuestión en ramas más pequeñas y asigne un hilo a cada una de estas. Si a eso se suma el hecho de que es posible hacer coincidir la cantidad de ramas generadas con la cantidad de núcleos que se dispone; hace de este problema en particular, un candidato óptimo para explotar el potencial de este nuevo paradigma de hardware.

La flexibilidad característica del espacio de búsqueda permite generar una gran cantidad de divisiones relativamente equitativas a medida que el mismo aumenta de tamaño, lo que también hace a este problema fácilmente escalable hacia una red de computadores. La metodología que se explora en 4.3.2 y 4.3.3 es lo suficientemente genérica como para ser aplicada a un solo microprocesador multinúcleo o a una red de equipos de cómputo.

## 4.3.2. División del espacio de búsqueda

En esta sección se realiza una descripción del proceso de división del espacio de búsqueda. En la sección 4.3.2.1 se hace un primer acercamiento donde se explican los fundamentos lógicos del proceso. En la sección 4.3.2.2 se profundiza sobre la explicación a través del uso de un ejemplo concreto. Finalmente, en la sección 4.3.2.3 se brinda una representación en pseudocódigo.

### 4.3.2.1. Lógica de la división del espacio de búsqueda: cómo funciona

Como se menciona en la sección 4.3.1 para poder llevar a cabo una efectiva paralelización de procesos, se debe contar con una metodología que nos permita dividir la carga de trabajo de forma tal de poder asignar una cierta cantidad de trabajo a cada proceso individual, procurando que dicha repartición sea lo más equitativa posible.

En el contexto de esta Trabajo Final de Licenciatura se han abarcado algoritmos cuyo espacio de búsqueda viene dado por una matriz $M$ (o $M^t$ según el caso) que configura todas las combinaciones de (sub)isomorfismo "potenciales" y precisamente, tienen como fin indagar sobre $M$ valiéndose de diversas estrategias hasta dar con la combinación deseada. De lo anterior salta a la vista que para lograr repartir la carga de trabajo de los algoritmos estudiados, se debe buscar una manera de dividir el total de combinaciones que ofrece $M$. Si bien la cantidad de combinaciones que se pueden obtener va a depender directamente de las dimensiones de $M$ y de la distribución de sus posiciones activas, se puede pensar este problema de forma genérica para una $M$ de dimensiones $|a|$ x $|b|$ donde cada $i,j = 1$ (ver *figura 4.16*).



Repositorio Digital Institucional "José María Rosa"
UNLa.

| | 1 | 2 | 3 | … | b-1 | b |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| … | 1 | 1 | 1 | 1 | 1 | 1 |
| a-1 | 1 | 1 | 1 | 1 | 1 | 1 |
| a | 1 | 1 | 1 | 1 | 1 | 1 |

**Figura 4.16.** *M de dimensiones |a|x|b| con cada i,j=1*

Teniendo en cuenta las restricciones de construcción de *M*, luego la cantidad de combinaciones viene dada por:

$$Combinaciones = \prod_{i=1}^{a} (b - i + 1)$$

(4.6)

Como la amplitud del espacio de búsqueda viene dada por una multiplicación sucesiva de términos, si se divide uno de esos términos se estará dividiendo la totalidad del espacio de búsqueda por ese factor. Este concepto se aplica para repartir la carga de trabajo entre múltiples procesos, de la siguiente manera: (1) en primera instancia se estima la cantidad de procesos que desea correr (esto puede venir dado por la cantidad de núcleos con los que se cuente, la cantidad de computadores, o una combinación entre ambos). (2) En segunda lugar se determina cual es la fila de *M* que tiene la mayor cantidad de columnas activas (se busca la de mayor tamaño para lograr una división lo más equitativa posible, también funcionaria buscar un valor que sea múltiplo del número de procesos) y se la divide por el número de procesos que se desea correr. (3) Finalmente se crean tantas matrices *M* como divisiones se han podido obtener teniendo cada matriz un fragmento de la fila seleccionada.

En la sección 4.3.2.2 se lee un ejemplo concreto de este proceso y en la sección 4.3.2.3 se brinda una representación en pseudocódigo.

#### 4.3.2.2. Explicación con ejemplo del proceso de división del espacio de búsqueda

En esta sección se profundizan los conceptos abordados en 4.3.2.1 a través de la descripción de un ejemplo concreto en el que se puede observar como funciona el procedimiento por el cual se divide el espacio de búsqueda (las matrices utilizadas son meramente ilustrativas y no responden a un caso real).

Se da por supuesto que se quieren ejecutar 3 procesos en paralelo para buscar (sub)Isomorfimo sobre una matriz *M.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

En primera instacia se debe identificar cual de las filas de $M$ tiene el mayor numero de columnas activas (columnas en *1*), la *figura 4.17* representa lo anterior

| | 1 | 2 | 3 | 4 | 5 | 6 | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | = | 4 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 | = | 2 |
| 3 | 1 | 1 | 0 | 1 | 1 | 1 | = | 5 → *Fila con la máxima cantidad de columnas activas* |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | = | 2 |

**Figura 4.17.** *M con su fila más activa señalada*

Como se ve en la *figura 4.17* la tercera fila tiene 5 posiciones en *1* y es la mayor de todas. A continuación se debe determinar cómo serán repartidos estos *1*. Esto lo se hace creando una lista $D$ cuya longitud sea igual a la cantidad de hilos que se quieren generar y que contenga en cada una de sus posiciones la cantidad de *1* que serán asignados en cada matriz resultante. Como se quieren generar 3 hilos, se procede a hacer una división entera entre el valor de la fila 3 por el número de hilos, en este caso: 5 y 3 respectivamente:

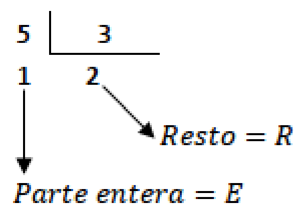



**Figura 4.18.** *División entre la cantidad de 1 en la máxima fila de M y el número de procesos. Parte entera y resto señalados*

Como era de esperarse, en la *figura 4.18* se señala que la división tiene el valor $E$ como parte entera y el valor $R$ como resto. Esto indica que a cada posición de la lista $D$ se debe asignarle el valor $E$ para empezar y que quedan $R$ por repartir. Para repartirlo simplemente se procede a realizar un proceso iterativo donde en cada paso adiciona $+1$ a una posición de $D$ hasta agotar $R$, como en este caso $R$ es igual a 2 solo las dos primeras posiciones de $D$ se ven afectadas. La *figura 4.19* describe este proceso.



Repositorio Digital Institucional "José María Rosa"
UNLa.

$$D = \{1, \quad 1, \quad 1\}$$

$$\downarrow +1 \quad \downarrow +1$$

$$D = \{2, \quad 2, \quad 1\}$$

**Figura 4.19.** *Creación de la lista D*

Casi sobre el final, se debe proceder a fraccionar $M$ y para ello se toma su fila número 3 y se generan 3 vectores a partir de esta, donde a cada uno de los vectores se le asigna una partición de la fila 3 según indique el valor correspondiente de la lista *D*. Finalmente se generan 3 copias de $M$ y se reemplazan sus respectivas filas número 3 por cada uno de los vectores generados. Este proceso se puede visualizar más fácilmente en la *figura 4.20.*

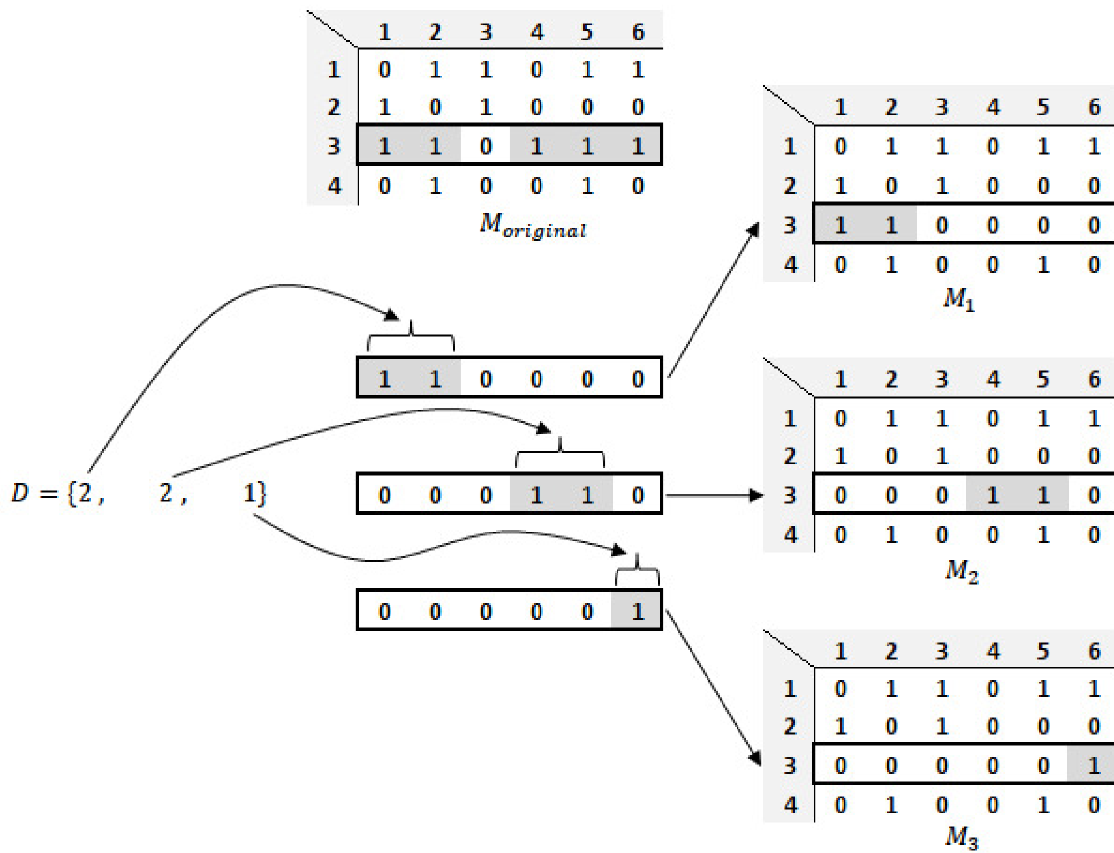



**Figura 4.20.** *Fraccionamiento de M*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Una vez obtenidas las particiones de *M* ($M_1$, $M_2$, $M_3$) solo queda disparar tres instancias del algoritmo elegido asignando una partición a cada uno de ellos para que sean procesadas por separado.

En este caso en particular, la carga de trabajo en cada *M* no quedó repartida equitativamente, teniendo $M_3$ 50% menos de carga, pero a medida que *M* crezca en tamaño, las diferencias entre los fragmentos serán cada vez menos significativas porcentualmente.

Este ejemplo no contempla el caso (poco probable pero posible) donde el número de hilos que se quiere generar es mayor al número de divisiones que se puede obtener de *M*, en cuyo caso sencillamente lo que se hace es generar menos hilos.

En la sección 4.3.2.3 *algoritmo 4.5* se encuentra una representación en pseudocódigo de este proceso, el cual contempla todas las eventualidades que pudieran ocurrir.

### 4.3.2.3. Algoritmo para dividir el espacio de búsqueda

En esta sección se brinda una representación en pseudocódigo de un algoritmo para dividir el espacio de búsqueda

Puede encontrar un ejemplo de este algoritmo explicado en lenguaje coloquial en la sección 4.3.2.2.

**Algoritmo:** Dividir_Espacio_de_Búsqueda

**Entrada:** M *//Matriz M*
d *//cantidad de hilos que se pretende generar*

**Salida:** {$M_1$....$M_n$} *//Conjunto de todas las partes de M que se pudo obtener*

```
1: H ← 0;
2: Para i ← 1 Hasta [N° de filas de M] Incremento 1
3:      t ← total de valores 1 en la fila i de M;
4:      Si (t > H)
5:           f ← i;
6:           H ← t;
7:      Fin Si
8: Fin Para
9: Si (H > d)
10:     E ← parte entera de H / d;
11:     R ← Resto de H /d;
12:     D[d] ← Crear arreglo de longitud d y asignar E en cada posición;
13:     Para i ← 1 Hasta R Incremento 1
14:          D[i] ← D[i]+1;
15:     Fin Para
16: Sino Si (H = d)
17:     D[d] ← Crear arreglo de longitud d y asignar 1 en cada posición;
```

**Algoritmo 4.5.a** *Procedimiento para dividir el espacio de búsqueda.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

18: **Sino**
19: D[H] ← Crear arreglo de longitud d y asignar 1 en cada posición;
20: **Fin Si**
21: Q ← 0;
22: **Para Cada** entero n en D[] **Hacer**
23: $M_x$ ← M;
24: V ← Tomar Fila f de M;
25: Buscar en V los primeros Q 1 y reemplazarlos por 0;
26: Ignorar los próximos n 1;
27: Colocar todo en 0 hasta terminar de recorrer V;
28: Reemplazar la fila f de $M_x$ por V;
29: {} ← {} + $M_{x;}$
30: Q ← Q + n;
31: **Fin para**
32: **Retornar** {};

**Algoritmo 4.5.b** *Procedimiento para dividir el espacio de búsqueda.*

### 4.3.3. Estrategia de Ejecución en paralelo de los algoritmos

Dividir el espacio de búsqueda como se vio en la sección 4.3.2, es solo mitad del trabajo. Para poder llevar a cabo la paralelizarían también se debe contar con una estructura que permita llevar el control de los procesos y facilite la comunicación entre los mismos. (1) Se debe ser capaz de determinar cuántos procesos lanzar en función de la cantidad de núcleos o aparatos de computo involucrados. (2) Se debe contar con mecanismos para detener los procesos en caso de que uno de ellos haya alcanzado el objetivo de forma satisfactoria, y también (3) se debe poder determinar que el (sub)isomorfismo buscado efectivamente no existe, en el caso de que todos los procesos arrojen resultados negativos.
A continuación se mencionan cada uno de estos problemas de forma separada, a la vez que se proponen soluciones:

**Cantidad de hilos a disparar:** Lo más apropiado seria hacer un conteo de cuantos núcleos de cómputo se dispone, ya sea en solo computador o en una red de estos, y disparar una cantidad equivalente de hilos, es decir, una relación 1:1 entre núcleos e hilos.

**Detener ejecución en caso de éxito:** Lo más apropiado para resolver este escenario sería contar con una variable global que todos los hilos puedan leer y modificar. En caso de que uno encuentre un resultado, debe dar señal de esto modificando dicha variable. Los hilos deben chequear el estado



Repositorio Digital Institucional "José María Rosa"
UNLa.

de esta variable en intervalos regulares de tiempo para darse por enterados y actuar en consecuencia.

**Concluir que no existe el (sub)isomorfismo buscado**: El hecho de estar procesando partes del espacio de búsqueda por separado hace imposible que un hilo en particular pueda concluir que el objetivo buscado no existe, ya que dicho objetivo bien puede encontrarse dentro del dominio de otro hilo. Para abordar este inconveniente, una opción viable es mantener un monitor de procesos que siga el rastro de cada hilo y en caso de que ninguno de ellos dé con el objetivo, concluir que efectivamente el objetivo buscado no existe.

En la *figura 4.21* se observa un diagrama flujo que engloba el proceso completo de ejecución en paralelo para su mejor comprensión.

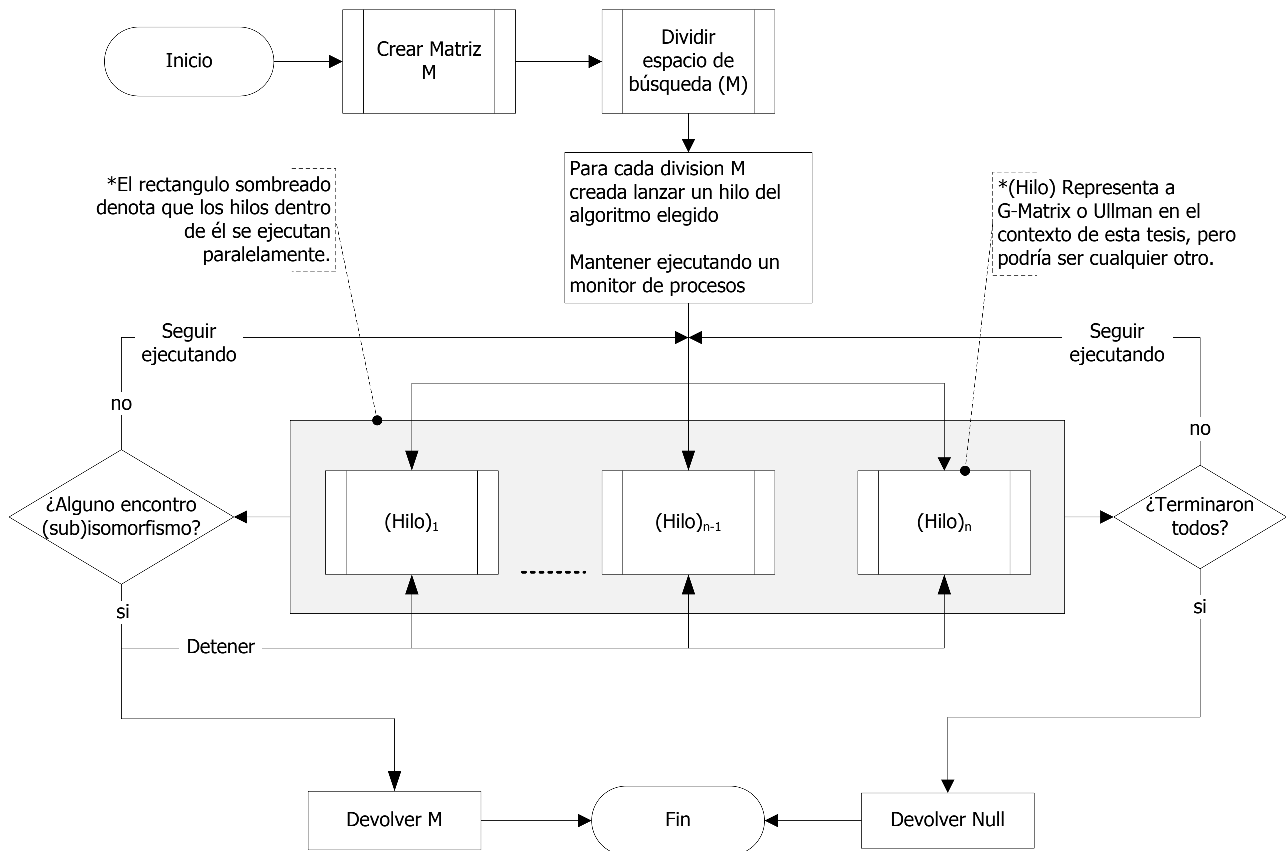



**Figura 4.21.** *Diagrama de ejecución en paralelo*



Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# 5. EXPERIMENTACIÓN

En este capítulo se presentan los resultados obtenidos de la ejecución de los algoritmos de búsqueda de (sub) Isomorfismo en grafos conexos, se explica el algoritmo utilizado para generar los grafos que conforman el set de datos de pruebas, se procede a describir el criterio puesto en marcha para la correcta selección de los parámetros de entrada de cada caso de prueba, finalmente se realiza una comparación de los resultados obtenidos acompañados de las pertinentes observaciones.

En la sección 5.1 se describen características inherentes a la experimentación y se señalan las tecnologías utilizadas durante el proceso. En la sección 5.2 se describe la funcionalidad del generador de grafos sintéticos. En la sección 5.3 se explica la configuración de la tabla de parámetros de entrada siguiendo el método de Montecarlo, se enumeran y explican las fórmulas matemáticas que intervienen en el proceso. En la sección 5.4 se observan los resultados obtenidos a través de la ejecución del algoritmo de Ullman original. En la sección 5.5 se observan los resultados obtenidos a través de la ejecución del algoritmo de Ullman aplicando la metodología de paralelización. En la sección 5.6 se observan los resultados obtenidos a través de la ejecución del algoritmo G-Matrix. En la sección 5.7 se observan los resultados obtenidos a través de la ejecución del algoritmo G-Matrix aplicando la metodología de paralización. Finalmente en la sección 5.8 se procede a hacer una comparación y valoración de los resultados obtenidos.

## 5.1. ENFOQUE DE LA EXPERIMENTACIÓN: GENERALIDADES

Esta experimentación tiene por objetivo recolectar una cantidad suficiente de resultados que permita abrir juicio sobre el desempeño de los algoritmos tratados en este Trabajo Final de Licenciatura con el fin de (a) Confirmar que el algoritmo G-Matrix propuesto mejora significativamente los tiempos de ejecución con respecto al algoritmo de Ullman que es uno de los más utilizados dentro del universo de los algoritmos de búsqueda de (sub) isomorfismo en grafos, y (b) Demostrar que la metodología de paralelización propuesta y las técnicas de división del espacio de búsqueda logran explotar mejor la característica multinucleo de los equipos de cómputo modernos traduciéndose en un mejor rendimiento de los algoritmos comparados con su contracara lineal.

Para llevar a cabo esta experimentación se utilizó la técnica de Montecarlo [Metropolis y Ulam, 1949], cada experimento fue ejecutado diez veces y se promedió el tiempo total de ejecución para disminuir las oscilaciones que pudieran presentarse. A cada experimento se le estableció un *time out* de un minuto, si transcurrido ese tiempo el algoritmo no es capaz de encontrar una solución se detiene el proceso.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Los algoritmos fueron implementados en el lenguaje de programación java utilizando las librerías correspondientes a OpenJDK 7 (ver Anexo).

## 5.2. GENERADOR DE GRAFOS SINTÉTICOS

En esta sección se presenta el proceso de generación de grafos sintético. En la sección 5.2.1 se abordan temas generales del proceso de generación de grafos que permitan búsquedas de (sub) isomorfismo en ellos, se describe la topología de los grafos generados y las estructuras de datos que los soportan. En la sección 5.2.2 se presenta el algoritmo correspondiente al generador de grafos en pseudocódigo. Por último, en la sección 5.2.3 se amplía un ejemplo del proceso completo para su mejor entendimiento.

### 5.2.1. Generalidades

Para poder llevar a cabo la búsqueda de (sub)Isomorfismo es necesario generar un set de datos de grafos donde cada muestra de dicho set debe contar con por lo menos dos grafos, el primero es el grafo "buscado" o grafo "objetivo" $G=(V,E)$ y un segundo grafo $G'=(V',E')$ de igual o mayor tamaño que $G$ en función de su número de vértices y aristas llamado grafo "contenedor", donde este último es igual a la unión de $G$ más un conjunto de vértices y aristas y se denota $G'= G \cup \{V,E\}$.
La estructura de datos que se usó para la representación y manipulación de los grafos fueron las listas de adyacencia (ver 2.2.3).
Las propiedades topológicas de los grafos generados son las siguientes:

- **Conexos:** Se debe de poder encontrar al menos un camino que una dos vértices cualesquiera pertenecientes al mismo grafo.
- **No dirigidos:** Todas las relaciones que unen los vértices son simétricas.
- **Sin relaciones reflexivas:** No se admiten relaciones de un vértice en sí mismo.

### 5.2.2. Algoritmo del generador de grafos sintéticos

En esta sección se presenta el pseudocódigo del algoritmo para generar grafos sintéticos, se explican sus parámetros de entrada y la salida que este produce. El algoritmo solo contempla la creación de un único grafo objetivo más su grafo contenedor, es decir, una sola muestra del set. La reiterativa ejecución de este algoritmo variando sus parámetros de entrada, genera así el set de grafos completo.



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Algoritmo:** Generador de Grafos Sintéticos

**Entrada**: G|V| *"número de vértices del grafo objetivo"*
G|E| *"número de aristas del grafo objetivo"*
G'|V'| *"número de vértices del grafo contenedor"*
G'|E'| *"número de aristas del grafo contenedor"*

**Salida**: {G=(V,E) , G'=(V',E')} *"conjunto formado por el grafo objetivo y el grafo contenedor"*

1: V , E , V' , E' = ∅
2: **Para** i=1 **Hasta** G|V| **Incremento** 1
3: V ← V ∪ crear vértice con id = i
4: **Fin Para**
5: **Para Cada** v∈V
6: E ← E ∪ crear arista $\{v_{id},v_{id+1}\}$
**7: Fin Para**
8: **Para** i=0 **Hasta** G|E| - (G|V| -1) **Incremento** 1
9: {} ← seleccionar aleatoriamente un v∈V
10: {} ← {} ∪ seleccionar aleatoriamente v∈V
11: E ← E ∪ {}
12: **Fin Para**
13: V' ← V
14: E' ← E
15: **Para** i=G|V| **Hasta** G'|V'| **Incremento** 1
16: V' ← V' ∪ crear vértice con id = i
17: **Fin Para**
18: **Para Cada** v∈(V'-V)
19: E' ← E' ∪ crear arista $\{v_{id},v_{id+1}\}$
20: **Fin Para**
21: n ← cantidad de elementos de E'
22: **Para** i=0 **Hasta** G'|E'| - n **Incremento** 1
23: {} ← seleccionar aleatoriamente un v∈(V'-V)
24: {} ← {} ∪ seleccionar aleatoriamente v∈V
25: E' ← E' ∪ {}
26: **Fin Para**
27: V' ← Permutar aleatoriamente los id's de V'
28: G ← {V, E}
29: G' ← {V', E'}
30: **Retornar** {G,G'}

***Algoritmo 5.1.*** *Generador de grafos sintéticos.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

El algoritmo tiene como fin generar una tupla que contenga a los dos grafos necesarios para desempeñar la búsqueda de (sub) isomorfismo. Para ello recibe cuatro parámetros de entrada que delimitan la cantidad de vértices y aristas que debe tener cada grafo, los parámetros que intervienen son:

$G|V|$: Representa la cantidad de vértices que debe tener el grafo objetivo y se lee como "número de vértices de $G$".

$G|E|$: Representa la cantidad de aristas que debe tener el grafo objetivo y se lee como "número de aristas de $G$".

$G'|V'|$: Representa la cantidad de vértices que debe tener el grafo contenedor y se lee como "número de vértices de $G'$".

$G'|E'|$: Representa la cantidad de aristas que debe tener el grafo contenedor y se lee como "número de aristas de $G'$".

Nótese que siempre debe cumplirse $G'|V'| >= G|V| \wedge G'|E'| >= G|E|$.
Terminada su ejecución el algoritmo retornará un conjunto que incluye a $G$ y a $G'$.

## 5.2.3. Explicación del proceso de generación de grafos pasó a paso

En esta sección se expande la explicación del algoritmo generador de grafos sintéticos presentado en 5.2.2 a través de un esquema de pasos que describe un ejemplo concreto con el fin de poder observar más detenidamente el proceso, y el concepto detrás del mismo brindando así un entendimiento más profundo del comportamiento esperado del algoritmo.
En este ejemplo se aborda la construcción del conjunto de grafos {objetivo, contenedor} teniendo estos cuatro vértices y cinco aristas, y siete vértices y doce aristas respectivamente. Con esto, los parámetros de entrada quedan establecidos como se describe a continuación:

$G|V|$: 4

$G|E|$: 5

$G'|V'|$: 7

$G'|E'|$: 12

A continuación se da lugar a la explicación paso a paso:



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Paso 1.** Se crean vértices aislados, tantos como indique *G|V|.*

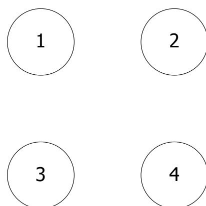



***Figura 5.1.*** *Paso 1 Vértices aislados.*

**Paso 2.** Se crean *G|V| -1* aristas y se las coloca de forma tal que exista un camino entre dos vértices cualesquiera tomados del *paso 1*, dando lugar a un grafo conexo y sin formar ningún ciclo.

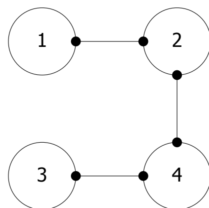



***Figura 5.2.*** *Paso 2 Vértices conectados*

**Paso 3.** Se crean aristas aleatoriamente hasta completar la cantidad especificada en *G|E|.* No importa que se formen ciclos. Nótese que en este punto el grafo objetivo está completo y satisface los requerimientos establecidos en los parámetros de entrada.

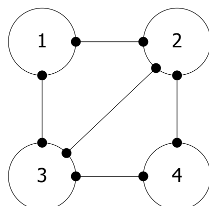



***Figura 5.3.*** *Paso 3 Grafo objetivo completado.*

**Paso 4.** Se crean vértices aislados hasta completar *G'|V'|* teniendo en cuenta los creados previamente.

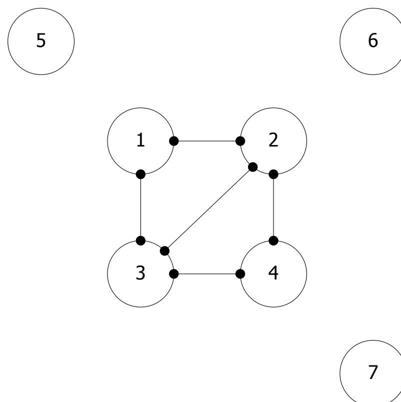



***Figura 5.4.*** *Paso 4 Grafo con vértices aislado.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Paso 5.** Se debe formar un camino entre todos los vértices desconectados creados en el *paso 4* y luego se debe seleccionar uno de ellos y unirlo a través de una arista a alguno de los vértices del grafo objetivo, de forma de lograr una estructura conexa.

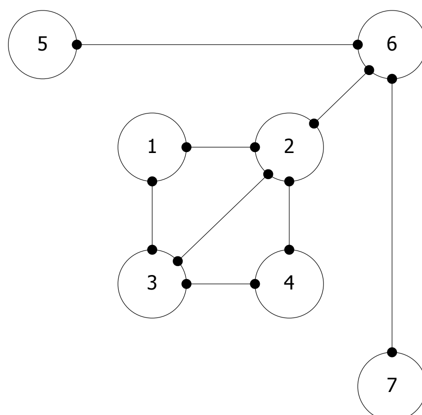



***Figura 5.5.*** *Paso 5 Grafo conexo.*

**Paso 6.** Se crean aristas entre los vértices propios del grafo contenedor (en este ejemplo 5, 6 y 7) y los vértices del grafo objetivo hasta completar la cantidad especificada en $G'/E'/$. No se crean aristas entre dos vértices pertenecientes al grafo objetivo.

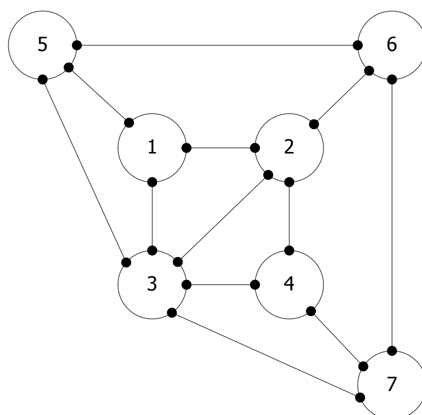



***Figura 5.6.*** *Paso 6 Grafo conexo completado.*

**Paso 7.** Se permuta aleatoriamente la numeración de los vértices para romper la relación que existe entre los id's del grafo objetivo y del grafo contenedor recientemente creado. Esto es importante porque a la hora de ejecutar la búsqueda de (sub)isomorfismo, la distribución matricial lograda será mucho más homogénea lo que agrega un grado más de dificultad a la búsqueda y aumenta el parecido con un caso real.



Repositorio Digital Institucional "José María Rosa"
UNLa.

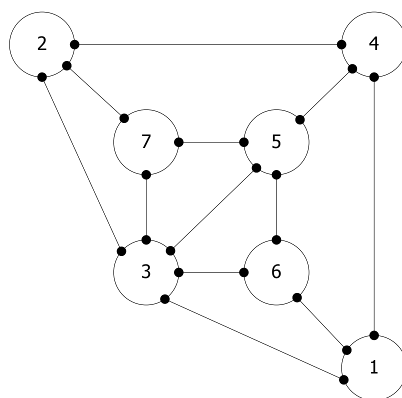



***Figura 5.7.*** *Paso 7 Grafo terminado con los id's permutados.*

Finalmente el proceso debe devolver como resultado el grafo obtenido en el *paso 3* que corresponde al grafo objetivo, y el grafo obtenido en el *paso 7* que corresponde al grafo contenedor.
En la *figura 5.8* se puede apreciar el resultado con el subisomorfismo debidamente señalado:

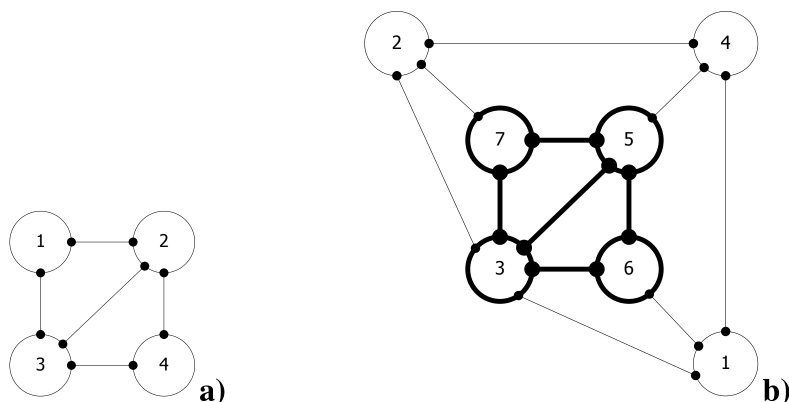



***Figura 5.8. a)*** *Grafo objetivo.* ***b)*** *Grafo contenedor con subisomorfismo señalado.*

# 5.3. CONFIGURACIÓN DE LA EXPERIMENTACIÓN, METODO DE MONTECARLO

En esta sección se expone la lógica desplegada para construir la tabla de parámetros de entrada que luego son utilizados por el generador de grafos para crear las muestras sujetas a experimentación. En la sección 5.3.1 se explica cómo está configurada la tabla de Montecarlo. En la sección 5.3.2 se explican las fórmulas matemáticas utilizadas para obtener los valores que completan la tabla. Finalmente en la sección 5.3.3 se presenta la tabla definitiva, combinando los conceptos expuestos en los apartados previos.



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 5.3.1. Estructura de la tabla de parámetros de entrada siguiendo el método de Montecarlo

Para llevar a delante la experimentación de los algoritmos y lograr un conjunto de resultados que permitiera evaluar el desempeño de los mismos de la forma más precisa posible, se decidió seguir el método de Montecarlo para configurar los parámetros que definen a los grafos a ser testeados.
Es preciso remarcar que existen cuatro parámetros en la creación de los dos grafos que intervienen en el proceso (el grafo objetivo y el grafo contenedor), estos son:

- $G|V|$: Cantidad de vértices del grafo objetivo.
- $D(G)$: Valor de densidad del grafo objetivo.
- $G'|V'|$: Cantidad de vértices del grafo contenedor.
- $D'(G')$: Valor de densidad del grafo contenedor.

Nótese que se eligió utilizar la densidad de los grafos como valor representativo de la cantidad de aristas con las que estos serán construidos, esto se hizo como una medida para aumentar la legibilidad y comprensión de los resultados a la hora de ser cortejados.
Es importante también tener en cuenta que estos parámetros están estrechamente ligados los unos a los otros. Habiendo definido un solo parámetros de forma arbitraria, automáticamente se condiciona los rangos de valores que pueden adoptar los parámetros restantes. Para sortear estas limitaciones físicas inherentes a los grafos, se confeccionó una serie de fórmulas que determinan el rango de valores que puede tomar cada uno de estos parámetros en función de aquellos parámetros que hayan sido establecidos previamente (ver 5.3.2.).

Otro aspecto importante a remarcar es que simplemente utilizando esos cuatro parámetros se pueden obtener miles de combinaciones que solo difieren muy levemente entre sí, y no aportan mayor información a la hora de ponderar el desempeño de los algoritmos, pero que bien pueden hacer a la experimentación totalmente inconducente ya que suman incontables horas de ejecución a todo el proceso, generan una gran cantidad de información redundante y dificultan la observación.
Teniendo en mente el objetivo de lograr una tabla de parámetros donde cada experimentación aporte la mayor cantidad de información y difiera significativamente en términos cualitativos de sus predecesores, se optó por tomar para cada parámetro tres valores posibles: el minino, el máximo y el promedio.



Repositorio Digital Institucional "José María Rosa"
UNLa.

El orden en que se definen los parámetros es el siguiente:

1. Se establece $G/V/$ como variable independiente y se le asigna un valor de forma arbitraria, $G/V/$ puede ser: 5, 15, 25, 40.
2. Se estable el rango de valores de $D(G)$ en función de $G/V/$ y se le asigna un valor minino, promedio, o máximo.
3. Se estable el rango de valores de $G'/V'|$ en función de $G/V/$ y se le asigna un valor minino, promedio, o máximo.
4. Se establece el rango de valores de $D'(G')$ en función de $G/V/, D(G)$ y $G'/V'/$ y se le asigna un valor minino, promedio o máximo.

Finalmente, para construir la tabla, se debe asegurar que están presentes todas las combinaciones posibles, para ello se procede de la misma manera que se utiliza cuando se construyen las tablas de verdad.

| G | | G' | |
|---|---|---|---|
| V | D | V’ | D' |
| 5 | Min | min | min |
| 5 | Min | min | prom |
| 5 | Min | min | max |
| 5 | Min | prom | min |
| 5 | Min | prom | prom |
| 5 | Min | prom | max |
| 5 | Min | max | min |
| 5 | Min | max | prom |
| 5 | Min | max | max |
| … | … | … | … |
| 40 | Max | min | min |
| 40 | Max | min | prom |
| 40 | Max | min | max |
| 40 | Max | prom | min |
| 40 | Max | prom | prom |
| 40 | Max | prom | max |
| 40 | Max | max | min |
| 40 | Max | max | prom |
| 40 | Max | max | max |

***Tabla 5.1.*** *Resumen de la Tabla de Montecarlo. Configuración de la tabla de parámetros de entrada*



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 5.3.2. Fórmulas que limitan el rango de valores de los parámetros de entrada

En esta sección se presentan las fórmulas y restricciones utilizadas para completar la tabla de Montecarlo con los parámetros de entrada. Esto surge a raíz de que la metodología utilizada para crear los grafos (ver 5.2.) si bien admite una amplia gama de combinaciones, también impone ciertas limitaciones que deben ser tenidas en cuenta para no generar grafos defectuosos, en términos de forma, que puedan afectar negativamente la ejecución de los algoritmos.

En primera instancia se tiene *E(D,V)* que es la fórmula para calcular el número de aristas de un grafo a partir de sus vértices y densidad, en la sección 2.2.1 se había definido que la densidad de un grafo viene dada por:

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

(5.1)

A partir de esta fórmula y aplicando un simple despeje de variables se llega a que el número de aristas E de un grafo en función de su número de vértices y su densidad es:

$$E(D,V) = \frac{DV^2 - DV}{2}$$

(5.2)

Se describen las fórmulas (5.3), (5.4), (5.5), (5.6), (5.7), (5.8), (5.9), (5.10), (5.11) y (5.12) que delimitan los rangos de valores de los parámetros de entrada:

*G|V|:* Representa número de vértices que se incluyen en el grafo objetivo. Cumple la función de variable independiente ya que se utiliza como punto de partida para construir los parámetros restantes. *G|V|* se define por extensión, formalmente:

$$G|V| = \{5, 15, 25, 40\}$$

(5.3)

*MinDenG(G|V|):* La mínima densidad posible de *G* en función de *G|V|.* Se debe tener en cuenta que para que el grafo sea conexo la mínima cantidad de aristas que se pueden utilizar es igual *G|V| -1.* Luego haciendo los reemplazos en (5.1) y reduciendo la expresión se llega a:

$$MinDenG(G|V|) = \frac{2}{V}$$

(5.4)



Repositorio Digital Institucional "José María Rosa"
UNLa.

*MaxDenG(G|V|):* La máxima densidad posible de $G$ en función de $G|V|$. En este caso no hay restricciones sobre la cantidad de aristas que puede tener $G$ por lo que su densidad máxima viene dada por:

$$MaxDenG(G|V|) = 1 \tag{5.5}$$

*PromDenG(G|V|):* La densidad promedio de $G$ en función de $G|V|$. Se calcula como un promedio de (5.4) y (5.5):

$$PromDenG(G|V|) = \frac{MinDenG(G|V|) + MaxDenG(G|V|)}{2} \tag{5.6}$$

*MinVerG'(G|V|):* La mínima cantidad de vértices de $G'$ en función de $G|V|$. En este caso la minina cantidad de vértices que puede tener el grafo contenedor, sabiendo que por construcción este debe ser más grande que el grafo objetivo, se desprende que:

$$MinVerG'(G|V|) = G|V| + 1 \tag{5.7}$$

*MaxVerG'(G|V|):* La máxima cantidad de vértices de $G'$. En este caso, físicamente no existe una restricción que impida llevar la cantidad de vértices de $G'$ hasta infinito, pero por términos prácticos se limitó este valor a cincuenta, luego:

$$MaxVerG' = 50 \tag{5.8}$$

*PromVerG'(G|V|):* La cantidad promedio de vértices de $G'$ en función de $G|V|$. Se calcula como un promedio de (5.7) y (5.8):

$$PromVerG'(G|V|) = \frac{MinVerG'(G|V|) + 50}{2} \tag{5.9}$$

*MinDenG'(D(G), G|V|,G'|V'|):* La mínima densidad de $G'$ en función de $D(G)$, $G|V|$ y $G'|V'|$. Para calcular la mínima densidad de $G'$ se debe partir de la base *que G'=G+(aristas extras).* En primera instancia se calcula la cantidad de aristas que tendrá $G$ en función de su densidad y sus vértices aplicando (5.2). Luego, se debe adicionar la mínima cantidad de aristas necesarias para conectar los vértices extra agregados que pertenecen a $G'$, esa cantidad equivale exactamente a



Repositorio Digital Institucional "José María Rosa"
UNLa.

*G'|V'| - G|V|.* Una vez que se tiene el número de aristas, para calcular la densidad se aplica (5.1), formalmente:

$$MinDenG'(D(G), G|V|, G|V'|) = \frac{2[E(D(G), G|V|) + (G'|V'| - G|V|)]}{G|V'|(G|V'| - 1)}$$

(5.10)

*MaxDenG'(D(G), G|V|,G'|V'|):* La máxima densidad de *G'* en función de *D(G), G|V|* y de *G'|V'|.* Para calcular la máxima densidad posible de *G'* primero se debe determinar la cantidad máxima de aristas en *G'* que nuestro proceso de construcción admite. Nótese que se busca no agregar aristas extra entre los vértices de *G*, con esa restricción en mente se desprende que la cantidad máxima de aristas de *G'* es igual a: a la a *todas los posibles aristas de G'* – (*las posibles aristas de G – las aristas de G*). La *figura 5.9* describe lo anterior:

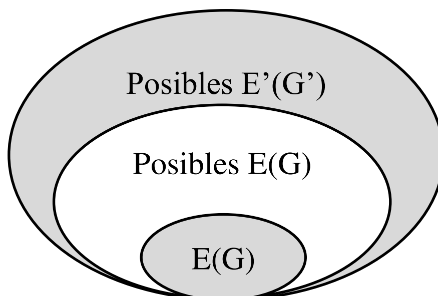



***Figura 5.9.*** *Diagrama de conjuntos. La parte sombreada representa la máxima cantidad de aristas de G'*

Finalmente, una vez que se tiene la cantidad máxima de aristas de *G'* se debe aplicar (5.1) para obtener la densidad. Formalmente:

$$MaxDenG'(D(G), G|V|, G|V'|) = \frac{2\left[C_{G'|V'|,2} - \left(C_{G|V|,2} - E(D(G), G|V|)\right)\right]}{G'|V'|(G'|V'| - 1)}$$

(5.11)

*PromDenG'(D(G), G|V|,G'|V'|):* La densidad promedio de *G'* función de *D(G), G|V|* y *de G'|V'|.* Se calcula como un promedio entre (5.10) y (5.11):

$$PromDenG'(D(G), G|V|, G|V'|) = \frac{MinDenG'(D(G), G|V|, G|V'|) + MaxDenG'(D(G), G|V|, G|V'|)}{2}$$

(5.12)



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 5.3.3. Tabla de Montecarlo completa con los valores de los parámetros de entrada

En esta sección se presenta la tabla final de Montecarlo con los parámetros de entrada ya calculados producto de la aplicación de los conceptos descriptos en las secciones 5.1 y 5.2.

Nótese que se agregó las columnas $E$ y $E'$ que representan la cantidad de aristas del grafo $G$ y $G'$ respectivamente, las volares de la columnas $E$ y $E'$ se calcularon aplicando (5.2) en cada caso.

| G | | | G' | | |
|---|---|---|---|---|---|
| V | D | E | V' | D' | E' |
| 5 | 40,0% | 4 | 6 | 33,3% | 5 |
| 5 | 40,0% | 4 | 6 | 46,7% | 7 |
| 5 | 40,0% | 4 | 6 | 60,0% | 9 |
| 5 | 40,0% | 4 | 28 | 7,1% | 27 |
| 5 | 40,0% | 4 | 28 | 52,8% | 199 |
| 5 | 40,0% | 4 | 28 | 98,4% | 372 |
| 5 | 40,0% | 4 | 50 | 4,0% | 49 |
| 5 | 40,0% | 4 | 50 | 51,8% | 634 |
| 5 | 40,0% | 4 | 50 | 99,5% | 1219 |
| 5 | 70,0% | 7 | 6 | 53,3% | 8 |
| 5 | 70,0% | 7 | 6 | 66,7% | 10 |
| 5 | 70,0% | 7 | 6 | 80,0% | 12 |
| 5 | 70,0% | 7 | 28 | 7,9% | 30 |
| 5 | 70,0% | 7 | 28 | 53,6% | 202 |
| 5 | 70,0% | 7 | 28 | 99,2% | 375 |
| 5 | 70,0% | 7 | 50 | 4,2% | 52 |
| 5 | 70,0% | 7 | 50 | 52,0% | 637 |
| 5 | 70,0% | 7 | 50 | 99,8% | 1222 |
| 5 | 100,0% | 10 | 6 | 73,3% | 11 |
| 5 | 100,0% | 10 | 6 | 86,7% | 13 |
| 5 | 100,0% | 10 | 6 | 100,0% | 15 |
| 5 | 100,0% | 10 | 28 | 8,7% | 33 |
| 5 | 100,0% | 10 | 28 | 54,4% | 205 |
| 5 | 100,0% | 10 | 28 | 100,0% | 378 |
| 5 | 100,0% | 10 | 50 | 4,5% | 55 |
| 5 | 100,0% | 10 | 50 | 52,2% | 640 |
| 5 | 100,0% | 10 | 50 | 100,0% | 1225 |
| 15 | 13,3% | 14 | 16 | 12,5% | 15 |
| 15 | 13,3% | 14 | 16 | 18,3% | 22 |
| 15 | 13,3% | 14 | 16 | 24,2% | 29 |
| 15 | 13,3% | 14 | 33 | 6,1% | 32 |
| 15 | 13,3% | 14 | 33 | 44,4% | 234 |
| 15 | 13,3% | 14 | 33 | 82,8% | 437 |

***Tabla 5.2.a.*** *Tabla de Montecarlo*. *Tabla con los valores de parámetros de entrada calculados.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | |
|---|---|---|---|---|---|
| V | D | E | V' | D' | E' |
| 15 | 13,3% | 14 | 50 | 4,0% | 49 |
| 15 | 13,3% | 14 | 50 | 48,3% | 591 |
| 15 | 13,3% | 14 | 50 | 92,6% | 1134 |
| 15 | 56,7% | 59 | 16 | 50,0% | 60 |
| 15 | 56,7% | 59 | 16 | 55,8% | 67 |
| 15 | 56,7% | 59 | 16 | 61,7% | 74 |
| 15 | 56,7% | 59 | 33 | 14,6% | 77 |
| 15 | 56,7% | 59 | 33 | 52,9% | 279 |
| 15 | 56,7% | 59 | 33 | 91,3% | 482 |
| 15 | 56,7% | 59 | 50 | 7,7% | 94 |
| 15 | 56,7% | 59 | 50 | 52,0% | 636 |
| 15 | 56,7% | 59 | 50 | 96,2% | 1179 |
| 15 | 100,0% | 105 | 16 | 88,3% | 106 |
| 15 | 100,0% | 105 | 16 | 94,2% | 113 |
| 15 | 100,0% | 105 | 16 | 100,0% | 120 |
| 15 | 100,0% | 105 | 33 | 23,3% | 123 |
| 15 | 100,0% | 105 | 33 | 61,6% | 325 |
| 15 | 100,0% | 105 | 33 | 100,0% | 528 |
| 15 | 100,0% | 105 | 50 | 11,4% | 140 |
| 15 | 100,0% | 105 | 50 | 55,7% | 682 |
| 15 | 100,0% | 105 | 50 | 100,0% | 1225 |
| 25 | 8,0% | 24 | 26 | 7,7% | 25 |
| 25 | 8,0% | 24 | 26 | 11,4% | 37 |
| 25 | 8,0% | 24 | 26 | 15,1% | 49 |
| 25 | 8,0% | 24 | 38 | 5,3% | 37 |
| 25 | 8,0% | 24 | 38 | 33,0% | 232 |
| 25 | 8,0% | 24 | 38 | 60,7% | 427 |
| 25 | 8,0% | 24 | 50 | 4,0% | 49 |
| 25 | 8,0% | 24 | 50 | 40,7% | 499 |
| 25 | 8,0% | 24 | 50 | 77,5% | 949 |
| 25 | 54,0% | 162 | 26 | 50,2% | 163 |
| 25 | 54,0% | 162 | 26 | 53,8% | 175 |
| 25 | 54,0% | 162 | 26 | 57,5% | 187 |
| 25 | 54,0% | 162 | 38 | 24,9% | 175 |
| 25 | 54,0% | 162 | 38 | 52,6% | 370 |
| 25 | 54,0% | 162 | 38 | 80,4% | 565 |
| 25 | 54,0% | 162 | 50 | 15,3% | 187 |
| 25 | 54,0% | 162 | 50 | 52,0% | 637 |
| 25 | 54,0% | 162 | 50 | 88,7% | 1087 |
| 25 | 100,0% | 300 | 26 | 92,6% | 301 |
| 25 | 100,0% | 300 | 26 | 96,3% | 313 |
| 25 | 100,0% | 300 | 26 | 100,0% | 325 |
| 25 | 100,0% | 300 | 38 | 44,5% | 313 |

***Tabla 5.2.b.*** *Tabla de Montecarlo. Tabla con los valores de parámetros de entrada calculados.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | |
|---|---|---|---|---|---|
| V | D | E | V' | D' | E' |
| 25 | 100,0% | 300 | 38 | 72,3% | 508 |
| 25 | 100,0% | 300 | 38 | 100,0% | 703 |
| 25 | 100,0% | 300 | 50 | 26,5% | 325 |
| 25 | 100,0% | 300 | 50 | 63,3% | 775 |
| 25 | 100,0% | 300 | 50 | 100,0% | 1225 |
| 40 | 5,0% | 39 | 41 | 4,9% | 40 |
| 40 | 5,0% | 39 | 41 | 7,3% | 59 |
| 40 | 5,0% | 39 | 41 | 9,6% | 79 |
| 40 | 5,0% | 39 | 45 | 4,4% | 44 |
| 40 | 5,0% | 39 | 45 | 14,8% | 146 |
| 40 | 5,0% | 39 | 45 | 25,2% | 249 |
| 40 | 5,0% | 39 | 50 | 4,0% | 49 |
| 40 | 5,0% | 39 | 50 | 21,8% | 266 |
| 40 | 5,0% | 39 | 50 | 39,5% | 484 |
| 40 | 52,5% | 409 | 41 | 50,0% | 410 |
| 40 | 52,5% | 409 | 41 | 52,4% | 429 |
| 40 | 52,5% | 409 | 41 | 54,8% | 449 |
| 40 | 52,5% | 409 | 45 | 41,8% | 414 |
| 40 | 52,5% | 409 | 45 | 52,2% | 516 |
| 40 | 52,5% | 409 | 45 | 62,5% | 619 |
| 40 | 52,5% | 409 | 50 | 34,2% | 419 |
| 40 | 52,5% | 409 | 50 | 52,0% | 636 |
| 40 | 52,5% | 409 | 50 | 69,7% | 854 |
| 40 | 100,0% | 780 | 41 | 95,2% | 781 |
| 40 | 100,0% | 780 | 41 | 97,6% | 800 |
| 40 | 100,0% | 780 | 41 | 100,0% | 820 |
| 40 | 100,0% | 780 | 45 | 79,3% | 785 |
| 40 | 100,0% | 780 | 45 | 89,6% | 887 |
| 40 | 100,0% | 780 | 45 | 100,0% | 990 |
| 40 | 100,0% | 780 | 50 | 64,5% | 790 |
| 40 | 100,0% | 780 | 50 | 82,2% | 1007 |
| 40 | 100,0% | 780 | 50 | 100,0% | 1225 |

***Tabla 5.2.c.*** *Tabla de Montecarlo**.** Tabla con los valores de parámetros de entrada calculados.*

# 5.4. RESULTADOS DE LA EJECUCIÓN DEL ALGORITMO DE ULLMAN

En esta sección se presentan los resultados obtenidos a raíz de la ejecución del algoritmo de Ullman para cada caso de experimentación propuesto en la tabla 5.2. En la sección 5.4.1 se exponen los resultados de la tabla. En la sección 5.4.2 se realizan observaciones sobre el desempeño del algoritmo.



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 5.4.1. Tabla de resultados

Se presenta la *tabla 5.3* de resultados del algoritmo de Ullman. Las columnas debajo de $G$ y $G'$ representan los parámetros de los grafos construidos. "Tiempo en ms" representa el tiempo promedio que demoró el algoritmo, obtenido del tiempo total de ejecución divido diez (que fue la precisión utilizada). El resultado "60000" significa que el algoritmo no fue capaz de hallar una solución dentro de los límites de tiempo estipulados.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 5 | 4 | 40,00% | 6 | 5 | 33,30% | 0 |
| 5 | 4 | 40,00% | 6 | 7 | 46,70% | 1 |
| 5 | 4 | 40,00% | 6 | 9 | 60,00% | 0 |
| 5 | 4 | 40,00% | 28 | 27 | 7,10% | 1573 |
| 5 | 4 | 40,00% | 28 | 199 | 52,60% | 199 |
| 5 | 4 | 40,00% | 28 | 372 | 98,40% | 0 |
| 5 | 4 | 40,00% | 50 | 49 | 4,00% | 37012 |
| 5 | 4 | 40,00% | 50 | 634 | 51,80% | 1158 |
| 5 | 4 | 40,00% | 50 | 1219 | 99,50% | 0 |
| 5 | 7 | 70,00% | 6 | 8 | 53,30% | 0 |
| 5 | 7 | 70,00% | 6 | 10 | 66,70% | 0 |
| 5 | 7 | 70,00% | 6 | 12 | 80,00% | 0 |
| 5 | 7 | 70,00% | 28 | 30 | 7,90% | 0 |
| 5 | 7 | 70,00% | 28 | 202 | 53,40% | 264 |
| 5 | 7 | 70,00% | 28 | 375 | 99,20% | 0 |
| 5 | 7 | 70,00% | 50 | 52 | 4,20% | 0 |
| 5 | 7 | 70,00% | 50 | 637 | 52,00% | 1512 |
| 5 | 7 | 70,00% | 50 | 1222 | 99,80% | 0 |
| 5 | 10 | 100,00% | 6 | 11 | 73,30% | 0 |
| 5 | 10 | 100,00% | 6 | 13 | 86,70% | 0 |
| 5 | 10 | 100,00% | 6 | 15 | 100,00% | 0 |
| 5 | 10 | 100,00% | 28 | 33 | 8,70% | 0 |
| 5 | 10 | 100,00% | 28 | 205 | 54,20% | 152 |
| 5 | 10 | 100,00% | 28 | 378 | 100,00% | 0 |
| 5 | 10 | 100,00% | 50 | 55 | 4,50% | 0 |
| 5 | 10 | 100,00% | 50 | 640 | 52,20% | 1450 |
| 5 | 10 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 15 | 14 | 13,30% | 16 | 15 | 12,50% | 60000 |
| 15 | 14 | 13,30% | 16 | 22 | 18,30% | 60000 |
| 15 | 14 | 13,30% | 16 | 29 | 24,20% | 60000 |
| 15 | 14 | 13,30% | 33 | 32 | 6,10% | 60000 |
| 15 | 14 | 13,30% | 33 | 234 | 44,30% | 60000 |

***Tabla 5.3.a.*** *Resultados de la ejecución del algoritmo de Ullman*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 15 | 14 | 13,30% | 33 | 437 | 82,80% | 42042 |
| 15 | 14 | 13,30% | 50 | 49 | 4,00% | 60000 |
| 15 | 14 | 13,30% | 50 | 591 | 48,20% | 60000 |
| 15 | 14 | 13,30% | 50 | 1134 | 92,60% | 42000 |
| 15 | 59 | 56,20% | 16 | 60 | 50,00% | 9469 |
| 15 | 59 | 56,20% | 16 | 67 | 55,80% | 50369 |
| 15 | 59 | 56,20% | 16 | 74 | 61,70% | 60000 |
| 15 | 59 | 56,20% | 33 | 77 | 14,60% | 3198 |
| 15 | 59 | 56,20% | 33 | 279 | 52,80% | 60000 |
| 15 | 59 | 56,20% | 33 | 482 | 91,30% | 54040 |
| 15 | 59 | 56,20% | 50 | 94 | 7,70% | 2798 |
| 15 | 59 | 56,20% | 50 | 636 | 51,90% | 60000 |
| 15 | 59 | 56,20% | 50 | 1179 | 96,20% | 24455 |
| 15 | 105 | 100,00% | 16 | 106 | 88,30% | 0 |
| 15 | 105 | 100,00% | 16 | 113 | 94,20% | 0 |
| 15 | 105 | 100,00% | 16 | 120 | 100,00% | 0 |
| 15 | 105 | 100,00% | 33 | 123 | 23,30% | 0 |
| 15 | 105 | 100,00% | 33 | 325 | 61,60% | 60000 |
| 15 | 105 | 100,00% | 33 | 528 | 100,00% | 0 |
| 15 | 105 | 100,00% | 50 | 140 | 11,40% | 0 |
| 15 | 105 | 100,00% | 50 | 682 | 55,70% | 60000 |
| 15 | 105 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 25 | 24 | 8,00% | 26 | 25 | 7,70% | 60000 |
| 25 | 24 | 8,00% | 26 | 37 | 11,40% | 60000 |
| 25 | 24 | 8,00% | 26 | 49 | 15,10% | 60000 |
| 25 | 24 | 8,00% | 38 | 37 | 5,30% | 60000 |
| 25 | 24 | 8,00% | 38 | 232 | 33,00% | 60000 |
| 25 | 24 | 8,00% | 38 | 427 | 60,70% | 60000 |
| 25 | 24 | 8,00% | 50 | 49 | 4,00% | 60000 |
| 25 | 24 | 8,00% | 50 | 499 | 40,70% | 60000 |
| 25 | 24 | 8,00% | 50 | 949 | 77,50% | 54000 |
| 25 | 162 | 54,00% | 26 | 163 | 50,10% | 60000 |
| 25 | 162 | 54,00% | 26 | 175 | 53,80% | 60000 |
| 25 | 162 | 54,00% | 26 | 187 | 57,50% | 60000 |
| 25 | 162 | 54,00% | 38 | 175 | 24,90% | 60000 |
| 25 | 162 | 54,00% | 38 | 370 | 52,60% | 60000 |
| 25 | 162 | 54,00% | 38 | 565 | 80,40% | 60000 |
| 25 | 162 | 54,00% | 50 | 187 | 15,30% | 60000 |
| 25 | 162 | 54,00% | 50 | 637 | 52,00% | 60000 |
| 25 | 162 | 54,00% | 50 | 1087 | 88,70% | 60000 |
| 25 | 300 | 100,00% | 26 | 301 | 92,60% | 0 |
| 25 | 300 | 100,00% | 26 | 313 | 96,30% | 0 |
| 25 | 300 | 100,00% | 26 | 325 | 100,00% | 0 |

***Tabla 5.3.b.*** *Resultados de la ejecución del algoritmo de Ullman*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 25 | 300 | 100,00% | 38 | 313 | 44,50% | 0 |
| 25 | 300 | 100,00% | 38 | 508 | 72,30% | 42000 |
| 25 | 300 | 100,00% | 38 | 703 | 100,00% | 0 |
| 25 | 300 | 100,00% | 50 | 325 | 26,50% | 0 |
| 25 | 300 | 100,00% | 50 | 775 | 63,30% | 60000 |
| 25 | 300 | 100,00% | 50 | 1225 | 100,00% | 1 |
| 40 | 39 | 5,00% | 41 | 40 | 4,90% | 60000 |
| 40 | 39 | 5,00% | 41 | 59 | 7,20% | 60000 |
| 40 | 39 | 5,00% | 41 | 79 | 9,60% | 60000 |
| 40 | 39 | 5,00% | 45 | 44 | 4,40% | 60000 |
| 40 | 39 | 5,00% | 45 | 146 | 14,70% | 60000 |
| 40 | 39 | 5,00% | 45 | 249 | 25,20% | 60000 |
| 40 | 39 | 5,00% | 50 | 49 | 4,00% | 60000 |
| 40 | 39 | 5,00% | 50 | 266 | 21,70% | 60000 |
| 40 | 39 | 5,00% | 50 | 484 | 39,50% | 60000 |
| 40 | 409 | 52,40% | 41 | 410 | 50,00% | 60000 |
| 40 | 409 | 52,40% | 41 | 429 | 52,30% | 60000 |
| 40 | 409 | 52,40% | 41 | 449 | 54,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 414 | 41,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 516 | 52,10% | 60000 |
| 40 | 409 | 52,40% | 45 | 619 | 62,50% | 60000 |
| 40 | 409 | 52,40% | 50 | 419 | 34,20% | 60000 |
| 40 | 409 | 52,40% | 50 | 636 | 51,90% | 60000 |
| 40 | 409 | 52,40% | 50 | 854 | 69,70% | 60000 |
| 40 | 780 | 100,00% | 41 | 781 | 95,20% | 1 |
| 40 | 780 | 100,00% | 41 | 800 | 97,60% | 1 |
| 40 | 780 | 100,00% | 41 | 820 | 100,00% | 1 |
| 40 | 780 | 100,00% | 45 | 785 | 79,30% | 1 |
| 40 | 780 | 100,00% | 45 | 887 | 89,60% | 1 |
| 40 | 780 | 100,00% | 45 | 990 | 100,00% | 1 |
| 40 | 780 | 100,00% | 50 | 790 | 64,50% | 1 |
| 40 | 780 | 100,00% | 50 | 1007 | 82,20% | 1 |
| 40 | 780 | 100,00% | 50 | 1225 | 100,00% | 1 |

***Tabla 5.3.c.*** *Resultados de la ejecución del algoritmo de Ullman*

## 5.4.2. Valoración del algoritmo, estadísticas, observaciones

Tras la ejecución de la experimentación se obtiene que el algoritmo de Ullman fue capaz de resolver satisfactoriamente el 55,55% de los casos propuestos en tan solo 6 minutos y 7 segundos, mientras que no pudo resolver el 44,55% restante habiendo demorado en este conjunto 48 minutos.



Repositorio Digital Institucional "José María Rosa"
UNLa.

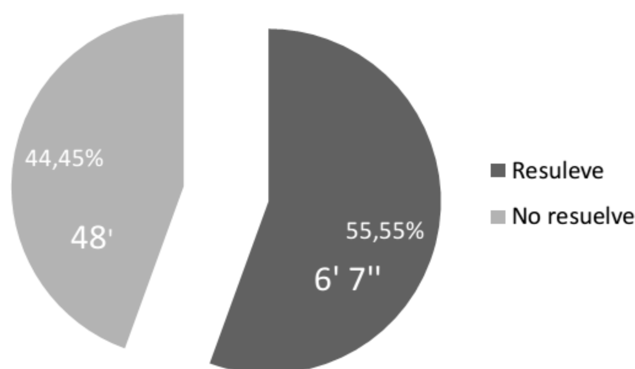



***Figura 5.10.*** *Grafico circular - Ullman. Muestra los porcentajes de casos resueltos y no resueltos.*

Dado el contraste tan amplio de tiempo entre los casos resueltos y los no resueltos, se conjetura la existencia de un umbral en los parámetros de entrada, donde una vez atravesado, el algoritmo incrementa exponencialmente sus tiempos de ejecución.

La *figura 5.11* analiza los casos que el algoritmo no pudo resolver en función de las densidades de los grafos de entrada:

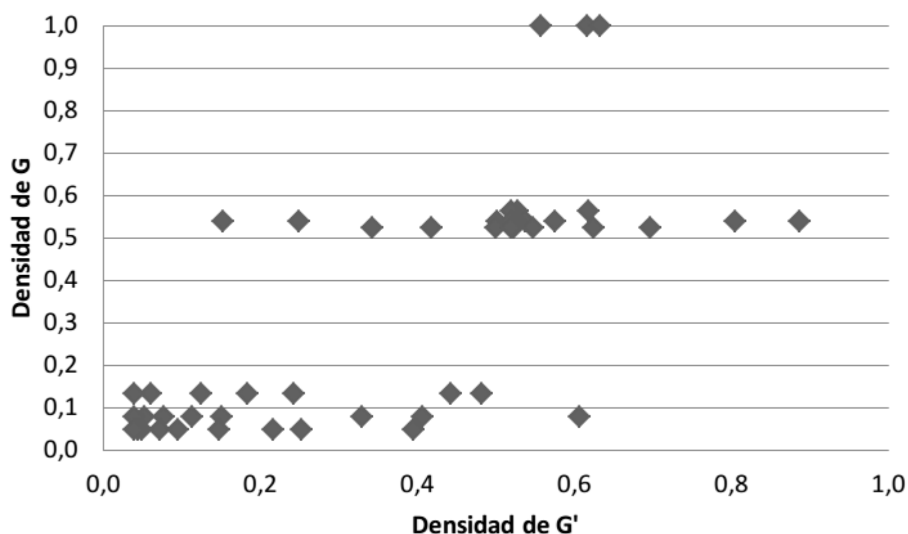



***Figura 5.11.*** *Gráfico de dispersión – Ullman. Muestra los casos no resueltos en función de las densidades de G y G'.*

En la *figura 5.11* se puede observar los rangos de parámetros que presentan mayor dificultad al algoritmo. Se distinguen claramente dos acumulaciones que tienen lugar cuando la densidad de $G$ y $G'$ es baja, es decir, cuando busca (sub)isomorfismo sobre dos grafos dispersos. Y cuando la densidad de $G$ y $G'$ se encuentra en el rango de [50%, 60 %] siendo este último caso el que constituye una mayor dificultad.

La *figura 5.12* analiza los casos que el algoritmo sí pudo resolver en función de las densidades de los grafos de entrada:



Repositorio Digital Institucional "José María Rosa"
UNLa.

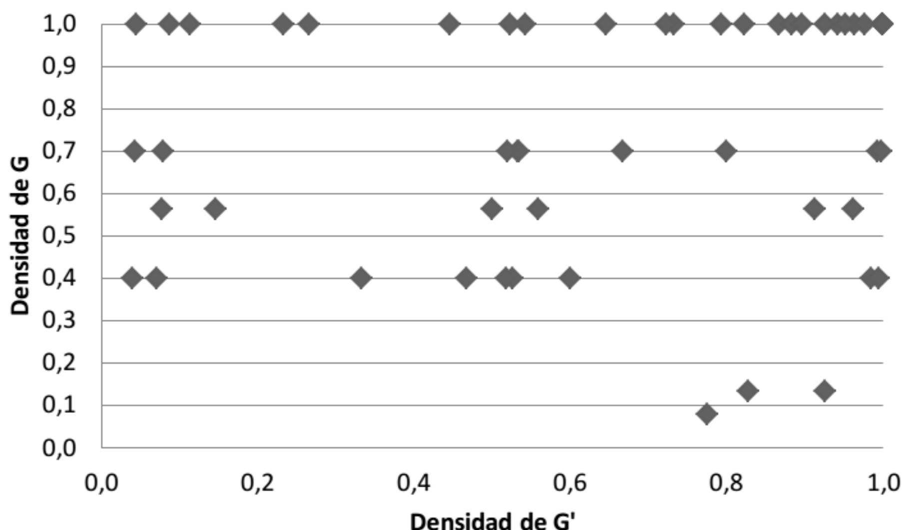



***Figura 5.12.*** *Gráfico de dispersión - Ullman. Muestra los casos resueltos en función de las densidades de G y G'.*

En la *figura 5.12* se observa que los casos que mejor resuelve el algoritmo son aquellos donde tanto la densidad de $G$ como de $G'$ es muy elevada.

# 5.5. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO DE ULLMAN APLICANDO PARALELISMO.

En esta sección se presentan los resultados obtenidos a raíz de la ejecución del algoritmo de Ullman aplicando el método de paralelización propuesto en 4.3 para cada caso de experimentación de la tabla 5.2. En la sección 5.5.1 se exponen los resultados de la tabla. En la sección 5.5.2 se realizan observaciones sobre el desempeño del algoritmo.

## 5.5.1. Tabla de resultados

Se presenta la *tabla 5.4* de resultados del algoritmo de Ullman paralelizado. Las columnas debajo de $G$ y $G'$ representan los parámetros de los grafos construidos. "Tiempo en ms" representa el tiempo promedio que demoró el algoritmo, obtenido del tiempo total de ejecución divido diez (que fue la precisión utilizada). El resultado "60000" significa que el algoritmo no fue capaz de hallar una solución dentro de los límites de tiempo estipulados.



Repositorio Digital Institucional "José María Rosa" UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 5 | 4 | 40,00% | 6 | 5 | 33,30% | 1 |
| 5 | 4 | 40,00% | 6 | 7 | 46,70% | 1 |
| 5 | 4 | 40,00% | 6 | 9 | 60,00% | 0 |
| 5 | 4 | 40,00% | 28 | 27 | 7,10% | 558 |
| 5 | 4 | 40,00% | 28 | 199 | 52,60% | 34 |
| 5 | 4 | 40,00% | 28 | 372 | 98,40% | 0 |
| 5 | 4 | 40,00% | 50 | 49 | 4,00% | 11630 |
| 5 | 4 | 40,00% | 50 | 634 | 51,80% | 342 |
| 5 | 4 | 40,00% | 50 | 1219 | 99,50% | 0 |
| 5 | 7 | 70,00% | 6 | 8 | 53,30% | 0 |
| 5 | 7 | 70,00% | 6 | 10 | 66,70% | 0 |
| 5 | 7 | 70,00% | 6 | 12 | 80,00% | 0 |
| 5 | 7 | 70,00% | 28 | 30 | 7,90% | 1 |
| 5 | 7 | 70,00% | 28 | 202 | 53,40% | 12 |
| 5 | 7 | 70,00% | 28 | 375 | 99,20% | 0 |
| 5 | 7 | 70,00% | 50 | 52 | 4,20% | 2 |
| 5 | 7 | 70,00% | 50 | 637 | 52,00% | 79 |
| 5 | 7 | 70,00% | 50 | 1222 | 99,80% | 0 |
| 5 | 10 | 100,00% | 6 | 11 | 73,30% | 0 |
| 5 | 10 | 100,00% | 6 | 13 | 86,70% | 0 |
| 5 | 10 | 100,00% | 6 | 15 | 100,00% | 0 |
| 5 | 10 | 100,00% | 28 | 33 | 8,70% | 0 |
| 5 | 10 | 100,00% | 28 | 205 | 54,20% | 110 |
| 5 | 10 | 100,00% | 28 | 378 | 100,00% | 0 |
| 5 | 10 | 100,00% | 50 | 55 | 4,50% | 0 |
| 5 | 10 | 100,00% | 50 | 640 | 52,20% | 485 |
| 5 | 10 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 15 | 14 | 13,30% | 16 | 15 | 12,50% | 60000 |
| 15 | 14 | 13,30% | 16 | 22 | 18,30% | 60000 |
| 15 | 14 | 13,30% | 16 | 29 | 24,20% | 60000 |
| 15 | 14 | 13,30% | 33 | 32 | 6,10% | 60000 |
| 15 | 14 | 13,30% | 33 | 234 | 44,30% | 60000 |
| 15 | 14 | 13,30% | 33 | 437 | 82,80% | 42000 |
| 15 | 14 | 13,30% | 50 | 49 | 4,00% | 60000 |
| 15 | 14 | 13,30% | 50 | 591 | 48,20% | 60000 |
| 15 | 14 | 13,30% | 50 | 1134 | 92,60% | 36416 |
| 15 | 59 | 56,20% | 16 | 60 | 50,00% | 15100 |
| 15 | 59 | 56,20% | 16 | 67 | 55,80% | 48623 |
| 15 | 59 | 56,20% | 16 | 74 | 61,70% | 54329 |
| 15 | 59 | 56,20% | 33 | 77 | 14,60% | 4683 |
| 15 | 59 | 56,20% | 33 | 279 | 52,80% | 60000 |
| 15 | 59 | 56,20% | 33 | 482 | 91,30% | 48917 |
| 15 | 59 | 56,20% | 50 | 94 | 7,70% | 3341 |

***Tabla 5.4.a.*** *Resultados de la ejecución del algoritmo de Ullman paralelizado*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 15 | 59 | 56,20% | 50 | 636 | 51,90% | 60000 |
| 15 | 59 | 56,20% | 50 | 1179 | 96,20% | 24002 |
| 15 | 105 | 100,00% | 16 | 106 | 88,30% | 0 |
| 15 | 105 | 100,00% | 16 | 113 | 94,20% | 0 |
| 15 | 105 | 100,00% | 16 | 120 | 100,00% | 0 |
| 15 | 105 | 100,00% | 33 | 123 | 23,30% | 0 |
| 15 | 105 | 100,00% | 33 | 325 | 61,60% | 60000 |
| 15 | 105 | 100,00% | 33 | 528 | 100,00% | 0 |
| 15 | 105 | 100,00% | 50 | 140 | 11,40% | 0 |
| 15 | 105 | 100,00% | 50 | 682 | 55,70% | 60000 |
| 15 | 105 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 25 | 24 | 8,00% | 26 | 25 | 7,70% | 60000 |
| 25 | 24 | 8,00% | 26 | 37 | 11,40% | 60000 |
| 25 | 24 | 8,00% | 26 | 49 | 15,10% | 60000 |
| 25 | 24 | 8,00% | 38 | 37 | 5,30% | 60000 |
| 25 | 24 | 8,00% | 38 | 232 | 33,00% | 60000 |
| 25 | 24 | 8,00% | 38 | 427 | 60,70% | 60000 |
| 25 | 24 | 8,00% | 50 | 49 | 4,00% | 60000 |
| 25 | 24 | 8,00% | 50 | 499 | 40,70% | 60000 |
| 25 | 24 | 8,00% | 50 | 949 | 77,50% | 54000 |
| 25 | 162 | 54,00% | 26 | 163 | 50,20% | 60000 |
| 25 | 162 | 54,00% | 26 | 175 | 53,80% | 60000 |
| 25 | 162 | 54,00% | 26 | 187 | 57,50% | 60000 |
| 25 | 162 | 54,00% | 38 | 175 | 24,90% | 60000 |
| 25 | 162 | 54,00% | 38 | 370 | 52,60% | 60000 |
| 25 | 162 | 54,00% | 38 | 565 | 80,40% | 60000 |
| 25 | 162 | 54,00% | 50 | 187 | 15,30% | 60000 |
| 25 | 162 | 54,00% | 50 | 637 | 52,00% | 60000 |
| 25 | 162 | 54,00% | 50 | 1087 | 88,70% | 60000 |
| 25 | 300 | 100,00% | 26 | 301 | 92,60% | 1 |
| 25 | 300 | 100,00% | 26 | 313 | 96,30% | 0 |
| 25 | 300 | 100,00% | 26 | 325 | 100,00% | 0 |
| 25 | 300 | 100,00% | 38 | 313 | 44,50% | 1 |
| 25 | 300 | 100,00% | 38 | 508 | 72,30% | 33193 |
| 25 | 300 | 100,00% | 38 | 703 | 100,00% | 1 |
| 25 | 300 | 100,00% | 50 | 325 | 26,50% | 0 |
| 25 | 300 | 100,00% | 50 | 775 | 63,30% | 60000 |
| 25 | 300 | 100,00% | 50 | 1225 | 100,00% | 1 |
| 40 | 39 | 5,00% | 41 | 40 | 4,90% | 60000 |
| 40 | 39 | 5,00% | 41 | 59 | 7,20% | 60000 |
| 40 | 39 | 5,00% | 41 | 79 | 9,60% | 60000 |
| 40 | 39 | 5,00% | 45 | 44 | 4,40% | 60000 |
| 40 | 39 | 5,00% | 45 | 146 | 14,70% | 60000 |

***Tabla 5.4.b.** Resultados de la ejecución del algoritmo de Ullman paralelizado*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 40 | 39 | 5,00% | 45 | 249 | 25,20% | 60000 |
| 40 | 39 | 5,00% | 50 | 49 | 4,00% | 60000 |
| 40 | 39 | 5,00% | 50 | 266 | 21,70% | 60000 |
| 40 | 39 | 5,00% | 50 | 484 | 39,50% | 60000 |
| 40 | 409 | 52,40% | 41 | 410 | 50,00% | 60000 |
| 40 | 409 | 52,40% | 41 | 429 | 52,30% | 60000 |
| 40 | 409 | 52,40% | 41 | 449 | 54,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 414 | 41,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 516 | 52,10% | 60000 |
| 40 | 409 | 52,40% | 45 | 619 | 62,50% | 60000 |
| 40 | 409 | 52,40% | 50 | 419 | 34,20% | 60000 |
| 40 | 409 | 52,40% | 50 | 636 | 51,90% | 60000 |
| 40 | 409 | 52,40% | 50 | 854 | 69,70% | 60000 |
| 40 | 780 | 100,00% | 41 | 781 | 95,20% | 2 |
| 40 | 780 | 100,00% | 41 | 800 | 97,60% | 1 |
| 40 | 780 | 100,00% | 41 | 820 | 100,00% | 1 |
| 40 | 780 | 100,00% | 45 | 785 | 79,30% | 2 |
| 40 | 780 | 100,00% | 45 | 887 | 89,60% | 2 |
| 40 | 780 | 100,00% | 45 | 990 | 100,00% | 1 |
| 40 | 780 | 100,00% | 50 | 790 | 64,50% | 1 |
| 40 | 780 | 100,00% | 50 | 1007 | 82,20% | 1 |
| 40 | 780 | 100,00% | 50 | 1225 | 100,00% | 2 |

***Tabla 5.4.c.*** *Resultados de la ejecución del algoritmo de Ullman paralelizado*

## 5.5.2. Valoración del algoritmo, estadísticas, observaciones

Tras la ejecución de la experimentación se obtiene que el algoritmo de Ullman paralelizado fue capaz de resolver satisfactoriamente el 56,48% de los casos propuestos en tan solo 6 minutos y 18 segundos, mientras que no puedo resolver el 43,52% restante habiendo demorado en este conjunto 47 minutos

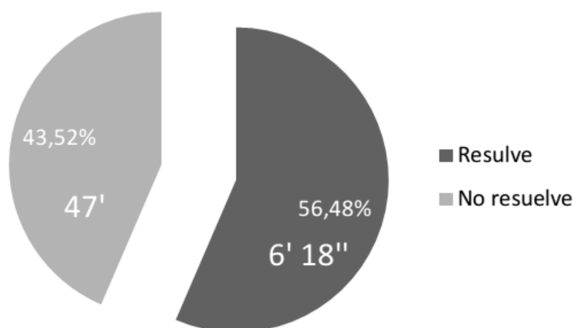



***Figura 5.13.*** *Grafico circular. Muestra los porcentajes de casos resueltos y no resueltos.*



Repositorio Digital Institucional "José María Rosa"
UNLa.

En las *figuras 5.14 y 5.15* se muestran los gráficos de dispersión sobre los casos que el algoritmo no pudo resolver y de los que sí pudo resolver respectivamente:

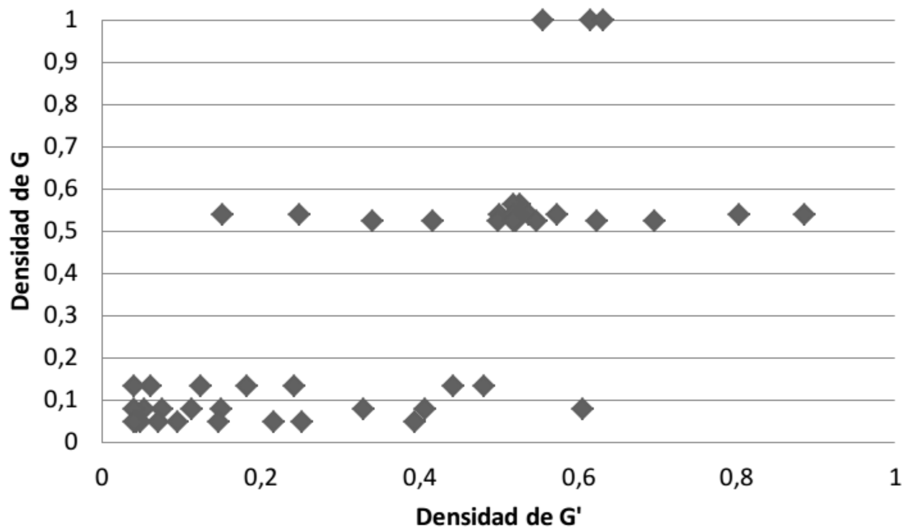



***Figura 5.14.*** *Gráfico de dispersión – Ullman paralelizado. Muestra los casos no resueltos en función de las densidades de G y G'.*

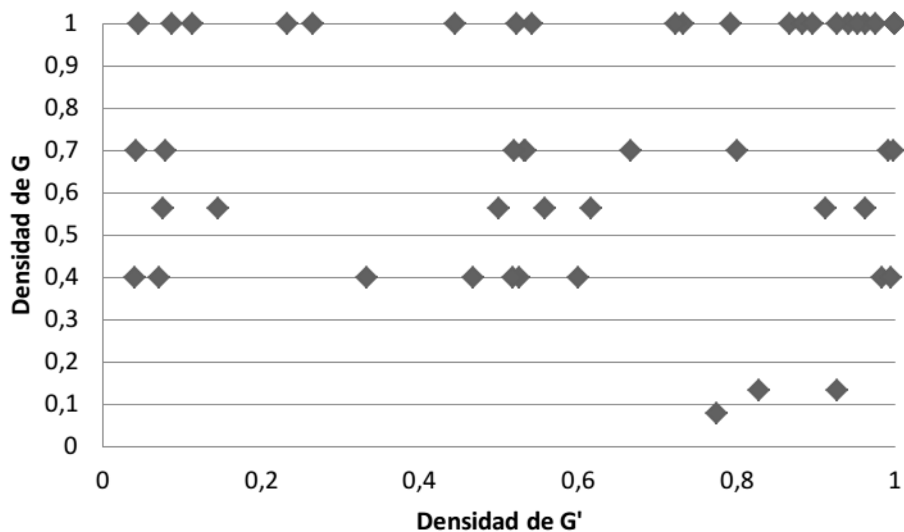



***Figura 5.15.*** *Gráfico de dispersión – Ullman Paralelizado. Muestra los casos resueltos en función de las densidades de G y G'.*

Como se observa a partir de las *figuras 5.14 y 5.15*, el algoritmo de Ullman paralelizado tiene un comportamiento casi idéntico al de su contracara lineal. No presenta mayores diferencias en cuanto a su capacidad para resolver los problemas. Pero, si bien ambos lograron resolver un poco más 50% de los casos, el algoritmo de Ullman paralelo presenta una pequeña mejoría porcentual.



Repositorio Digital Institucional "José María Rosa"
UNLa.

## 5.6. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO G-MATRIX

En esta sección se presentan los resultados obtenidos a raíz de la ejecución del algoritmo G-Matrix (ver 4.2) para cada caso de experimentación propuesto en la tabla 5.2. En la sección 5.6.1 se exponen los resultados de la tabla. En la sección 5.6.2 se realizan observaciones sobre el desempeño del algoritmo.

### 5.6.1. Tabla de resultados

Se presenta la tabla 5.5 de resultados del algoritmo G-Matrix. Las columnas debajo de $G$ y $G'$ representan los parámetros de los grafos construidos. "Tiempo en ms" representa el tiempo promedio que demoró el algoritmo, obtenido del tiempo total de ejecución divido diez (que fue la precisión utilizada). El resultado "60000" significa que el algoritmo no fue capaz de hallar una solución dentro de los límites de tiempo estipulados.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 5 | 4 | 40,00% | 6 | 5 | 33,30% | 1 |
| 5 | 4 | 40,00% | 6 | 7 | 46,70% | 0 |
| 5 | 4 | 40,00% | 6 | 9 | 60,00% | 0 |
| 5 | 4 | 40,00% | 28 | 27 | 7,10% | 0 |
| 5 | 4 | 40,00% | 28 | 199 | 52,60% | 0 |
| 5 | 4 | 40,00% | 28 | 372 | 98,40% | 0 |
| 5 | 4 | 40,00% | 50 | 49 | 4,00% | 0 |
| 5 | 4 | 40,00% | 50 | 634 | 51,80% | 0 |
| 5 | 4 | 40,00% | 50 | 1219 | 99,50% | 0 |
| 5 | 7 | 70,00% | 6 | 8 | 53,30% | 0 |
| 5 | 7 | 70,00% | 6 | 10 | 66,70% | 0 |
| 5 | 7 | 70,00% | 6 | 12 | 80,00% | 0 |
| 5 | 7 | 70,00% | 28 | 30 | 7,90% | 0 |
| 5 | 7 | 70,00% | 28 | 202 | 53,40% | 0 |
| 5 | 7 | 70,00% | 28 | 375 | 99,20% | 0 |
| 5 | 7 | 70,00% | 50 | 52 | 4,20% | 0 |
| 5 | 7 | 70,00% | 50 | 637 | 52,00% | 1 |
| 5 | 7 | 70,00% | 50 | 1222 | 99,80% | 0 |
| 5 | 10 | 100,00% | 6 | 11 | 73,30% | 0 |
| 5 | 10 | 100,00% | 6 | 13 | 86,70% | 0 |
| 5 | 10 | 100,00% | 6 | 15 | 100,00% | 0 |

***Tabla 5.5.a.*** *Resultados de la ejecución del algoritmo G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 5 | 10 | 100,00% | 28 | 33 | 8,70% | 0 |
| 5 | 10 | 100,00% | 28 | 205 | 54,20% | 1 |
| 5 | 10 | 100,00% | 28 | 378 | 100,00% | 0 |
| 5 | 10 | 100,00% | 50 | 55 | 4,50% | 0 |
| 5 | 10 | 100,00% | 50 | 640 | 52,20% | 4 |
| 5 | 10 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 15 | 14 | 13,30% | 16 | 15 | 12,50% | 0 |
| 15 | 14 | 13,30% | 16 | 22 | 18,30% | 0 |
| 15 | 14 | 13,30% | 16 | 29 | 24,20% | 0 |
| 15 | 14 | 13,30% | 33 | 32 | 6,10% | 0 |
| 15 | 14 | 13,30% | 33 | 234 | 44,30% | 0 |
| 15 | 14 | 13,30% | 33 | 437 | 82,80% | 0 |
| 15 | 14 | 13,30% | 50 | 49 | 4,00% | 0 |
| 15 | 14 | 13,30% | 50 | 591 | 48,20% | 0 |
| 15 | 14 | 13,30% | 50 | 1134 | 92,60% | 0 |
| 15 | 59 | 56,20% | 16 | 60 | 50,00% | 30 |
| 15 | 59 | 56,20% | 16 | 67 | 55,80% | 826 |
| 15 | 59 | 56,20% | 16 | 74 | 61,70% | 8179 |
| 15 | 59 | 56,20% | 33 | 77 | 14,60% | 174 |
| 15 | 59 | 56,20% | 33 | 279 | 52,80% | 60000 |
| 15 | 59 | 56,20% | 33 | 482 | 91,30% | 36880 |
| 15 | 59 | 56,20% | 50 | 94 | 7,70% | 20 |
| 15 | 59 | 56,20% | 50 | 636 | 51,90% | 54092 |
| 15 | 59 | 56,20% | 50 | 1179 | 96,20% | 24535 |
| 15 | 105 | 100,00% | 16 | 106 | 88,30% | 0 |
| 15 | 105 | 100,00% | 16 | 113 | 94,20% | 0 |
| 15 | 105 | 100,00% | 16 | 120 | 100,00% | 0 |
| 15 | 105 | 100,00% | 33 | 123 | 23,30% | 0 |
| 15 | 105 | 100,00% | 33 | 325 | 61,60% | 60000 |
| 15 | 105 | 100,00% | 33 | 528 | 100,00% | 0 |
| 15 | 105 | 100,00% | 50 | 140 | 11,40% | 0 |
| 15 | 105 | 100,00% | 50 | 682 | 55,70% | 60000 |
| 15 | 105 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 25 | 24 | 8,00% | 26 | 25 | 7,70% | 0 |
| 25 | 24 | 8,00% | 26 | 37 | 11,40% | 1 |
| 25 | 24 | 8,00% | 26 | 49 | 15,10% | 1 |
| 25 | 24 | 8,00% | 38 | 37 | 5,30% | 0 |
| 25 | 24 | 8,00% | 38 | 232 | 33,00% | 0 |
| 25 | 24 | 8,00% | 38 | 427 | 60,70% | 0 |
| 25 | 24 | 8,00% | 50 | 49 | 4,00% | 0 |
| 25 | 24 | 8,00% | 50 | 499 | 40,70% | 0 |
| 25 | 24 | 8,00% | 50 | 949 | 77,50% | 0 |
| 25 | 162 | 54,00% | 26 | 163 | 50,20% | 14726 |
| 25 | 162 | 54,00% | 26 | 175 | 53,80% | 39618 |

***Tabla 5.5.b.*** *Resultados de la ejecución del algoritmo G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 25 | 162 | 54,00% | 26 | 187 | 57,50% | 56747 |
| 25 | 162 | 54,00% | 38 | 175 | 24,90% | 13117 |
| 25 | 162 | 54,00% | 38 | 370 | 52,60% | 60000 |
| 25 | 162 | 54,00% | 38 | 565 | 80,40% | 60000 |
| 25 | 162 | 54,00% | 50 | 187 | 15,30% | 28534 |
| 25 | 162 | 54,00% | 50 | 637 | 52,00% | 60000 |
| 25 | 162 | 54,00% | 50 | 1087 | 88,70% | 60000 |
| 25 | 300 | 100,00% | 26 | 301 | 92,60% | 2 |
| 25 | 300 | 100,00% | 26 | 313 | 96,30% | 1 |
| 25 | 300 | 100,00% | 26 | 325 | 100,00% | 1 |
| 25 | 300 | 100,00% | 38 | 313 | 44,50% | 1 |
| 25 | 300 | 100,00% | 38 | 508 | 72,30% | 30004 |
| 25 | 300 | 100,00% | 38 | 703 | 100,00% | 1 |
| 25 | 300 | 100,00% | 50 | 325 | 26,50% | 1 |
| 25 | 300 | 100,00% | 50 | 775 | 63,30% | 60000 |
| 25 | 300 | 100,00% | 50 | 1225 | 100,00% | 2 |
| 40 | 39 | 5,00% | 41 | 40 | 4,90% | 1 |
| 40 | 39 | 5,00% | 41 | 59 | 7,20% | 10 |
| 40 | 39 | 5,00% | 41 | 79 | 9,60% | 13 |
| 40 | 39 | 5,00% | 45 | 44 | 4,40% | 0 |
| 40 | 39 | 5,00% | 45 | 146 | 14,70% | 246 |
| 40 | 39 | 5,00% | 45 | 249 | 25,20% | 12090 |
| 40 | 39 | 5,00% | 50 | 49 | 4,00% | 0 |
| 40 | 39 | 5,00% | 50 | 266 | 21,70% | 18 |
| 40 | 39 | 5,00% | 50 | 484 | 39,50% | 267 |
| 40 | 409 | 52,40% | 41 | 410 | 50,00% | 56215 |
| 40 | 409 | 52,40% | 41 | 429 | 52,30% | 45336 |
| 40 | 409 | 52,40% | 41 | 449 | 54,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 414 | 41,80% | 49023 |
| 40 | 409 | 52,40% | 45 | 516 | 52,10% | 60000 |
| 40 | 409 | 52,40% | 45 | 619 | 62,50% | 60000 |
| 40 | 409 | 52,40% | 50 | 419 | 34,20% | 50569 |
| 40 | 409 | 52,40% | 50 | 636 | 51,90% | 60000 |
| 40 | 409 | 52,40% | 50 | 854 | 69,70% | 60000 |
| 40 | 780 | 100,00% | 41 | 781 | 95,20% | 8 |
| 40 | 780 | 100,00% | 41 | 800 | 97,60% | 6 |
| 40 | 780 | 100,00% | 41 | 820 | 100,00% | 7 |
| 40 | 780 | 100,00% | 45 | 785 | 79,30% | 7 |
| 40 | 780 | 100,00% | 45 | 887 | 89,60% | 7 |
| 40 | 780 | 100,00% | 45 | 990 | 100,00% | 8 |
| 40 | 780 | 100,00% | 50 | 790 | 64,50% | 7 |
| 40 | 780 | 100,00% | 50 | 1007 | 82,20% | 8 |
| 40 | 780 | 100,00% | 50 | 1225 | 100,00% | 7 |

***Tabla 5.5.c.*** *Resultados de la ejecución del algoritmo G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

### 5.6.2. Valoración del algoritmo, estadísticas, observaciones

Tras la ejecución de la experimentación se obtiene que el algoritmo G-Matrix fue capaz de resolver satisfactoriamente el 88% de los casos propuestos en tan solo 8 minutos y 41 segundos, mientras que no puedo resolver el 12% restante habiendo demorado en este conjunto 13 minutos.

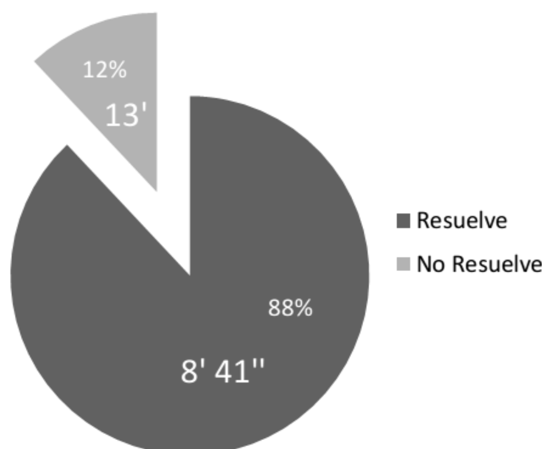



***Figura 5.16.*** *Grafico circular – G-Matrix. Muestra los porcentajes de casos resueltos y no resueltos.*

La *figura 5.17* muestra los casos que el algoritmo G-Matrix no fue capaz de resolver dentro de los tiempos estipulados:

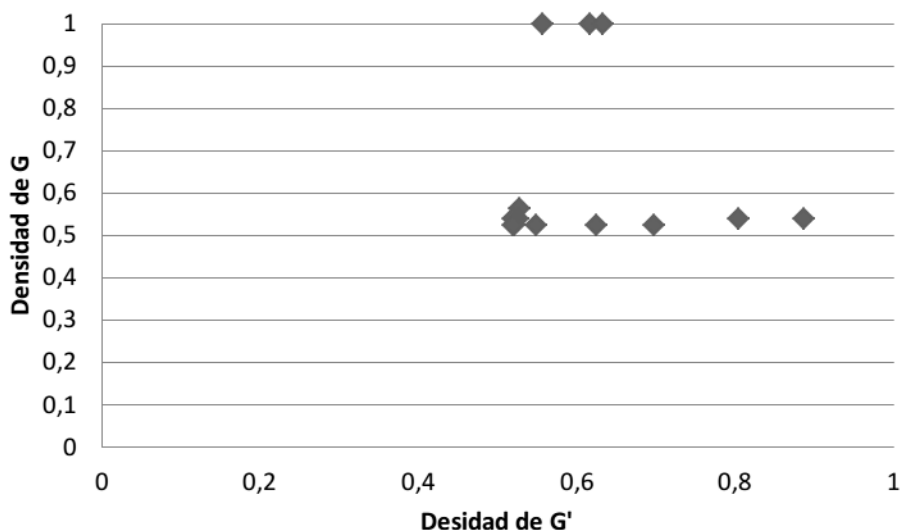



***Figura 5.17.*** *Gráfico de dispersión – G-Matrix. Muestra los casos no resueltos en función de las densidades de G y G'.*

Se observa claramente que los casos donde $G$ y $G'$ están entre un 50% y 60% de densidad respectivamente, suponen la mayor dificultad para algoritmo, haciendo que este no sea capaz de encontrar una solución.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Se analiza la *figura 5.18* que muestra los casos que el algoritmo G-Matrix sí fue capaz de resolver dentro de los tiempos estipulados:

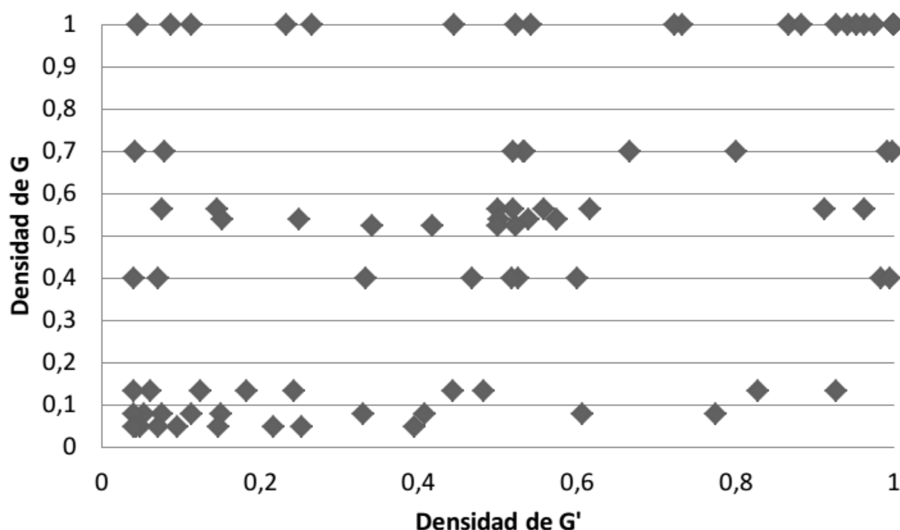



***Figura 5.18.*** *Gráfico de dispersión – G-Matrix. Muestra los casos resueltos en función de las densidades de G y G'.*

Se observa que el algoritmo es capaz de desempeñarse correctamente para la mayoría de las combinaciones de densidades de entrada, teniendo además una ligera facilidad para resolver casos donde tanto $G$ como $G'$ se encuentran ente 5% y 20% de densidad, así como también, casos donde $G$ y $G'$ se encuentran entre un 80% y 100% de densidad.

# 5.7. RESULTADOS OBTENIDOS DE LA EJECUCIÓN DEL ALGORITMO G-MATRIX APLICANDO PARALELISMO.

En esta sección se presentan los resultados obtenidos a raíz de la ejecución del algoritmo G-Matrix (ver 4.2) aplicando el método de paralelización propuesto en 4.3 para cada caso de experimentación de la tabla 5.2. En la sección 5.7.1 se exponen los resultados de la tabla. En la sección 5.7.2 se realizan observaciones sobre el desempeño del algoritmo.

## 5.7.1. Tabla de resultados

Se presenta la *tabla 5.6* de resultados del algoritmo G-Matrix paralelizado. Las columnas debajo de $G$ y $G'$ representan los parámetros de los grafos construidos. "Tiempo en ms" representa el tiempo promedio que demoró el algoritmo, obtenido del tiempo total de ejecución divido diez (que fue la



Repositorio Digital Institucional "José María Rosa"
UNLa.

precisión utilizada). El resultado “60000” significa que el algoritmo no fue capaz de hallar una solución dentro de los límites de tiempo estipulados.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 5 | 4 | 40,00% | 6 | 5 | 33,30% | 1 |
| 5 | 4 | 40,00% | 6 | 7 | 46,70% | 1 |
| 5 | 4 | 40,00% | 6 | 9 | 60,00% | 0 |
| 5 | 4 | 40,00% | 28 | 27 | 7,10% | 1 |
| 5 | 4 | 40,00% | 28 | 199 | 52,60% | 1 |
| 5 | 4 | 40,00% | 28 | 372 | 98,40% | 0 |
| 5 | 4 | 40,00% | 50 | 49 | 4,00% | 0 |
| 5 | 4 | 40,00% | 50 | 634 | 51,80% | 0 |
| 5 | 4 | 40,00% | 50 | 1219 | 99,50% | 0 |
| 5 | 7 | 70,00% | 6 | 8 | 53,30% | 0 |
| 5 | 7 | 70,00% | 6 | 10 | 66,70% | 1 |
| 5 | 7 | 70,00% | 6 | 12 | 80,00% | 0 |
| 5 | 7 | 70,00% | 28 | 30 | 7,90% | 0 |
| 5 | 7 | 70,00% | 28 | 202 | 53,40% | 0 |
| 5 | 7 | 70,00% | 28 | 375 | 99,20% | 0 |
| 5 | 7 | 70,00% | 50 | 52 | 4,20% | 0 |
| 5 | 7 | 70,00% | 50 | 637 | 52,00% | 1 |
| 5 | 7 | 70,00% | 50 | 1222 | 99,80% | 0 |
| 5 | 10 | 100,00% | 6 | 11 | 73,30% | 0 |
| 5 | 10 | 100,00% | 6 | 13 | 86,70% | 0 |
| 5 | 10 | 100,00% | 6 | 15 | 100,00% | 0 |
| 5 | 10 | 100,00% | 28 | 33 | 8,70% | 0 |
| 5 | 10 | 100,00% | 28 | 205 | 54,20% | 1 |
| 5 | 10 | 100,00% | 28 | 378 | 100,00% | 0 |
| 5 | 10 | 100,00% | 50 | 55 | 4,50% | 0 |
| 5 | 10 | 100,00% | 50 | 640 | 52,20% | 3 |
| 5 | 10 | 100,00% | 50 | 1225 | 100,00% | 0 |
| 15 | 14 | 13,30% | 16 | 15 | 12,50% | 0 |
| 15 | 14 | 13,30% | 16 | 22 | 18,30% | 0 |
| 15 | 14 | 13,30% | 16 | 29 | 24,20% | 0 |
| 15 | 14 | 13,30% | 33 | 32 | 6,10% | 0 |
| 15 | 14 | 13,30% | 33 | 234 | 44,30% | 0 |
| 15 | 14 | 13,30% | 33 | 437 | 82,80% | 0 |
| 15 | 14 | 13,30% | 50 | 49 | 4,00% | 1 |
| 15 | 14 | 13,30% | 50 | 591 | 48,20% | 1 |
| 15 | 14 | 13,30% | 50 | 1134 | 92,60% | 1 |
| 15 | 59 | 56,20% | 16 | 60 | 50,00% | 38 |
| 15 | 59 | 56,20% | 16 | 67 | 55,80% | 1081 |
| 15 | 59 | 56,20% | 16 | 74 | 61,70% | 12232 |

***Tabla 5.6.a.*** *Resultados de la ejecución del algoritmo G-Matrix paralelizado*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 15 | 59 | 56,20% | 33 | 77 | 14,60% | 193 |
| 15 | 59 | 56,20% | 33 | 279 | 52,80% | 60000 |
| 15 | 59 | 56,20% | 33 | 482 | 91,30% | 24037 |
| 15 | 59 | 56,20% | 50 | 94 | 7,70% | 16 |
| 15 | 59 | 56,20% | 50 | 636 | 51,90% | 60000 |
| 15 | 59 | 56,20% | 50 | 1179 | 96,20% | 12073 |
| 15 | 105 | 100,00% | 16 | 106 | 88,30% | 1 |
| 15 | 105 | 100,00% | 16 | 113 | 94,20% | 0 |
| 15 | 105 | 100,00% | 16 | 120 | 100,00% | 0 |
| 15 | 105 | 100,00% | 33 | 123 | 23,30% | 1 |
| 15 | 105 | 100,00% | 33 | 325 | 61,60% | 60000 |
| 15 | 105 | 100,00% | 33 | 528 | 100,00% | 0 |
| 15 | 105 | 100,00% | 50 | 140 | 11,40% | 1 |
| 15 | 105 | 100,00% | 50 | 682 | 55,70% | 60000 |
| 15 | 105 | 100,00% | 50 | 1225 | 100,00% | 1 |
| 25 | 24 | 8,00% | 26 | 25 | 7,70% | 1 |
| 25 | 24 | 8,00% | 26 | 37 | 11,40% | 1 |
| 25 | 24 | 8,00% | 26 | 49 | 15,10% | 1 |
| 25 | 24 | 8,00% | 38 | 37 | 5,30% | 1 |
| 25 | 24 | 8,00% | 38 | 232 | 33,00% | 1 |
| 25 | 24 | 8,00% | 38 | 427 | 60,70% | 1 |
| 25 | 24 | 8,00% | 50 | 49 | 4,00% | 1 |
| 25 | 24 | 8,00% | 50 | 499 | 40,70% | 1 |
| 25 | 24 | 8,00% | 50 | 949 | 77,50% | 1 |
| 25 | 162 | 54,00% | 26 | 163 | 50,20% | 19658 |
| 25 | 162 | 54,00% | 26 | 175 | 53,80% | 39088 |
| 25 | 162 | 54,00% | 26 | 187 | 57,50% | 53260 |
| 25 | 162 | 54,00% | 38 | 175 | 24,90% | 10700 |
| 25 | 162 | 54,00% | 38 | 370 | 52,60% | 60000 |
| 25 | 162 | 54,00% | 38 | 565 | 80,40% | 60000 |
| 25 | 162 | 54,00% | 50 | 187 | 15,30% | 20913 |
| 25 | 162 | 54,00% | 50 | 637 | 52,00% | 60000 |
| 25 | 162 | 54,00% | 50 | 1087 | 88,70% | 60000 |
| 25 | 300 | 100,00% | 26 | 301 | 92,60% | 2 |
| 25 | 300 | 100,00% | 26 | 313 | 96,30% | 2 |
| 25 | 300 | 100,00% | 26 | 325 | 100,00% | 2 |
| 25 | 300 | 100,00% | 38 | 313 | 44,50% | 2 |
| 25 | 300 | 100,00% | 38 | 508 | 72,30% | 18052 |
| 25 | 300 | 100,00% | 38 | 703 | 100,00% | 2 |
| 25 | 300 | 100,00% | 50 | 325 | 26,50% | 2 |
| 25 | 300 | 100,00% | 50 | 775 | 63,30% | 60000 |
| 25 | 300 | 100,00% | 50 | 1225 | 100,00% | 2 |
| 40 | 39 | 5,00% | 41 | 40 | 4,90% | 1 |

***Tabla 5.6.b.*** *Resultados de la ejecución del algoritmo G-Matrix paralelizado*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| G | | | G' | | | Tiempo en ms |
|---|---|---|---|---|---|---|
| V | E | Densidad | V' | E' | Densidad' | |
| 40 | 39 | 5,00% | 41 | 59 | 7,20% | 5 |
| 40 | 39 | 5,00% | 41 | 79 | 9,60% | 12 |
| 40 | 39 | 5,00% | 45 | 44 | 4,40% | 1 |
| 40 | 39 | 5,00% | 45 | 146 | 14,70% | 87 |
| 40 | 39 | 5,00% | 45 | 249 | 25,20% | 979 |
| 40 | 39 | 5,00% | 50 | 49 | 4,00% | 1 |
| 40 | 39 | 5,00% | 50 | 266 | 21,70% | 3 |
| 40 | 39 | 5,00% | 50 | 484 | 39,50% | 87 |
| 40 | 409 | 52,40% | 41 | 410 | 50,00% | 57223 |
| 40 | 409 | 52,40% | 41 | 429 | 52,30% | 45783 |
| 40 | 409 | 52,40% | 41 | 449 | 54,80% | 60000 |
| 40 | 409 | 52,40% | 45 | 414 | 41,80% | 46683 |
| 40 | 409 | 52,40% | 45 | 516 | 52,10% | 60000 |
| 40 | 409 | 52,40% | 45 | 619 | 62,50% | 60000 |
| 40 | 409 | 52,40% | 50 | 419 | 34,20% | 46931 |
| 40 | 409 | 52,40% | 50 | 636 | 51,90% | 60000 |
| 40 | 409 | 52,40% | 50 | 854 | 69,70% | 60000 |
| 40 | 780 | 100,00% | 41 | 781 | 95,20% | 6 |
| 40 | 780 | 100,00% | 41 | 800 | 97,60% | 6 |
| 40 | 780 | 100,00% | 41 | 820 | 100,00% | 6 |
| 40 | 780 | 100,00% | 45 | 785 | 79,30% | 6 |
| 40 | 780 | 100,00% | 45 | 887 | 89,60% | 6 |
| 40 | 780 | 100,00% | 45 | 990 | 100,00% | 6 |
| 40 | 780 | 100,00% | 50 | 790 | 64,50% | 6 |
| 40 | 780 | 100,00% | 50 | 1007 | 82,20% | 6 |
| 40 | 780 | 100,00% | 50 | 1225 | 100,00% | 5 |

***Tabla 5.6.c.*** *Resultados de la ejecución del algoritmo G-Matrix paralelizado*

## 5.7.2. Valoración del algoritmo, estadísticas, observaciones

Tras la ejecución de la experimentación se obtiene que el algoritmo G-Matrix paralelizado fue capaz de resolver satisfactoriamente el 87% de los casos propuestos en tan solo 6 minutos y 49 segundos, mientras que no puedo resolver el 13% restante habiendo demorado en este conjunto 14 minutos.



Repositorio Digital Institucional "José María Rosa"
UNLa.

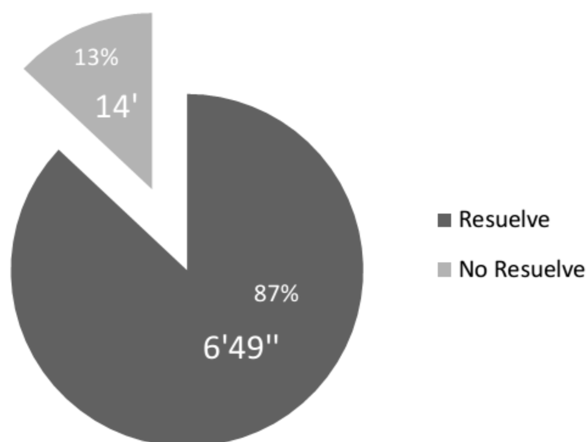



***Figura 5.19.*** *Gráfico circular – G-Matrix paralelizado. Muestra los porcentajes de casos resueltos y no resueltos.*

La *figura 5.20* muestra los casos que el algoritmo G-Matrix paralelizado no fue capaz de resolver dentro de los tiempos estipulados:

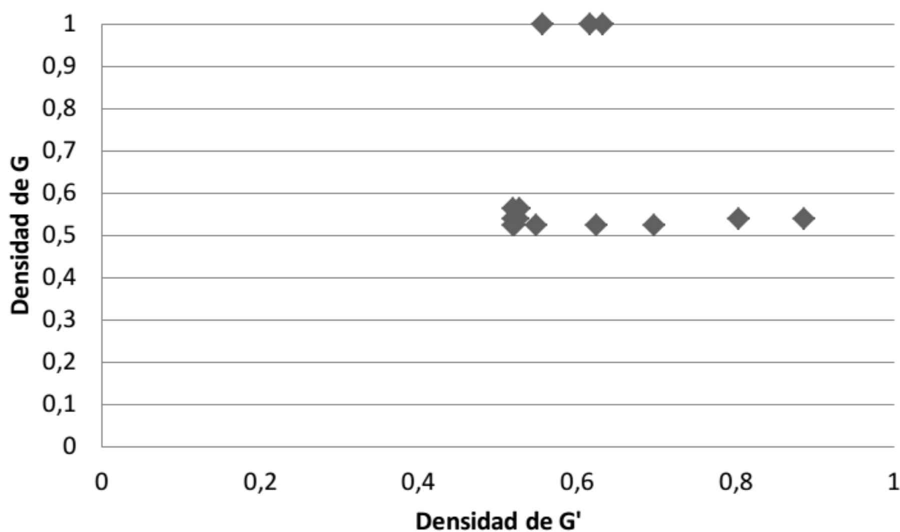



***Figura 5.20.*** *Gráfico de dispersión – G-Matrix paralelizado. Muestra los casos resueltos en función de las densidades de G y G'.*

Al igual que su contracara lineal, los casos donde *G* y *G'* están entre un 50% y 60% de densidad respectivamente, suponen la mayor dificultad para algoritmo haciendo que este no sea capaz de encontrar una solución.

Se analiza la *figura 5.21* que muestra los casos que el algoritmo G-Matrix sí fue capaz de resolver dentro de los tiempos estipulados:



Repositorio Digital Institucional "José María Rosa"
UNLa.

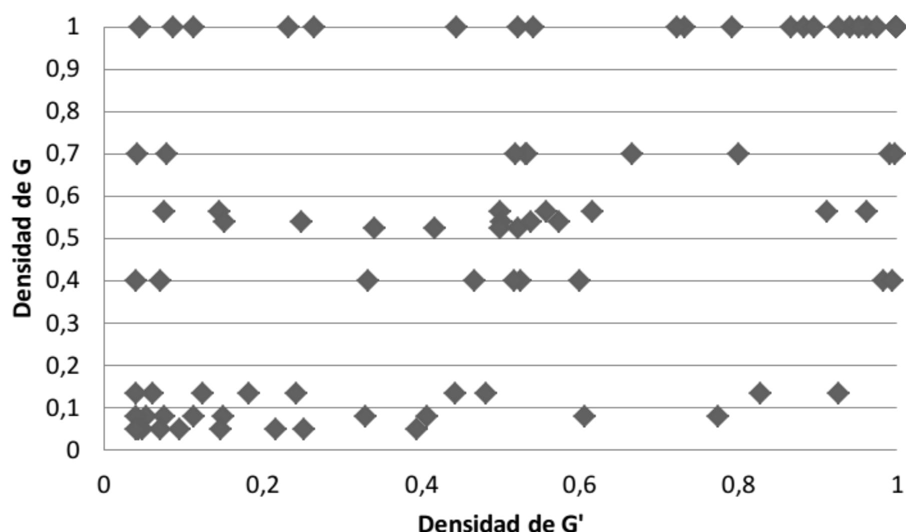



***Figura 5.21.*** *Gráfico de dispersión – G-Matrix paralelizado. Muestra los casos resueltos en función de las densidades de G y G'.*

Se observa que el algoritmo es capaz de desempeñarse correctamente para la mayoría de las combinaciones de densidades de entrada, teniendo además una ligera facilidad para resolver casos donde tanto $G$ como $G'$ se encuentran ente 5% y 20% de densidad, así como también, casos donde $G$ y $G'$ se encuentran entre un 80% y 100% de densidad. Se destaca que dentro de los casos que pudieron resolverse, el algoritmo paralelizado se desempeñó significativamente más rápido que su contracara lineal.

# 5.8. COMPARACIÓN Y EVALUACIÓN DE LOS RESULTADOS OBTENIDOS

En esta sección se da lugar a las comparaciones necesarias para contrastar el desempeño de los algoritmos evaluados en este Trabajo Final de Licenciatura. En la sección 5.8.1 se realiza un primer acercamiento a la comparación de resultados y se presenta una tabla con los porcentajes de éxito de los algoritmos para resolver los casos de prueba. En la sección 5.8.2 por medio del test de Wilcoxon, se busca comprobar la superioridad del algoritmo G-Matrix con respecto al algoritmo de Ullman, así como también, comprobar que el método de paralelización propuesto mejora el rendimiento de los algoritmos con respecto a sus versiones lineales. Finalmente en la sección 5.8.3 se confeccionan apreciaciones sobre los resultados de los test.



Repositorio Digital Institucional "José María Rosa"
UNLa.

## 5.8.1. Tabla comparativa de casos con resultado

La *tabla 5.7* contrasta el porcentaje de casos de experimentación que cada algoritmo fue capaz de resolver dentro de los límites de tiempo establecidos y funciona como una medida de la eficacia de los mismos.

| Algoritmo | % de casos que resuelve |
|---|---|
| Ullman | 55,55% |
| Ullman paralelo | 56,48% |
| G-Matrix | 88% |
| G-Matrix Paralelo | 87% |

***Tabla 5.7.*** *Algoritmo – Porcentaje. Muestra el porcentaje de casos con solución para cada algoritmo*

La *tabla 5.7* permite hacer una observación a grandes rasgos del desempeño de los algoritmos. En ella se destaca que el algoritmo G-Matrix propuesto para este Trabajo Final de Licenciatura es capaz de resolver hasta un 32% más de casos de prueba que el algoritmo de Ullman. En términos cualitativos G-Matrix se desempeñó 58% mejor que Ullman.
Por otro lado, nótese, que si bien G-Matrix lineal resolvió un 1% más de casos que G-Matrix paralelo, este último requirió significativamente menos tiempo de ejecución.

## 5.8.2. Tests de Wilcoxon

En esta sección se realizan los test de Wilcoxon para comprobar la eficacia de las aportaciones propuestas en este Trabajo Final de Licenciatura. En la sección 5.8.2.1 se profundizan aspectos a considerar sobre el test de Wilcoxon que será llevado acabo. En la sección 5.8.2.2 se compara el algoritmo G-Matrix contra el algoritmo de Ullman. En la sección 5.8.2.3 se compara Ullman contra Ullman paralelo. Finalmente, en la sección 5.8.2.4 se compara G-Matrix contra G-Matrix paralelo.

### 5.8.2.1. Generalidades

La prueba de los rangos con signo de Wilcoxon es una prueba no paramétrica para comparar la mediana de dos muestras relacionadas y determinar si existen diferencias entre ellas. Se utiliza como alternativa a la “prueba t” de Student cuando no se puede suponer la normalidad de dichas muestras.
Para realizar este test se procede de la siguiente manera:



Repositorio Digital Institucional "José María Rosa"
UNLa.

Primero se establecen las hipótesis, que generalmente consisten en una hipótesis neutra que es la que se busca refutar y una hipótesis alternativa que puede enunciar o bien que existen diferencias entre las muestras, o que la mediana de las muestras de un grupo es significativamente mayor (o menor) que la mediana de las muestras del otro grupo. En nuestro caso la hipótesis alternativa será direccional y enuncia que la mediana de las muestras del grupo de control *"a"* es significativamente mayor que la mediana de las muestras del grupo experimental *"b".*

$$H_0: M_e\, a = M_e\, b$$

$$H_a: M_e\, a > M_e\, b$$

Luego se procede a generar una tabla con las muestras, a la izquierda se sitúan las muestras de control *"a"* y a la derecha las muestras experimentales *"b".* Se calculan las diferencias para *Xa - Xb* y a aquellas diferencias "no neutras" se las ordenan según valor absoluto y se les asigna un rango de orden.

Nótese que en nuestro caso aquellas diferencias en tiempo menores a diez milisegundos fueron tomadas como neutras y excluidas de los resultados porque no se considera que aporten una cantidad significativa de información como para ser tenidas en cuenta en este test.

Finalmente se reemplazan las diferencias por los rangos asignados respetando el signo positivo o negativo que poseía la diferencia originalmente.

| a | b | Diferencias | Rangos asignados |
|---|---|---|---|
| Xa | Xb | (Xa - Xb) | (+;-)Rango |
| Xa | Xb | (Xa - Xb) | (+;-)Rango |
| Xa | Xb | (Xa - Xb) | (+;-)Rango |
| ……. | …….. | …… | …… |

***Tabla 5.8.*** *Estructura de una tabla para Test de Wilcoxon*

En nuestro caso, se evalúan los resultados de la tabla Wilcoxon por medio de una aproximación a la normal:

$$Z = \begin{cases} \dfrac{(W - \mu_w) - 0{,}5}{\sigma_w} \; Si\, W \geq \mu_w \\ \\ \dfrac{(W - \mu_w) + 0{,}5}{\sigma_w} \; Si\, W < \mu_w \end{cases} \tag{5.13}$$



Repositorio Digital Institucional "José María Rosa"
UNLa.

Donde:

$W$ es la sumatoria de los valores pertenecientes a la columna de rangos asignados, formalmente:

$$W = \sum(rangos\ asignados) \quad (5.14)$$

$\mu_w$ es la media esperada para $W$. Como se había establecido anteriormente, en la hipótesis nula se esperaría que el valor de $W$ se aproximara a cero, dentro de los límites de variabilidad aleatorios. Esto es equivalente a decir que cualquier valor particular de $W$ pertenece a una distribución muestral cuya media es igual a cero [Vassarstats, 2014], formalmente:

$$\mu_w = 0 \quad (5.15)$$

Finalmente $\sigma_w$ es la desviación estándar esperada para $W$ y se calcula a través de:

$$\sigma_w = \sqrt[2]{\frac{n(n+1)(2n+1)}{6}} \quad (5.16)$$

Donde:

$$n = (cantidad\ de\ muestras\ con\ valor) \quad (5.17)$$

Por último, una vez obtenido el valor de $Z$, solo queda compararlo en la tabla 5.9 de valores críticos de $Z$ para determinar su significancia estadística.

| Nivel de significancia estadística para α | | | | | |
|---|---|---|---|---|---|
| Test direccional | | | | | |
| 0.1 | 0.05 | 0.025 | 0.01 | 0.005 | 0.0005 |
| Test no direccional | | | | | |
| 0.2 | 0.1 | 0.05 | 0.02 | 0.01 | 0.001 |
| Z crítico | | | | | |
| 1.282 | 1.645 | 1.960 | 2.326 | 2.576 | 3.291 |

***Tabla 5.9.*** *Valores críticos de Z*



Repositorio Digital Institucional "José María Rosa"
UNLa.

#### 5.8.2.2. Ullman Vs G-Matrix

A continuación se realiza el test de Wilcoxon, como medio de comparación no-paramétrico entre el algoritmo de Ullman contra el algoritmo G-Matrix. Se busca comprobar que la mediana de los tiempos de ejecución del algoritmo de Ullman es significativamente mayor a la mediana de G-Matrix, asegurando así que este último tiene un mejor desempeño estadístico. Se busca como mínimo un 95% de confianza de que el resultado no sea por azar ($\alpha \leq 0{,}05$).
Nuestras hipótesis son:

$$H_0\colon M_e\, Ullman\ =\ M_e\, GMatrix$$

$$H_a\colon M_e\, Ullman\ >M_e\, GMatrix$$

Nótese que al momento de realizar el test las diferencias de resultados entre ambos algoritmos menores a 10 milisegundos fueron omitidas porque no se considera que aporten una cantidad de información significativa.
Se muestra la *tabla 5.10* con los rangos asignados, las columnas con los nombres de los algoritmos corresponden a los tiempos en milisegundos obtenidos de la ejecución de los casos de prueba, solo se muestran las filas cuya diferencia en tiempos de ejecución sea significativa, las filas con diferencias neutras fueron descartadas

| Ullman | G-Matrix | Diferencias | Rangos asignados |
|---|---|---|---|
| 1573 | 0 | 1573 | 8 |
| 199 | 0 | 199 | 3 |
| 37012 | 0 | 37012 | 22 |
| 1158 | 0 | 1158 | 5 |
| 264 | 0 | 264 | 4 |
| 1512 | 1 | 1511 | 7 |
| 152 | 1 | 151 | 2 |
| 1450 | 4 | 1446 | 6 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 42042 | 0 | 42042 | 24 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |

***Tabla 5.10.a.*** *Wilcoxon, Ullman vs. G-Matrix*



Repositorio Digital Institucional "José María Rosa"
UNLa.

| Ullman | G-Matrix | Diferencias | Rangos asignados |
|---|---|---|---|
| 42000 | 0 | 42000 | 23 |
| 9469 | 30 | 9439 | 15 |
| 50369 | 826 | 49543 | 28 |
| 60000 | 8179 | 51821 | 29 |
| 3198 | 174 | 3024 | 10 |
| 54040 | 36880 | 17160 | 19 |
| 2798 | 20 | 2778 | 9 |
| 60000 | 54092 | 5908 | 13 |
| 24455 | 24535 | -80 | -1 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 1 | 59999 | 37 |
| 60000 | 1 | 59999 | 37 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 0 | 60000 | 46 |
| 54000 | 0 | 54000 | 30 |
| 60000 | 14726 | 45274 | 25 |
| 60000 | 39618 | 20382 | 20 |
| 60000 | 56747 | 3253 | 11 |
| 60000 | 13117 | 46883 | 26 |
| 60000 | 28534 | 31466 | 21 |
| 42000 | 30004 | 11996 | 17 |
| 60000 | 1 | 59999 | 37 |
| 60000 | 10 | 59990 | 35 |
| 60000 | 13 | 59987 | 34 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 246 | 59754 | 32 |
| 60000 | 12090 | 47910 | 27 |
| 60000 | 0 | 60000 | 46 |
| 60000 | 18 | 59982 | 33 |
| 60000 | 267 | 59733 | 31 |
| 60000 | 56215 | 3785 | 12 |
| 60000 | 45336 | 14664 | 18 |
| 60000 | 49023 | 10977 | 16 |
| 60000 | 50569 | 9431 | 14 |

***Tabla 5.10.b.*** *Wilcoxon, Ullman vs. G-Matrix*

A continuación se procede a calcular las variables que intervienen en el test a partir del procesamiento de la información contenida en la *tabla 5.10* aplicando (5.13), (5.14), (5.15), (5.16) y (5.17):



Repositorio Digital Institucional "José María Rosa"
UNLa.

$W = 1429$

$\mu_W = 0$

$n = 53$

$\sigma_W = 225{,}192$

$Z = 6{,}323$

Finalmente a partir de $Z$, se busca en la *tabla 5.9* y se ve el p-valor para un test unidireccional:

$P\text{-valor} < 0{,}0005$

**Los resultados son significativamente estadísticos, se rechaza $H_0$. Se concluye que la mediana de los tiempos de ejecución de Ullman es mayor que la mediana de los tiempos de ejecución de G-Matrix para** $\alpha \leq 0{,}05$ $(P<0.0005)$.

Dado que con lo anterior se comprueba que efectivamente Ullman demora significativamente más en resolver los mismos problemas que G-Matrix, se puede decir que el desempeño del algoritmo G-Matrix supera al desempeño del algoritmo de Ullman

#### 5.8.2.3. Ullman vs Ullman paralelo

A continuación se realiza el test de Wilcoxon, como medio de comparación no-paramétrico entre el algoritmo de Ullman contra el algoritmo Ullman paralelo. Se busca comprobar que la mediana de los tiempos de ejecución del algoritmo de Ullman es significativamente mayor a la mediana de Ullman paralelo, asegurando así que este último tiene un mejor desempeño estadístico. Se busca como mínimo un 95% de confianza de que el resultado no sea por azar ($\alpha \leq 0{,}05$). Nuestras hipótesis son:

$$H_0\!: M_e\, Ullman = M_e\, Ullman\ paralelo$$

$$H_a\!: M_e\, Ullman > M_e\, Ullman\ paralelo$$

Nótese que al momento de realizar el test las diferencias de resultados entre ambos algoritmos menores a 10 milisegundos fueron omitidas porque no se considera que aporten una cantidad de información significativa.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Se muestra la *tabla 5.11* con los rangos asignados, las columnas con los nombres de los algoritmos corresponden a los tiempos en milisegundos obtenidos de la ejecución de los casos de prueba, solo se muestran las filas cuya diferencia en tiempos de ejecución sea significativa, las filas con diferencias neutras fueron descartadas

| Ullman | Ullman Paralelo | Diferencias | Rangos asignados |
|---|---|---|---|
| 1573 | 558 | 1015 | 9 |
| 199 | 34 | 165 | 3 |
| 37012 | 11630 | 25382 | 18 |
| 1158 | 342 | 816 | 7 |
| 264 | 12 | 252 | 4 |
| 1512 | 79 | 1433 | 10 |
| 152 | 110 | 42 | 1,5 |
| 1450 | 485 | 965 | 8 |
| 42042 | 42000 | 42 | 1,5 |
| 42000 | 36416 | 5584 | 14 |
| 9469 | 15100 | -5631 | -15 |
| 50369 | 48623 | 1746 | 12 |
| 60000 | 54329 | 5671 | 16 |
| 3198 | 4683 | -1485 | -11 |
| 54040 | 48917 | 5123 | 13 |
| 2798 | 3341 | -543 | -6 |
| 24455 | 24002 | 453 | 5 |
| 42000 | 33193 | 8807 | 17 |

***Tabla 5.11.*** *Wilcoxon, Ullman lineal vs. Ullman paralelo*

A continuación se procede a calcular las variables que intervienen en el test a partir del procesamiento de la información contenida en la *tabla 5.11* aplicando (5.13), (5.14), (5.15), (5.16) y (5.17):

$$W = 107$$

$$\mu_W = 0$$

$$n = 18$$

$$\sigma_W = 45{,}923$$

$$Z = 2{,}32$$

Finalmente a partir de Z, se busca en la *tabla 5.9* y se ve el p-valor para un test unidireccional:

$$P\text{-}valor = 0.01$$



Repositorio Digital Institucional "José María Rosa"
UNLa.

**Los resultados son significativamente estadísticos, se rechaza $H_0$. Se concluye que la mediana de los tiempos de ejecución de Ullman es mayor que la mediana de los tiempos de ejecución de Ullman paralelo para** $\alpha \leq 0,05$ $(P=0.01)$.

Dado que con lo anterior se comprueba que efectivamente Ullman lineal demora significativamente más en resolver los mismos problemas que Ullman paralelizado, se puede decir que el método de paralelización propuesto supone una mejora estadística con respecto a su contracara lineal.

#### 5.8.2.4. G-Matrix Vs G-Matrix paralelo

A continuación se realiza el test de Wilcoxon, como medio de comparación no-paramétrico entre el algoritmo G-Matrix contra el algoritmo G-Matrix paralelo. Se busca comprobar que la mediana de los tiempos de ejecución del algoritmo G-Matrix es significativamente mayor a la mediana de G-Matrix paralelo, asegurando así que este último tiene un mejor desempeño estadístico. Se busca como mínimo un 95% de confianza de que el resultado no sea por azar ($\alpha \leq 0,05$). Nuestras hipótesis son:

$$H_0\text{: } M_e\, GMatrix = M_e\, GMatrix\ paralelo$$

$$H_a\text{: } M_e\, GMatrix > M_e\, GMatrix\ paralelo$$

Nótese que al momento de realizar el test las diferencias de resultados entre ambos algoritmos menores a 10 milisegundos fueron omitidas porque no se considera que aporten una cantidad de información significativa.

Se muestra la *tabla 5.12* con los rangos asignados, las columnas con los nombres de los algoritmos corresponden a los tiempos en milisegundos obtenidos de la ejecución de los casos de prueba, solo se muestran las filas cuya diferencia en tiempos de ejecución sea significativa, las filas con diferencias neutras fueron descartadas

| G-Matrix | G-Matrix Paralelo | Diferencias | Rangos asignados |
|---|---|---|---|
| 826 | 1081 | -255 | -5 |
| 8179 | 12232 | -4053 | -13 |
| 174 | 193 | -19 | -2 |
| 36880 | 24037 | 12843 | 20 |

***Tabla 5.12. a.*** *Wilcoxon, G-Matrix lineal vs. G-Matrix paralelo*



Repositorio Digital Institucional "José María Rosa" UNLa.

| G-Matrix | G-Matrix Paralelo | Diferencias | Rangos asignados |
|---|---|---|---|
| 54092 | 60000 | -5908 | -15 |
| 24535 | 12073 | 12462 | 19 |
| 14726 | 19658 | -4932 | -14 |
| 39618 | 39088 | 530 | 7 |
| 56747 | 53260 | 3487 | 11 |
| 13117 | 10700 | 2417 | 10 |
| 28534 | 20913 | 7621 | 16 |
| 30004 | 18052 | 11952 | 18 |
| 246 | 87 | 159 | 3 |
| 12090 | 979 | 11111 | 17 |
| 18 | 3 | 15 | 1 |
| 267 | 87 | 180 | 4 |
| 56215 | 57223 | -1008 | -8 |
| 45336 | 45783 | -447 | -6 |
| 49023 | 46683 | 2340 | 9 |
| 50569 | 46931 | 3638 | 12 |

***Tabla 5.12.b.*** *Wilcoxon, G-Matrix lineal vs. G-Matrix paralelo*

A continuación se procede a calcular las variables que intervienen en el test a partir del procesamiento de la información contenida en la *tabla 5.12* aplicando (5.13), (5.14), (5.15), (5.16) y (5.17):

$W = 84$

$\mu_W = 0$

$n = 20$

$\sigma_W = 53{,}57$

$Z = 1{,}56$

Finalmente a partir de *Z*, se busca en la *tabla 5.9* y se ve el p-valor para un test unidireccional:

$0.1 > P\text{-valor} > 0{,}05$

**Los resultados no son significativamente estadísticos, se acepta $H_0$ para** $\alpha \leq 0{,}05$ $(0{,}1 > P\text{-valor} > 0{,}05)$

Lo anterior dice que no es posible afirmar con un 95% de seguridad que G-Matrix paralelo sea mejor que G-Matrix lineal, por lo que se debe aceptar la hipótesis nula. Sin embargo, es preciso mencionar que los resultados obtenidos son muy alentadores ya que con un P-valor no mayor 0,1 se



Repositorio Digital Institucional "José María Rosa"
UNLa.

puede decir que existen por lo menos un 90% de probabilidades de que el método de paralelización propuesto suponga una mejoría con respecto a su contracara lineal.

### 5.8.3. Apreciaciones finales

Tanto el nuevo algoritmo G-Matrix como el método de paralelización propuestos en este Trabajo Final de Licenciatura han demostrado ser efectivas herramientas en lo que a búsqueda de (sub)isomorfismos se refiere. Particularmente se destaca el notorio aumento en rendimiento de G-Matrix por sobre Ullman, logrando en nuestros test, hasta un 58% de mejoría.

En 5.8.2.2 se observa que el tiempo que tarda Ullman para resolver un caso de sub (isomorfismo) determinado, es significativamente mayor al tiempo que tarda G-Matrix para resolver el mismo caso. Por lo que se puede decir que G-Matrix se desempeña mejor en lo que a tiempos de ejecución se refiere. Si bien se buscaba alcanzar un nivel de confianza del 95% para rechazar la hipótesis nula, el P-valor obtenido, menor a 0,0005 nos permite decir que G-Matrix supera a Ullman con más de un 99,9% de seguridad.

Por otro lado en 5.8.2.3 se tiene que el tiempo que tarda Ullman para resolver un caso de sub (isomorfismo) determinado, es significativamente mayor al tiempo que tarda Ullman paralelo para resolver el mismo caso. Por lo que se puede decir que el método de paralelización aplicado al algoritmo de Ullman, mejora el desempeño en lo que a tiempos de ejecución se refiere. Si bien se buscaba alcanzar un nivel de confianza del 95% para rechazar la hipótesis nula, el P-valor obtenido, igual a 0,01 nos permite decir que Ullman paralelo supera a Ullman con un 99% de seguridad.

Finalmente en 5.8.2.4 no se alcanzó el nivel de confianza mínimo establecido en 95% para poder afirmar que el método de paralelización aplicado al algoritmo G-Matrix mejora el desempeño del mismo. No obstante, dado que el P-valor obtenido está entre los límites de 0,1 y 0,05 nos permite formular que existen sospechas fundadas para creer que G-Matrix paralelizado es mejor que G-Matrix lineal con más de un 90% de seguridad.



Repositorio Digital Institucional "José María Rosa"
UNLa.

# 6. CONCLUSIONES

En este capítulo se presentan las conclusiones de este Trabajo Final de Licenciatura. En la sección 6.1 se describen los aportes realizados y en la sección 6.2 se señalan futuras líneas de investigación.

## 6.1. APORTES DEL TRABAJO FINAL DE LICENCIATURA

A lo largo de este trabajo se han investigado técnicas para mejorar los tiempos de ejecución en búsquedas de (sub)isomorfismo sobre grafos conexos, con lo que se ha conseguido dar respuesta a las preguntas de investigación formuladas en la sección 3.3. Tomando al algoritmo de Ullman como punto de partida, se propusieron técnicas para lograr recortar más eficientemente el árbol de búsqueda en cada iteración, así como también se han utilizado heurísticas más inteligentes que permiten dar con la solución en menor tiempo. La combinación de ambos acercamientos dio como resultado un nuevo algoritmo, al cual se lo denominó G-Matrix. A lo anterior, se suma la propuesta de una metodología para fraccionar eficazmente el árbol de búsqueda, que hace posible ejecutar la tarea sobre arquitecturas de múltiples procesadores o sobre una red de computadores si se lo prefiere, lo que se traduce en mejoras significativas sobre los tiempos de ejecución.

Para llevar a cabo las comparaciones se creó un algoritmo capaz generar pares de grafos sintéticos en función de unos parámetros de entrada preestablecidos, al mismo se lo describe en 5.2.

Para demostrar la eficiencia de G-Matrix, se lo puso a prueba contra el algoritmo de Ullman. Y para demostrar la eficiencia de nuestra técnica de división del espacio de búsqueda, se implementaron versiones paralelizadas de G-Matrix y Ullman y se las comparó contra sus respectivas versiones lineales.

Las pruebas se realizaron siguiendo el método de Montecarlo, y los resultados obtenidos tras la ejecución se cotejaron aplicando el test de rangos con signos de Wilcoxon para muestras no paramétricas.

Los resultados nos permiten afirmar que G-Matrix supera a Ullman con un nivel de confianza mayor al 99,9%. Por otro lado, nuestra propuesta de paralelización y división del espacio de búsqueda también demostró ser muy efectiva, obteniendo un nivel de confianza del 99% en Ullman vs. Ullman Paralelo, y un nivel de confianza entre [90, 95)% en G-Matrix vs. G-Matrix Paralelo.

Finalmente, las aportaciones realizadas en este Trabajo Final de Licenciatura se resumen en:



Repositorio Digital Institucional "José María Rosa"
UNLa.

I. Se presentó a G-Matrix, un nuevo algoritmo para realizar búsquedas de (sub)isomorfismo exactas sobre grafos conexos, que demostró ser superador al algoritmo de Ullman.

II. Se propuso una metodología para fraccionar el espacio de búsqueda sobre el que operan los algoritmos, permitiendo así realizar ejecuciones en paralelo, las cuales demuestran obtener significativamente mejores resultados que sus contracaras lineales.

III. Se desarrolló un algoritmo parametrizable para la generación de pares de grafos sintéticos para realizar búsquedas de (sub)isomorfismo.

## 6.2. FUTURAS LÍNEAS DE INVESTIGACIÓN

Sería interesante que en futuros esfuerzos sobre la base de este trabajo se contemple:

- Extender la funcionalidad de G-Matrix hacia grafos que tengan tanto vértices como aristas etiquetados, ya que el actual G-Matrix solo busca (sub)isomorfismo sobre la estructura topológica de los grafos.

- Investigar otras heurísticas que conduzcan la ejecución de G-Matrix por caminos que permitan recortar aún más el árbol de búsqueda para obtener mejores tiempos de ejecución.

- Estudiar en profundidad el resultado obtenido tras aplicar el método de división del espacio búsqueda, no mirando el tamaño de cada fragmento obtenido sino más bien prestando especial atención a que la carga de trabajo quede los más homogéneamente distribuida posible entre los procesadores.

- Probar el comportamiento de G-Matrix sobre grafos que incluyan relaciones del tipo no-simétricas.

- Comparar el desempeño de G-Matrix contra el algoritmo VF2 [Cordella et al, 2004], el cual goza de buena reputación entre la literatura del tema abordado en este trabajo.



Repositorio Digital Institucional "José María Rosa"
UNLa.

# 7. REFERENCIAS

Biggs, Norman (1993). Algebraic Graph Theory (2nd ed.). Cambridge University Press. ISBN 0-521-45897-8.

Charm , 2014. Programming Laboratory, Department of Computer Science University of Illinois http://charm.cs.uiuc.edu/research/masterSlave Página Vigente al 27/10/2014.

Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. International journal of pattern recognition and artificial intelligence, 18(03), 265-298.

Cook, S. A. (1971, May). The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158). ACM.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub) graph isomorphism algorithm for matching large graphs. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 26(10), 1367-1372.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms (Vol. 2, pp. 531-549). Cambridge: MIT press.

Dahm, N., Bunke, H., Caelli, T., & Gao, Y. (2012, November). Topological features and iterative node elimination for speeding up subgraph isomorphism detection. In Pattern Recognition (ICPR), 2012 21st International Conference on (pp. 1164-1167). IEEE.

De La Higuera, C., Janodet, J. C., Samuel, É., Damiand, G., & Solnon, C. (2013). Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science*, *498*, 76-99.

Ferro, A., Giugno, R., Pigola, G., Pulvirenti, A., Skripin, D., Bader, G. D., & Shasha, D. (2007). NetMatch: a Cytoscape plugin for searching biological networks. Bioinformatics, 23(7), 910-912.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Gallagher, B. (2006). Matching structure and semantics: A survey on graph-based pattern matching. AAAI FS, 6, 45-53.

Gross, J. L., & Yellen, J. (Eds.). (2003). Handbook of graph theory. CRC press.
ISO 690

Knuth, D. (1968). The Art of Computer Programming 1: Fundamental Algorithms 2: Seminumerical Algorithms 3: Sorting and Searching.

Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. In Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on (pp. 313-320). IEEE.

Lee, J., Han, W. S., Kasperovics, R., & Lee, J. H. (2012, December). An in-depth comparison of subgraph isomorphism algorithms in graph databases. In Proceedings of the VLDB Endowment (Vol. 6, No. 2, pp. 133-144). VLDB Endowment.

Luce, R. D. (1952). A note on Boolean matrix theory. Proceedings of the American Mathematical Society, 3(3), 382-388.

Metropolis, N., & Ulam, S. (1949). The monte carlo method. Journal of the American statistical association, 44(247), 335-341.

Michael, T. G., & Tamassia, R. (2002). Algorithm Design, Foundations, Analysis and Internet Examples.

Ogras, U. Y., & Marculescu, R. (2005, March). Energy-and performance-driven NoC communication architecture synthesis using a decomposition approach. In Proceedings of the conference on Design, Automation and Test in Europe-Volume 1 (pp. 352-357). IEEE Computer Society.

Ohlrich, M., Ebeling, C., Ginting, E., & Sather, L. (1993, July). SubGemini: identifying subcircuits using a fast subgraph isomorphism algorithm. In Proceedings of the 30th international Design Automation Conference (pp. 31-37). ACM.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Przulj, N., Corneil, D. G., & Jurisica, I. (2006). Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. Bioinformatics, 22(8), 974-980.

Rahman, S. A., Bashton, M., Holliday, G. L., Schrader, R., & Thornton, J. M. (2009). Journal of Cheminformatics. Journal of cheminformatics, 1, 12.

Snijders, T. A., Pattison, P. E., Robins, G. L., & Handcock, M. S. (2006). New specifications for exponential random graph models. Sociological methodology, 36(1), 99-153.

Ullman, J. R. (1976). An algorithm for subgraph isomorphism. Journal of the ACM (JACM), 23(1), 31-42.

Vassarstats, 2014. Concepts & Applications of Inferential Statistics. http://vassarstats.net/textbook/index.html. Página Vigente al 29/09/2014.

Voss, Sara, and Jaspal Subhlok. Performance of general graph isomorphism algorithms. Diss. Coe College, 2009.

Wang, H. (2010). Managing and mining graph data (Vol. 40). C. C. Aggarwal (Ed.). New York: Springer.

Wegener, Ingo (2005), Complexity Theory: Exploring the Limits of Efficient Algorithms, Springer, p. 81, ISBN 9783540210450.

Yan, X., Yu, P. S., & Han, J. (2005, June). Substructure similarity search in graph databases. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data (pp. 766-777). ACM.

Zhu, F., Qu, Q., Lo, D., Yan, X., Han, J., & Yu, P. S. (2011). Mining top-k large structural patterns in a massive network. Proceedings of the VLDB Endowment, 4(11), 807-818.



Repositorio Digital Institucional "José María Rosa"
UNLa.

Repositorio Digital Institucional "José María Rosa"
UNLa.

# VIII. ANEXO

En esta sección se presenta todo el desarrollo de programación realizado para este Trabajo Final de Licenciatura, la programación fue hecha en Java, cada sub-sección representa un paquete y cada sub-sub-sección representa una clase.

## VIII.1 ALGORITMOS

### VIII.1.1 Ullman

```
package tesis.graphs.algorithms;

public class Ullman {

	public static Matrix buildStartMatrix(Matrix A, Matrix B){

		/** Build a |VA|x|VB| M matrix*/
		Matrix M = new Matrix(A.getHeight(), B.getWidth());
		int[] sumRow = A.getRowSum();
		int[] sumColumn = B.getColumnSum();

		/** Initialize M's values so that Mij=1 <=> Grade(Ai)<= Grade(Bj)*/
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (sumColumn[j] >= sumRow[i]) {
					M.setIJ(i, j);
				}
			}
		}


		/**Begin refinement process so that ∀k(Aik = 1 => ∃p(Mkp^Bpj = 1))*/
		Matrix mFiltered = M.clone();
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (M.getIJ(i, j)) {
					for (int x = 0; x < A.getWidth(); x++) {
						if (A.getIJ(i, x)) {
							int y;
							for (y = 0; y < B.getHeight(); y++) {
								if (B.getIJ(y, j)) {
									if (M.getIJ(x, y)) {
										break;
									}
								}
							}
							if (!(y<B.getHeight())) {
								mFiltered.resetIJ(i, j);
								break;
							}
						}
					}
				}
			}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
		}
		return mFiltered;
	}

	private static Matrix buildResultMatrix(int J, int[] results) {
		Matrix Mr = new Matrix(results.length, J);
		for (int i = 0; i < results.length; i++) {
			Mr.setIJ(i, results[i]);
		}
		return Mr;
	}

	@SuppressWarnings("unused")
	private static Matrix M_MBt_deprecated(Matrix Mr, Matrix B) {
		Matrix C = B.clone();
		for (int i = 0; i < Mr.getHeight(); i++) {
			for (int j = 0; j < Mr.getWidth(); j++) {
				if(Mr.getIJ(i, j)){
					C.columnJ_To_ColumnI(j, i);
					C.rowJ_To_RowI(j, i);
				}
			}
		}
		return C;
	}

	private static Matrix M_MBt(Matrix Mr, Matrix B) {

		return Matrix.multiple(Mr, Matrix.getTransposed(Matrix.multiple(Mr, B)));

	}

	/**B > A
	 * @throws TimeOut */
	public static boolean is_SubIsomorphism(Matrix A, Matrix B){

		boolean areIsomorphic = false;
		long time_start, time_end, time_elapsed;
		Matrix M = buildStartMatrix(A, B);

		time_start = System.currentTimeMillis(); //start experiment
		/** generate all M permutations following a Deep-First-Search*/
		int[] results = new int[M.getHeight()];
		int l = -1;
		boolean goingUpWard = false;
		boolean doNext = true;

		while (doNext) {
			/*start hack*/
			if(System.currentTimeMillis() - time_start >= 60000) break;
			/*end hack*/
			if (goingUpWard) {
				l--;
				if (l != -1) {
					results[l]++;
					boolean hasNext = false;
					while (results[l] < M.getWidth()) {
						if (M.getIJ(l, results[l])) {
							int i = 0;
							while (i < l) {
								if (results[l] == results[i]) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                                        break;
                                    }
                                    i++;
                                }
                                if (i == l) {
                                    hasNext = true;
                                    break;
                                }
                            }
                            results[l]++;
                        }
                        if (hasNext) {
                            goingUpWard = false;
                        } else {
                            goingUpWard = true;

                        }
                    } else {
                        doNext = false;
                    }

                } else {
                    l++;
                    if (l != results.length) {
                        results[l] = 0;
                        boolean hasNext = false;
                        while (results[l] < M.getWidth()) {
                            if (M.getIJ(l, results[l])) {
                                int i = 0;
                                while (i < l) {
                                    if (results[l] == results[i]) {
                                        break;
                                    }
                                    i++;
                                }
                                if (i == l) {
                                    hasNext = true;
                                    break;
                                }
                            }
                            results[l]++;
                        }
                        if (hasNext) {
                            goingUpWard = false;
                        } else {
                            goingUpWard = true;
                        }
                    } else {
                        /** Block: Once M has been chose we begin the isomorphism
checking test*/
                        Matrix Mr = buildResultMatrix(M.getWidth(), results);
                        Matrix C = M_MBt(Mr, B);
                        if(Matrix.contains(A, C)){
                            areIsomorphic=true;
                            break;
                        }
                        /** End block*/

                        goingUpWard = true;

                        /** control time*/
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
							time_end = System.currentTimeMillis();
							time_elapsed = time_end -time_start;
							if(time_elapsed >= 60000){
								break;
							}
							/** END control*/
						}
					}
				}

				return areIsomorphic;
		}
}
```

## VIII.1.2. Ullman Paralelo

```
package tesis.graphs.algorithms;

import java.util.ArrayList;

import tesis.graphs.tests.MySignal;

public class UllmanThreads implements Runnable {

		private boolean foundSubIsomorphism;
		private ArrayList<Thread> ALT;
		private Matrix Mdevide,A,B;
		private MySignal mySignal;

		public UllmanThreads(Matrix Mdevide,Matrix A,Matrix B, MySignal mySignal,ArrayList<Thread>
ALT){
				this.Mdevide = Mdevide;
				this.A = A;
				this.B =B;
				this.foundSubIsomorphism = false;
				this.mySignal = mySignal;
				this.ALT = ALT;
		}

		public boolean getFoundIsomorphism(){
				return this.foundSubIsomorphism;
		}

		public static Matrix buildStartMatrix(Matrix A, Matrix B){

				/** Build a |VA|x|VB| M matrix*/
				Matrix M = new Matrix(A.getHeight(), B.getWidth());
				int[] sumRow = A.getRowSum();
				int[] sumColumn = B.getColumnSum();

				/** Initialize M's values so that Mij=1 <=> Grade(Ai)<= Grade(Bj)*/
				for (int i = 0; i < M.getHeight(); i++) {
						for (int j = 0; j < M.getWidth(); j++) {
								if (sumColumn[j] >= sumRow[i]) {
										M.setIJ(i, j);
								}
						}
				}
```



Repositorio Digital Institucional "José María Rosa" UNLa.

```
		/**Begin refinement process so that ∀k(Aik = 1 => ∃p(Mkp^Bpj = 1))*/
		Matrix mFiltered = M.clone();
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (M.getIJ(i, j)) {
					for (int x = 0; x < A.getWidth(); x++) {
						if (A.getIJ(i, x)) {
							int y;
							for (y = 0; y < B.getHeight(); y++) {
								if (B.getIJ(y, j)) {
									if (M.getIJ(x, y)) {
										break;
									}
								}
							}
							if (!(y<B.getHeight())) {
								mFiltered.resetIJ(i, j);
								break;
							}
						}
					}
				}
			}
		}
		return mFiltered;
	}

	/**divide work load of the matrix better for threading**/
	public static ArrayList<Matrix> getAPartOfM(Matrix MFiltered){

		/**initialize variables **/
		int CoresCount = Runtime.getRuntime().availableProcessors();
		ArrayList<Matrix> MatrixList = new ArrayList<Matrix>();
		for(int i = 0; i<CoresCount;i++){
			MatrixList.add(MFiltered.clone());
		}
		int selectedRow = 0;
		boolean foundedRow = false;
		int[] sumRows = MFiltered.getRowSum();
		/** **/

		/**Selects the row to be used **/
		for (int i = 0; i < sumRows.length;i++){
			if (sumRows[i]%CoresCount==0){
				selectedRow=i;
				foundedRow=true;
				break;
			}
		}
		if(!foundedRow){
			int maxvalue = 0;
			for (int i = 0; i<sumRows.length;i++){
				if(sumRows[i]>maxvalue){
					maxvalue = sumRows[i];
					selectedRow=i;
				}
			}
		}
		/** **/
		int onesToLeave = (int) sumRows[selectedRow] / CoresCount;
		int rest = sumRows[selectedRow]%CoresCount;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
        int[] x = new int[CoresCount];
        for(int i = 0; i< x.length;i++){
            if(rest > 0){
                rest--;
                x[i]= onesToLeave +1;
            }else{
                x[i]= onesToLeave;
            }
        }

        int shift = 0;
        for(int i = 0; i< MatrixList.size();i++){
            int counter = shift;
            Matrix M = MatrixList.get(i);

            for(int j=0; j< M.getWidth();j++){
                if(counter>0 && M.getIJ(selectedRow,j)){
                    M.resetIJ(selectedRow, j);
                    counter--;
                }
            }
            shift+=x[i];
            for(int j=0; j< M.getWidth();j++){
                if(M.getIJ(selectedRow,j)){
                    if(x[i]>0){
                        x[i]--;
                    }else{
                        M.resetIJ(selectedRow, j);
                    }
                }
            }
        }
        return MatrixList;
    }

    private static Matrix buildResultMatrix(int J, int[] results) {
        Matrix Mr = new Matrix(results.length, J);
        for (int i = 0; i < results.length; i++) {
            Mr.setIJ(i, results[i]);
        }
        return Mr;
    }

    @SuppressWarnings("unused")
    private static Matrix M_MBt_deprecated(Matrix Mr, Matrix B) {
        Matrix C = B.clone();
        for (int i = 0; i < Mr.getHeight(); i++) {
            for (int j = 0; j < Mr.getWidth(); j++) {
                if(Mr.getIJ(i, j)){
                    C.columnJ_To_ColumnI(j, i);
                    C.rowJ_To_RowI(j, i);
                }
            }
        }
        return C;
    }

    private static Matrix M_MBt(Matrix Mr, Matrix B) {

        return Matrix.multiple(Mr, Matrix.getTransposed(Matrix.multiple(Mr, B)));
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
}

/**B > A
 * @throws TimeOut */
public boolean is_SubIsomorphism(Matrix M,Matrix A, Matrix B) {

        boolean areIsomorphic = false;

        /** generate all M permutations following a Deep-First-Search*/

        int[] results = new int[M.getHeight()];
        int l = -1;
        boolean goingUpWard = false;
        boolean doNext = true;
        long timeEnd;
        long timeStart = System.currentTimeMillis();
        while (doNext) {
                //START hack
                if(mySignal.isFinished() || Thread.currentThread().isInterrupted()){break;}
                //END hack
                if (goingUpWard) {
                        l--;
                        if (l != -1) {
                                results[l]++;
                                boolean hasNext = false;
                                while (results[l] < M.getWidth()) {
                                        if (M.getIJ(l, results[l])) {
                                                int i = 0;
                                                while (i < l) {
                                                        if (results[l] == results[i]) {
                                                                break;
                                                        }
                                                        i++;
                                                }
                                                if (i == l) {
                                                        hasNext = true;
                                                        break;
                                                }
                                        }
                                        results[l]++;
                                }
                                if (hasNext) {
                                        goingUpWard = false;
                                } else {
                                        goingUpWard = true;

                                }
                        } else {
                                doNext = false;
                        }

                } else {
                        l++;
                        if (l != results.length) {
                                results[l] = 0;
                                boolean hasNext = false;
                                while (results[l] < M.getWidth()) {
                                        if (M.getIJ(l, results[l])) {
                                                int i = 0;
                                                while (i < l) {
                                                        if (results[l] == results[i]) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                                                break;
                                        }
                                        i++;
                                }
                                if (i == l) {
                                        hasNext = true;
                                        break;
                                }
                        }
                        results[l]++;
                }
                if (hasNext) {
                        goingUpWard = false;
                } else {
                        goingUpWard = true;
                }
        } else {
                /** Block: Once M has been chose we begin the isomorphism
checking test*/

                Matrix Mr = buildResultMatrix(M.getWidth(), results);
                Matrix C = M_MBt(Mr, B);
                if(Matrix.contains(A, C)){
                        areIsomorphic=true;
                        break;
                }
                /** End block*/

                goingUpWard = true;
        }
    }
    /** control time*/
    timeEnd = System.currentTimeMillis();
    if(timeEnd - timeStart > 60000){
        Thread.currentThread().interrupt();
    }
    /** END control*/
  }
  return areIsomorphic;
}

@Override
public void run() {
    foundSubIsomorphism = is_SubIsomorphism(Mdevide,A, B);
    if(foundSubIsomorphism){
        mySignal.setTimeEnd(System.currentTimeMillis());
        mySignal.setFinished();
        for(Thread th : ALT){
            if(th.isAlive())
            th.interrupt();
        }
    }
}
}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

# VIII.1.3 G-Matrix

```
package tesis.graphs.algorithms;

import java.util.ArrayList;
import java.util.LinkedList;

public class GMatrix {
	private static Matrix buildStartMatrix(Matrix A, Matrix B) {

		/** Build a |VA|x|VB| M matrix */
		Matrix M = new Matrix(A.getHeight(), B.getWidth());
		int[] sumRow = A.getRowSum();
		int[] sumColumn = B.getColumnSum();

		/** Initialize M's values so that Mij=1 <=> Grade(Ai)<= Grade(Bj) */
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (sumColumn[j] >= sumRow[i]) {
					M.setIJ(i, j);
				}
			}
		}

		/** Begin refinement process so that ∀k(Aik = 1 => ∃p(Mkp^Bpj = 1)) */
		Matrix mFiltered = M.clone();
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (M.getIJ(i, j)) {
					for (int x = 0; x < A.getWidth(); x++) {
						if (A.getIJ(i, x)) {
							int y;
							for (y = 0; y < B.getHeight(); y++) {
								if (B.getIJ(y, j)) {
									if (M.getIJ(x, y)) {
										break;
									}
								}
							}
							if (!(y < B.getHeight())) {
								mFiltered.resetIJ(i, j);
								break;
							}
						}
					}
				}
			}
		}
		return mFiltered;
	}

	/** B > A */
	public static boolean is_SubIsomorphism(Matrix A, Matrix B) {

		Matrix M = GMatrix.buildStartMatrix(A, B);
		M = Matrix.getTransposed(M);
		B = Matrix.getTransposed(B);
		boolean foundIsomorphism = false;
		boolean[] mUsableRow = new boolean[M.getHeight()];
		boolean[] mUsableColumn = new boolean[M.getWidth()];
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
for (int i = 0; i < mUsableRow.length; i++) {
        mUsableRow[i] = true;
}
for (int i = 0; i < mUsableColumn.length; i++) {
        mUsableColumn[i] = true;
}

// get A into List format
ArrayList<int[]> aList = new ArrayList<int[]>();
for (int i = 0; i < A.getHeight(); i++) {
        for (int j = 0; j < A.getWidth(); j++) {
                if (A.getIJ(i, j)) {
                        int[] ij = new int[2];
                        ij[0] = i;
                        ij[1] = j;
                        aList.add(ij);
                }
        }
}

// order aList execution into aListSorted
// get minimum column in jMin
int[] McolumnSum = M.getColumnSum();
int min = McolumnSum[0];
int jMin = 0;
for (int i = 0; i < McolumnSum.length; i++) {
        if (McolumnSum[i] < min) {
                min = McolumnSum[i];
                jMin = i;
        }
}

// start sorting
ArrayList<int[]> aListSorted = new ArrayList<int[]>();
boolean flag = false;
while (aList.size() > 0) {
        for (int[] ij : aList) {
                if (ij[1] == jMin) {
                        aListSorted.add(ij);
                        aList.remove(ij);
                        jMin = ij[0];
                        flag = true;
                        break;
                }
        }
        if (flag) {
                flag = false;
        } else {
                flag = false;
                jMin++;
        }
}
// END
// Start main loop
long timeEnd;
long timeStart = System.currentTimeMillis();
LinkedList<ExecutionResource> LLER = new LinkedList<ExecutionResource>();
int aIndex = 0;
boolean goingDown = true;
while (aIndex != -1) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                if (aIndex < aListSorted.size() - 1) {
                    if (goingDown) {
                        LLER.add(saveInER(M, aListSorted.get(aIndex), mUsableRow,
                                    mUsableColumn));
                    } else {
                        restoreFromER(LLER.getLast(), M, mUsableRow, mUsableColumn,
                                    aListSorted.get(aIndex));
                    }

                    if (foundBranch(LLER.getLast(), M, B, mUsableRow,
                                mUsableColumn, aListSorted.get(aIndex))) {
                        aIndex++;
                        goingDown = true;
                    } else {
                        aIndex--;
                        LLER.removeLast();
                        goingDown = false;
                    }
                } else {
                    LLER.add(saveInER(M, aListSorted.get(aIndex), mUsableRow,
                                mUsableColumn));
                    if (foundBranch(LLER.getLast(), M, B, mUsableRow,
                                mUsableColumn, aListSorted.get(aIndex))) {
                        aIndex = -1;
                        foundIsomorphism = true;
                    } else {
                        aIndex--;
                        LLER.removeLast();
                        goingDown = false;
                    }
                }
                timeEnd = System.currentTimeMillis();
                if (timeEnd - timeStart > 60000) {
                    aIndex = -1;
                }
            }
            // END main-loop
            return foundIsomorphism;
        }

        private static ExecutionResource saveInER(Matrix M, int[] ij,
                    boolean[] mUsableRow, boolean[] mUsableColumn) {

            boolean[] mColumnFirst = new boolean[M.getHeight()];
            boolean[] mColumnSecond = new boolean[M.getHeight()];
            boolean[] mUsableRowCopy = new boolean[mUsableRow.length];
            boolean[] mUsableColumnCopy = new boolean[mUsableColumn.length];

            for (int i = 0; i < M.getHeight(); i++) {
                mColumnFirst[i] = M.getIJ(i, ij[1]);
                mColumnSecond[i] = M.getIJ(i, ij[0]);
                mUsableRowCopy[i] = mUsableRow[i];
            }
            for (int i = 0; i < mUsableColumn.length; i++) {
                mUsableColumnCopy[i] = mUsableColumn[i];
            }

            ExecutionResource ER = new ExecutionResource(mColumnFirst,
                        mColumnSecond, mUsableRowCopy, mUsableColumnCopy);

            return ER;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
	}

	private static void restoreFromER(ExecutionResource ER, Matrix M,
			boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {
		for (int i = 0; i < M.getHeight(); i++) {
			if (ER.getmColumnFirst()[i]) {
				M.setIJ(i, ij[1]);
			} else {
				M.resetIJ(i, ij[1]);
			}

			if (ER.getmColumnSecond()[i]) {
				M.setIJ(i, ij[0]);
			} else {
				M.resetIJ(i, ij[0]);
			}
			mUsableRow[i] = ER.getmUsableRow()[i];
		}
		for (int i = 0; i < mUsableColumn.length; i++) {
			mUsableColumn[i] = ER.getmUsableColumn()[i];
		}

	}

	private static void reduce(Matrix M, Matrix B, int executionIndex,
			boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {

		M.setColumn(executionIndex, ij[1]);
		mUsableColumn[ij[1]] = false;
		mUsableRow[executionIndex] = false;

		if (mUsableColumn[ij[0]]) {
			for (int i = 0; i < M.getHeight(); i++) {
				if (M.getIJ(i, ij[0]) && mUsableRow[i]
						&& B.getIJ(i, executionIndex)) {
					M.setIJ(i, ij[0]);
				} else {
					M.resetIJ(i, ij[0]);
				}
			}
		} else {
			for (int i = 0; i < M.getHeight(); i++) {
				if (M.getIJ(i, ij[0]) && B.getIJ(i, executionIndex)) {
					M.setIJ(i, ij[0]);
				} else {
					M.resetIJ(i, ij[0]);
				}
			}
		}
	}

	private static boolean foundBranch(ExecutionResource ER, Matrix M,
			Matrix B, boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {

		boolean foundBranch = false;

		for (int i = ER.getExecutionIndex(); i < M.getHeight(); i++) {
			if (M.getIJ(i, ij[1])) {
				reduce(M, B, i, mUsableRow, mUsableColumn, ij);
				for (int n = 0; n < M.getHeight(); n++) {
					if (M.getIJ(n, ij[0])) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                                        foundBranch = true;
                                        break;
                                    }
                                }
                                if (foundBranch) {

                                    ER.setExecutionIndex(i + 1);
                                    break;
                                } else {
                                    restoreFromER(ER, M, mUsableRow, mUsableColumn, ij);

                                }
                            }
                        }

                        return foundBranch;
                    }
}
```

## VIII.1.4. G-Matrix Paralelo

```
package tesis.graphs.algorithms;

import java.util.ArrayList;
import java.util.LinkedList;

import tesis.graphs.tests.MySignal;

public class GMatrixThreads implements Runnable {

        private boolean foundSubIsomorphism;
        private ArrayList<Thread> ALThre;
        private Matrix Mdevide, A, B;
        private MySignal mySignal;

        public GMatrixThreads(Matrix Mdevide, Matrix A, Matrix B,
                        MySignal mySignal, ArrayList<Thread> ALThre) {
                this.Mdevide = Mdevide;
                this.A = A;
                this.B = B;
                this.foundSubIsomorphism = false;
                this.mySignal = mySignal;
                this.ALThre = ALThre;
        }

        public boolean getFoundIsomorphism() {
                return this.foundSubIsomorphism;
        }

        public static Matrix buildStartMatrix(Matrix A, Matrix B) {

                /** Build a |VA|x|VB| M matrix */
                Matrix M = new Matrix(A.getHeight(), B.getWidth());
                int[] sumRow = A.getRowSum();
                int[] sumColumn = B.getColumnSum();

                /** Initialize M's values so that Mij=1 <=> Grade(Ai)<= Grade(Bj) */
                for (int i = 0; i < M.getHeight(); i++) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				for (int j = 0; j < M.getWidth(); j++) {
					if (sumColumn[j] >= sumRow[i]) {
						M.setIJ(i, j);
					}
				}
			}



			/** Begin refinement process so that ∀k(Aik = 1 => ∃p(Mkp^Bpj = 1)) */
			Matrix mFiltered = M.clone();
			for (int i = 0; i < M.getHeight(); i++) {
				for (int j = 0; j < M.getWidth(); j++) {
					if (M.getIJ(i, j)) {
						for (int x = 0; x < A.getWidth(); x++) {
							if (A.getIJ(i, x)) {
								int y;
								for (y = 0; y < B.getHeight(); y++) {
									if (B.getIJ(y, j)) {
										if (M.getIJ(x, y)) {
											break;
										}
									}
								}
								if (!(y < B.getHeight())) {
									mFiltered.resetIJ(i, j);
									break;
								}
							}
						}
					}
				}
			}
			return mFiltered;
		}

		/** divide work load of the matrix better for threading **/
		public static ArrayList<Matrix> getAPartOfM(Matrix MFiltered) {

			/** initialize variables **/
			int CoresCount = Runtime.getRuntime().availableProcessors();
			ArrayList<Matrix> MatrixList = new ArrayList<Matrix>();
			for (int i = 0; i < CoresCount; i++) {
				MatrixList.add(MFiltered.clone());
			}
			int selectedRow = 0;
			boolean foundedRow = false;
			int[] sumRows = MFiltered.getRowSum();
			/** **/

			/** Selects the row to be used **/
			for (int i = 0; i < sumRows.length; i++) {
				if (sumRows[i] % CoresCount == 0) {
					selectedRow = i;
					foundedRow = true;
					break;
				}
			}
			if (!foundedRow) {
				int maxvalue = 0;
				for (int i = 0; i < sumRows.length; i++) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                    if (sumRows[i] > maxvalue) {
                        maxvalue = sumRows[i];
                        selectedRow = i;
                    }
                }
            }
            /** **/
            int onesToLeave = (int) sumRows[selectedRow] / CoresCount;
            int rest = sumRows[selectedRow] % CoresCount;
            int[] x = new int[CoresCount];
            for (int i = 0; i < x.length; i++) {
                if (rest > 0) {
                    rest--;
                    x[i] = onesToLeave + 1;
                } else {
                    x[i] = onesToLeave;
                }
            }

            int shift = 0;
            for (int i = 0; i < MatrixList.size(); i++) {
                int counter = shift;
                Matrix M = MatrixList.get(i);

                for (int j = 0; j < M.getWidth(); j++) {
                    if (counter > 0 && M.getIJ(selectedRow, j)) {
                        M.resetIJ(selectedRow, j);
                        counter--;
                    }
                }
                shift += x[i];
                for (int j = 0; j < M.getWidth(); j++) {
                    if (M.getIJ(selectedRow, j)) {
                        if (x[i] > 0) {
                            x[i]--;
                        } else {
                            M.resetIJ(selectedRow, j);
                        }
                    }
                }
            }
            return MatrixList;
        }

        /**
         * B > A
         *
         * @throws TimeOut
         */
        public boolean is_SubIsomorphism(Matrix M, Matrix A, Matrix B) {

            M = Matrix.getTransposed(M);
            B = Matrix.getTransposed(B);

            boolean foundIsomorphism = false;
            boolean[] mUsableRow = new boolean[M.getHeight()];
            boolean[] mUsableColumn = new boolean[M.getWidth()];
            for (int i = 0; i < mUsableRow.length; i++) {
                mUsableRow[i] = true;
            }
            for (int i = 0; i < mUsableColumn.length; i++) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
			mUsableColumn[i] = true;
		}

		// get A into List format
		ArrayList<int[]> aList = new ArrayList<int[]>();
		for (int i = 0; i < A.getHeight(); i++) {
			for (int j = 0; j < A.getWidth(); j++) {
				if (A.getIJ(i, j)) {
					int[] ij = new int[2];
					ij[0] = i;
					ij[1] = j;
					aList.add(ij);
				}
			}
		}

		// order aList execution into aListSorted
		// get minimum column in jMin
		int[] McolumnSum = M.getColumnSum();
		int min = McolumnSum[0];
		int jMin = 0;
		for (int i = 0; i < McolumnSum.length; i++) {
			if (McolumnSum[i] < min) {
				min = McolumnSum[i];
				jMin = i;
			}
		}

		// start sorting
		ArrayList<int[]> aListSorted = new ArrayList<int[]>();
		boolean flag = false;
		while (aList.size() > 0) {
			for (int[] ij : aList) {
				if (ij[1] == jMin) {
					aListSorted.add(ij);
					aList.remove(ij);
					jMin = ij[0];
					flag = true;
					break;
				}
			}
			if (flag) {
				flag = false;
			} else {
				flag = false;
				jMin++;
			}
		}
		// END

		// Start main loop
		long timeEnd;
		long timeStart = System.currentTimeMillis();
		LinkedList<ExecutionResource> LLER = new LinkedList<ExecutionResource>();
		int aIndex = 0;
		boolean goingDown = true;
		while (aIndex != -1) {
			// Ugly hack
			if (mySignal.isFinished() || Thread.currentThread().isInterrupted()) {
				break;
			}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                // END Ugly hack

                if (aIndex < aListSorted.size() - 1) {
                        if (goingDown) {
                                LLER.add(saveInER(M, aListSorted.get(aIndex), mUsableRow,
                                                mUsableColumn));
                        } else {
                                restoreFromER(LLER.getLast(), M, mUsableRow, mUsableColumn,
                                                aListSorted.get(aIndex));
                        }

                        if (foundBranch(LLER.getLast(), M, B, mUsableRow,
                                        mUsableColumn, aListSorted.get(aIndex))) {
                                aIndex++;
                                goingDown = true;
                        } else {
                                aIndex--;
                                LLER.removeLast();
                                goingDown = false;
                        }

                } else {
                        LLER.add(saveInER(M, aListSorted.get(aIndex), mUsableRow,
                                        mUsableColumn));
                        if (foundBranch(LLER.getLast(), M, B, mUsableRow,
                                        mUsableColumn, aListSorted.get(aIndex))) {
                                aIndex = -1;
                                foundIsomorphism = true;
                        } else {
                                aIndex--;
                                LLER.removeLast();
                                goingDown = false;
                        }
                }
                timeEnd = System.currentTimeMillis();
                if (timeEnd - timeStart > 60000) {
                        aIndex = -1;
                }

        }
        // END main-loop

        return foundIsomorphism;
}

private static ExecutionResource saveInER(Matrix M, int[] ij,
                boolean[] mUsableRow, boolean[] mUsableColumn) {

        boolean[] mColumnFirst = new boolean[M.getHeight()];
        boolean[] mColumnSecond = new boolean[M.getHeight()];
        boolean[] mUsableRowCopy = new boolean[mUsableRow.length];
        boolean[] mUsableColumnCopy = new boolean[mUsableColumn.length];

        for (int i = 0; i < M.getHeight(); i++) {
                mColumnFirst[i] = M.getIJ(i, ij[1]);
                mColumnSecond[i] = M.getIJ(i, ij[0]);
                mUsableRowCopy[i] = mUsableRow[i];
        }
        for (int i = 0; i < mUsableColumn.length; i++) {
                mUsableColumnCopy[i] = mUsableColumn[i];
        }
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
        ExecutionResource ER = new ExecutionResource(mColumnFirst,
                mColumnSecond, mUsableRowCopy, mUsableColumnCopy);

        return ER;
    }

    private static void restoreFromER(ExecutionResource ER, Matrix M,
            boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {
        for (int i = 0; i < M.getHeight(); i++) {
            if (ER.getmColumnFirst()[i]) {
                M.setIJ(i, ij[1]);
            } else {
                M.resetIJ(i, ij[1]);
            }

            if (ER.getmColumnSecond()[i]) {
                M.setIJ(i, ij[0]);
            } else {
                M.resetIJ(i, ij[0]);
            }
            mUsableRow[i] = ER.getmUsableRow()[i];
        }
        for (int i = 0; i < mUsableColumn.length; i++) {
            mUsableColumn[i] = ER.getmUsableColumn()[i];
        }

    }

    private static void reduce(Matrix M, Matrix B, int executionIndex,
            boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {

        M.setColumn(executionIndex, ij[1]);
        mUsableColumn[ij[1]] = false;
        mUsableRow[executionIndex] = false;

        if (mUsableColumn[ij[0]]) {
            for (int i = 0; i < M.getHeight(); i++) {
                if (M.getIJ(i, ij[0]) && mUsableRow[i]
                        && B.getIJ(i, executionIndex)) {
                    M.setIJ(i, ij[0]);
                } else {
                    M.resetIJ(i, ij[0]);
                }
            }
        } else {
            for (int i = 0; i < M.getHeight(); i++) {
                if (M.getIJ(i, ij[0]) && B.getIJ(i, executionIndex)) {
                    M.setIJ(i, ij[0]);
                } else {
                    M.resetIJ(i, ij[0]);
                }
            }
        }
    }

    private static boolean foundBranch(ExecutionResource ER, Matrix M,
            Matrix B, boolean[] mUsableRow, boolean[] mUsableColumn, int[] ij) {

        boolean foundBranch = false;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
        for (int i = ER.getExecutionIndex(); i < M.getHeight(); i++) {
            if (M.getIJ(i, ij[1])) {

                reduce(M, B, i, mUsableRow, mUsableColumn, ij);

                for (int n = 0; n < M.getHeight(); n++) {
                    if (M.getIJ(n, ij[0])) {
                        foundBranch = true;
                        break;
                    }
                }
                if (foundBranch) {

                    ER.setExecutionIndex(i + 1);
                    break;
                } else {

                    restoreFromER(ER, M, mUsableRow, mUsableColumn, ij);

                }
            }
        }

        return foundBranch;
    }

    @Override
    public void run() {
        foundSubIsomorphism = is_SubIsomorphism(Mdevide, A, B);
        if (foundSubIsomorphism) {
            mySignal.setTimeEnd(System.currentTimeMillis());
            mySignal.setFinished();
            for (Thread th : ALThre) {
                if (th.isAlive())
                    th.interrupt();
            }
        }
    }
}
```

## VIII.1.5. Operaciones sobre Matrices

```
package tesis.graphs.algorithms;
public class Matrix implements Cloneable {

    public static void consoleOut(Matrix M, String heading) {
        System.out.println("__________BEGIN: " + heading + "__________");
        for (int i = 0; i < M.getHeight(); i++) {
            for (int j = 0; j < M.getWidth(); j++) {
                System.out.print(M.getIntIJ(i, j) + " ");
            }
            System.out.println("\n");
        }
        System.out.println("______________END_______________");
    }

    public static boolean contains(Matrix A, Matrix C) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
		boolean flag = true;
		for (int i = 0; i < A.getHeight(); i++) {
			for (int j = 0; j < A.getWidth(); j++) {
				if (A.getIJ(i, j)) {
					if (!C.getIJ(i, j)) {
						flag = false;
						break;
					}
				}
			}
			if (!flag)
				break;
		}
		return flag;
	}

	public static Matrix getTransposed(Matrix M) {
		Matrix T = new Matrix(M.getWidth(), M.getHeight());
		for (int i = 0; i < M.getHeight(); i++) {
			for (int j = 0; j < M.getWidth(); j++) {
				if (M.getIJ(i, j)) {
					T.setIJ(j, i);
				}
			}
		}
		return T;
	}

	public static Matrix multiple(Matrix A, Matrix B) {
		if (A.getWidth() == B.getHeight()) {
			Matrix C = new Matrix(A.getHeight(), B.getWidth());
			int m = A.getWidth();
			boolean flag = false;
			for (int i = 0; i < C.getHeight(); i++) {
				for (int j = 0; j < C.getWidth(); j++) {
					for (int k = 0; k < m; k++) {
						flag ^= (A.getIJ(i, k) && B.getIJ(k, j));
					}
					if (flag) {
						C.setIJ(i, j);
						flag = false;
					}
				}
			}
			return C;
		} else {
			System.out.println("Those matrix can't be multipled");
			return null;
		}
	}

	private boolean[][] matrix;
	private char[] columnLabels;
	private char[] rowLabels;
	private int height, width;

	public Matrix(int height, int width) {
		this.matrix = new boolean[height][width];
		setColumnLabels(columnLabels);
		setRowLabels(rowLabels);
		this.height = height;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
			this.width = width;
		}

		public Matrix clone() {
			Matrix M = new Matrix(getHeight(), getWidth());
			for (int i = 0; i < getHeight(); i++) {
				for (int j = 0; j < getWidth(); j++) {
					if (matrix[i][j]) {
						M.setIJ(i, j);
					}
				}
			}
			return M;
		}

		public void columnJ_To_ColumnI(int j, int i) {
			boolean aux;
			for (int r = 0; r < getHeight(); r++) {
				aux = matrix[r][i];
				matrix[r][i] = matrix[r][j];
				matrix[r][j] = aux;
			}
		}

		public boolean equals(Matrix M) {
			boolean flag = true;
			if (M.getHeight() != getHeight() || M.getWidth() != getWidth()) {
				flag = false;
			} else {
				for (int i = 0; i < getHeight(); i++) {
					for (int j = 0; j < getWidth(); j++) {
						if (M.getIJ(i, j) != matrix[i][j]) {
							flag = false;
						}
					}
				}
			}
			return flag;
		}

		public int[] getColumnSum() {
			int[] columnSum = new int[getWidth()];
			for (int j = 0; j < getWidth(); j++) {
				for (int i = 0; i < getHeight(); i++) {
					columnSum[j] += getIntIJ(i, j);
				}
			}
			return columnSum;
		}

		public int[] getRowSum() {
			int[] RowSum = new int[getHeight()];
			for (int i = 0; i < getHeight(); i++) {
				for (int j = 0; j < getWidth(); j++) {
					RowSum[i] += getIntIJ(i, j);
				}
			}
			return RowSum;
		}

		public int getHeight() {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
			return height;
		}

		public boolean getIJ(int i, int j) {
			return matrix[i][j];
		}

		public int getIntIJ(int i, int j) {
			return matrix[i][j] ? 1 : 0;
		}

		public int getWidth() {
			return width;
		}

		public void resetIJ(int i, int j) {
			matrix[i][j] = false;
		}

		public void rowJ_To_RowI(int j, int i) {
			boolean aux;
			for (int c = 0; c < getWidth(); c++) {
				aux = matrix[i][c];
				matrix[i][c] = matrix[j][c];
				matrix[j][c] = aux;
			}
		}

		public void setColumnLabels(char[] columnLabels) {
			// TODO setCOlumnLabels
			this.columnLabels = columnLabels;
		}

		public void setIJ(int i, int j) {
			matrix[i][j] = true;
		}

		public void setRowLabels(char[] rowLabels) {
			// TODO setRowLabels
			this.rowLabels = rowLabels;
		}

		public void setColumn(int I, int J){

			for(int i = 0; i<height;i++){
				matrix[i][J] = false;
			}
			matrix[I][J]=true;
		}
}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

## VIII.1.6. Recursos de Ejecución

```
package tesis.graphs.algorithms;

public class ExecutionResource {
	private int executionIndex;
	private boolean[] mColumnFirst;
	private boolean[] mColumnSecond;
	private boolean[] mUsableRow;
	private boolean[] mUsableColumn;


	public ExecutionResource(boolean[] mColumnFirst,
				boolean[] mColumnSecond, boolean[] mUsableRow,
				boolean[] mUsableColumn) {
		super();
		this.executionIndex = 0;
		this.mColumnFirst = mColumnFirst;
		this.mColumnSecond = mColumnSecond;
		this.mUsableRow = mUsableRow;
		this.mUsableColumn = mUsableColumn;
	}

	public int getExecutionIndex() {
		return executionIndex;
	}

	public boolean[] getmColumnFirst() {
		return mColumnFirst;
	}

	public boolean[] getmColumnSecond() {
		return mColumnSecond;
	}

	public boolean[] getmUsableRow() {
		return mUsableRow;
	}

	public boolean[] getmUsableColumn() {
		return mUsableColumn;
	}

	public void setExecutionIndex(int executionIndex) {
		this.executionIndex = executionIndex;
	}

	public void setmColumnFirst(boolean[] mColumnFirst) {
		this.mColumnFirst = mColumnFirst;
	}

	public void setmColumnSecond(boolean[] mColumnSecond) {
		this.mColumnSecond = mColumnSecond;
	}

	public void setmUsableRow(boolean[] mUsableRow) {
		this.mUsableRow = mUsableRow;
	}

	public void setmUsableColumn(boolean[] mUsableColumn) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
			this.mUsableColumn = mUsableColumn;
		}

}
```

# VIII.2. CREACION DE LOS SET DE DATOS

## VIII.2.1. Escribir resultados a disco

```
package tesis.graphs.model;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

public class FilePrinter {

	private static FilePrinter instance;
	private BufferedWriter out;

	private FilePrinter() {
		try {

			out = new BufferedWriter(new OutputStreamWriter(
					new FileOutputStream("/home/geronimo/Escritorio/Tests"),
					"utf-8"));

		} catch (FileNotFoundException e) {

			e.printStackTrace();
		} catch (UnsupportedEncodingException e) {

			e.printStackTrace();
		}
	}
	public static FilePrinter getInstance(){
		if(instance == null){
			instance = new FilePrinter();
		}
		return instance;
	}

	public void write(String text) {
		try {
			out.write(text);
			out.flush();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void close() {
		try {
			out.close();
		} catch (IOException e) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				e.printStackTrace();
			}
		}

}
```

## VIII.2.2. Excepción por definición de parámetros invalida

```
package tesis.graphs.model;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;

public class FilePrinter {

	private static FilePrinter instance;
	private BufferedWriter out;

	private FilePrinter() {
		try {

			out = new BufferedWriter(new OutputStreamWriter(
					new FileOutputStream("/home/geronimo/Escritorio/Tests"),
					"utf-8"));

		} catch (FileNotFoundException e) {

			e.printStackTrace();
		} catch (UnsupportedEncodingException e) {

			e.printStackTrace();
		}
	}
	public static FilePrinter getInstance(){
		if(instance == null){
			instance = new FilePrinter();
		}
		return instance;
	}

	public void write(String text) {
		try {
			out.write(text);
			out.flush();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public void close() {
		try {
			out.close();
		} catch (IOException e) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                        e.printStackTrace();
                }
        }

}
```

## VIII.2.3. Generador de etiquetas

```
package tesis.graphs.model;

public class LabelsCreator {

        private char[] labels;

        public LabelsCreator(int labelsNumber){
                labels = new char[labelsNumber];
                for(int i=0;i<labelsNumber; i++){
                        labels[i] = (char)(65+i);
                }
        }
        public char getRandomLabel(){
                return labels[RandomNumbers.getSimpleRamdom(0, labels.length-1)];
        }
}
```

## VIII.2.4. Grafo mayor (Contenedor)

```
package tesis.graphs.model;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;

import tesis.graphs.algorithms.Matrix;

public class LargeGraph {

        private ArrayList<LinkedList<Vertex>> largeGraph;
        private int vertexes;
        private int edges;
        private SmallGraph smallGraph;

        public LargeGraph(LabelsCreator LC, SmallGraph SG, int vertexes, int edges)
                        throws InvalidVertexesEdgesDefinition {
                this.largeGraph = new ArrayList<LinkedList<Vertex>>();
                this.vertexes = vertexes;
                this.edges = edges;
                this.smallGraph = SG;
                copySmallGraph_to_LargeGraph(SG);
                setVertexes(LC, vertexes, SG.getVertexesNumber());
                joinVertexes(SG.getVertexesNumber());
                createMissingEdges(edges, SG.getEdgesNumber(), SG.getVertexesNumber());
                permutateVertexesID();
        }
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
private void copySmallGraph_to_LargeGraph(SmallGraph SG) {
	/* START hack */
	LinkedList<Vertex> listOfVertexes = new LinkedList<Vertex>();
	/* END hack */

	for (LinkedList<Vertex> llSG : SG.getSmallGraph()) {
		Vertex v = llSG.getFirst().clone();
		LinkedList<Vertex> llLG = new LinkedList<Vertex>();
		llLG.addLast(v);
		largeGraph.add(llLG);
		/* START hack */
		listOfVertexes.add(v);
		/* END hack */
	}

	for (int i = 0; i < SG.getSmallGraph().size(); i++) {
		LinkedList<Vertex> llSG = SG.getSmallGraph().get(i);
		LinkedList<Vertex> llLG = largeGraph.get(i);

		Iterator<Vertex> IV = llSG.iterator();
		IV.next(); // This line is to skipping the first element of the list
		while (IV.hasNext()) {
			Vertex SGv = IV.next();
			for (Vertex c : listOfVertexes) {
				if (c.getId() == SGv.getId()) {
					llLG.addLast(c);
					break;
				}
			}
		}
	}

}

private void setVertexes(LabelsCreator LC, int vertexesLG, int vertexesSG) {
	int vertexToGenerate = vertexesLG - vertexesSG;
	for (int i = 0; i < vertexToGenerate; i++) {
		LinkedList<Vertex> L = new LinkedList<Vertex>();
		L.add(new Vertex(LC.getRandomLabel(), vertexesSG + i));
		largeGraph.add(L);
	}
}

/**
 * Join unbound vertexes and also creates only one connection between a
 * vertex from the SmallGraph and the unbound vertexes
 */
private void joinVertexes(int vertexSG) {
	for (int i = vertexSG - 1; i < largeGraph.size() - 1; i++) {
		largeGraph.get(i).add(largeGraph.get(i + 1).getFirst());
		largeGraph.get(i + 1).add(largeGraph.get(i).getFirst());
	}
}

private void createMissingEdges(int edgesLG, int edgesSG, int vertexesSG)
		throws InvalidVertexesEdgesDefinition {
	int missedEdges = edgesLG - edgesSG
			- (largeGraph.size() - vertexesSG - 1) - 1;
	if (missedEdges > 0) {
		int k = 0;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				long time_start, time_end;
				time_start = System.currentTimeMillis();
				while (k < missedEdges) {
					int v1 = RandomNumbers
							.getSimpleRamdom(0, largeGraph.size() - 1);
					int v2 = RandomNumbers.getSimpleRamdom(vertexesSG,
							largeGraph.size() - 1);
					time_end = System.currentTimeMillis();
					if (time_end - time_start > 20000) {
						throw new InvalidVertexesEdgesDefinition(
								"invalid paramenters");
					}
					if (!largeGraph.get(v1).contains(largeGraph.get(v2).getFirst())) {
						largeGraph.get(v1).add(largeGraph.get(v2).getFirst());
						largeGraph.get(v2).add(largeGraph.get(v1).getFirst());
						k++;
					}
				}
			}
		}

		private void permutateVertexesID() {
			/** Generate a permutated array */
			int[] permutate = new int[largeGraph.size()];
			for (int i = 0; i < permutate.length; i++) {
				permutate[i] = i;
			}
			int aux, random1, random2;

			for (int i = 0; i < permutate.length * 2; i++) {
				random1 = RandomNumbers.getSimpleRamdom(0, permutate.length - 1);
				random2 = RandomNumbers.getSimpleRamdom(0, permutate.length - 1);
				aux = permutate[random1];
				permutate[random1] = permutate[random2];
				permutate[random2] = aux;
			}

			/** start mixing vertexes */
			for (LinkedList<Vertex> lLLG : largeGraph) {
				lLLG.getFirst().setId(permutate[lLLG.getFirst().getId()]);
			}
		}

		public int getVertexesNumber() {
			return vertexes;
		}

		public int getEdgesNumber() {
			return edges;
		}

		public SmallGraph getSmallGraph() {
			return smallGraph;
		}

		public Matrix toMatrix() {
			Matrix M = new Matrix(vertexes, vertexes);

			for (LinkedList<Vertex> listOfVertexes : largeGraph) {
				Iterator<Vertex> iteratorVertex = listOfVertexes.iterator();
				Vertex firstVertex = iteratorVertex.next();
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				while (iteratorVertex.hasNext()) {
					M.setIJ(firstVertex.getId(), iteratorVertex.next().getId());
				}
			}

			return M;
	}

	public String toString() {
			String out = "";
			for (LinkedList<Vertex> ll : largeGraph) {
				Vertex v = ll.getFirst();
				out += "V" + " " + v.getId() + " " + v.getLabel() + "\n";
			}
			for (LinkedList<Vertex> ll : largeGraph) {
				Iterator<Vertex> iv = ll.iterator();
				Vertex rootV = iv.next();
				while (iv.hasNext()) {
					out += "E" + " " + rootV.getId() + " " + iv.next().getId()
							+ " " + "bond\n";
				}
			}
			return out;
	}

	public String toStringll() {
			String out = "";
			for (LinkedList<Vertex> ll : largeGraph) {
				for (Vertex v : ll) {
					out += "[" + v.getLabel() + "|" + v.getId() + "]->";
				}
				out += "\n";
			}
			return out;
	}
}
```

## VIII.2.5. Grafo menor (Objetivo)

```
package tesis.graphs.model;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;

import tesis.graphs.algorithms.Matrix;

public class SmallGraph {

	private ArrayList<LinkedList<Vertex>> smallGraph;
	private int vertexes;
	private int edges;

	public SmallGraph(LabelsCreator LC, int vertexes, int edges) {
		try {
			areParametersRight(vertexes, edges);
			this.vertexes = vertexes;
			this.edges = edges;
			smallGraph = new ArrayList<LinkedList<Vertex>>();
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				setVertexes(LC, vertexes);
				joinVertexes();
				createMissingEdges(edges);
			} catch (InvalidVertexesEdgesDefinition e) {
				e.printStackTrace();
			}
	}

	private void areParametersRight(int vertexes, int edges)
				throws InvalidVertexesEdgesDefinition {

			int minEdgesNumber = vertexes - 1;
			int maxEdgesNumer = (vertexes * (vertexes - 1)) / 2;
			if (!(edges >= minEdgesNumber && edges <= maxEdgesNumer)) {
				throw new InvalidVertexesEdgesDefinition(
						"Edges number must be between " + minEdgesNumber + " and "
								+ maxEdgesNumer + " for vertexes= " + vertexes);
			}
	}

	private void setVertexes(LabelsCreator LC, int vertexes) {

			for (int i = 0; i < vertexes; i++) {
				LinkedList<Vertex> L = new LinkedList<Vertex>();
				L.add(new Vertex(LC.getRandomLabel(), i));
				smallGraph.add(L);
			}
	}

	private void joinVertexes() {
			for (int i = 0; i < smallGraph.size() - 1; i++) {
				smallGraph.get(i).add(smallGraph.get(i + 1).getFirst());
				smallGraph.get(i + 1).add(smallGraph.get(i).getFirst());
			}
	}

	private void createMissingEdges(int edges) {
			int missingEdges = edges - (smallGraph.size() - 1);
			if (missingEdges > 0) {
				int k = 0;
				while (k < missingEdges) {
					int v1 = RandomNumbers
							.getSimpleRamdom(0, smallGraph.size() - 1);
					int v2 = RandomNumbers
							.getSimpleRamdom(0, smallGraph.size() - 1);

					if (!smallGraph.get(v1).contains(smallGraph.get(v2).getFirst())) {
						smallGraph.get(v1).add(smallGraph.get(v2).getFirst());
						smallGraph.get(v2).add(smallGraph.get(v1).getFirst());
						k++;
					}
				}
			}
	}

	public Matrix toMatrix() {
			Matrix M = new Matrix(vertexes, vertexes);

			for (LinkedList<Vertex> listOfVertexes : smallGraph) {
				Iterator<Vertex> iteratorVertex = listOfVertexes.iterator();
				Vertex firstVertex = iteratorVertex.next();
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				while (iteratorVertex.hasNext()) {
					M.setIJ(firstVertex.getId(), iteratorVertex.next().getId());
				}
			}

			return M;
		}

		public String toString() {
			String out = "";
			for (LinkedList<Vertex> ll : smallGraph) {
				Vertex v = ll.getFirst();
				out += "V" + " " + v.getId() + " " + v.getLabel() + "\n";
			}
			for (LinkedList<Vertex> ll : smallGraph) {
				Iterator<Vertex> iv = ll.iterator();
				Vertex rootV = iv.next();
				while (iv.hasNext()) {
					out += "E" + " " + rootV.getId() + " " + iv.next().getId()
								+ " " + "bond\n";
				}
			}
			return out;
		}

		public String toStringll() {
			String out = "";
			for (LinkedList<Vertex> ll : smallGraph) {
				for (Vertex v : ll) {
					out += "[" + v.getLabel() + "|" + v.getId() + "]->";
				}
				out += "\n";
			}
			return out;
		}

		public int getEdgesNumber() {
			return edges;
		}

		public ArrayList<LinkedList<Vertex>> getSmallGraph() {
			return smallGraph;
		}

		public int getVertexesNumber() {
			return vertexes;
		}
}
```

# VIII.2.6. Vértice

```
package tesis.graphs.model;

public class Vertex implements Cloneable {

		private char label;
		private int id;

		public Vertex(char label, int id) {
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                super();
                this.label = label;
                this.id = id;
        }

        public char getLabel() {
                return label;
        }

        public void setLabel(char label) {
                this.label = label;
        }

        public int getId() {
                return id;
        }

        public void setId(int id) {
                this.id = id;
        }
         public Vertex clone() {
             Vertex obj = null;
             try {
                obj = (Vertex) super.clone();
             } catch (CloneNotSupportedException ex) {
                System.out.println("Cloning is not possible");
             }
             return obj;
           }
}
```

## VIII.2.7. Generador de números aleatorios

```
package tesis.graphs.model;

public class RandomNumbers {

        /**Returns: x\from=<x<=upto*/
        public static int getSimpleRamdom(int from,int upto) {
                return (int)(Math.random()*(upto-from+1)+from);
        }

        public static int getPoisson(double lambda){

                double L = Math.exp(-lambda);
                double p = 1.0;
                int k = 0;

                do {
                        k++;
                        p *= Math.random();
                } while (p > L);
                return k - 1;
        }
}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

# VIII.3. CLASES DE PRUEBA

## VIII.3.1. Test de ejecución

```
package tesis.graphs.tests;

import java.text.DecimalFormat;
import java.util.ArrayList;

import tesis.graphs.algorithms.GMatrix;
import tesis.graphs.algorithms.GMatrixThreads;
import tesis.graphs.algorithms.Matrix;
import tesis.graphs.algorithms.Ullman;
import tesis.graphs.algorithms.UllmanThreads;
import tesis.graphs.model.FilePrinter;
import tesis.graphs.model.InvalidVertexesEdgesDefinition;
import tesis.graphs.model.LabelsCreator;
import tesis.graphs.model.LargeGraph;
import tesis.graphs.model.SmallGraph;

public class Test {
	public static void main(String[] args) {

		boolean foundSubIsomorphism = false;
		long time_start, time_end;
		long TimeUllman=0, TimeUllmanThread = 0, TimeGMatrix=0, TimeGMatrixThread = 0;

		DecimalFormat df = new DecimalFormat("#.###");
		FilePrinter FP = FilePrinter.getInstance();
		ParameterSelector PS = new ParameterSelector();

		LabelsCreator LC = null;
		LargeGraph LG = null;
		Matrix LGM = null;
		Matrix SGM = null;

		FP.write( "G|V|" + "\t"
				+ "G|E|" + "\t"
				+ "Densidad(G)" + "\t"
				+ "G'|V'|" + "\t"
				+ "G'|E'|" + "\t"
				+ "Densidad(G')" + "\t"
				+ "Ullman" + "\t"
				+ "UllmanThread" + "\t"
				+ "GMatrix" + "\t"
				+ "GMatrixThread"+"\n");

		System.out.print("G|V|" + "\t"
				+ "G|E|" + "\t"
				+ "Densidad(G)" + "\t"
				+ "G'|V'|" + "\t"
				+ "G'|E'|" + "\t"
				+ "Densidad(G')" + "\t"
				+ "Ullman" + "\t"
				+ "UllmanThread" + "\t"
				+ "GMatrix" + "\t"
				+ "GMatrixThread"+"\n");
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
            PS.setLocation(0);
            while (PS.hasNext()) {
                PS.next();
                for (double i = 1; i <= 10; i += 1) {
                    // START build largeGraph and SmallGraph in order to parameters
                    LC = new LabelsCreator(1);
                    try {
                        LG = new LargeGraph(LC, new SmallGraph(LC,
                                        PS.getSmallGraphVertexesNumber(),
                                        PS.getSmallGraphEdgesNumber()),
                                        PS.getLargeGraphVertexesNumber(),
                                        PS.getLargeGraphEdgesNumber());
                    } catch (InvalidVertexesEdgesDefinition e) {
                        e.printStackTrace();
                        break;
                    }
                    // Convert them into matrixes
                    LGM = LG.toMatrix();
                    SGM = LG.getSmallGraph().toMatrix();
                    // END


//==========================================================START TEST: Ullman
                    time_start = System.currentTimeMillis();
                    foundSubIsomorphism = Ullman.is_SubIsomorphism(SGM, LGM);
                    time_end = System.currentTimeMillis();

                    if (foundSubIsomorphism) {
                        TimeUllman = TimeUllman + (time_end-time_start);
                    }else{
                        TimeUllman = TimeUllman + 60000;
                    }
// ============================================================== END  Ullman

                    // Cleaning Variables for next test
                    LGM = LG.toMatrix();
                    SGM = LG.getSmallGraph().toMatrix();
                    time_end=0;
                    time_start=0;
                    foundSubIsomorphism=false;
                    // END


//==========================================================START TEST: GMatrix
                    time_start = System.currentTimeMillis();
                    foundSubIsomorphism = GMatrix.is_SubIsomorphism(SGM, LGM);
                    time_end = System.currentTimeMillis();

                    if (foundSubIsomorphism) {
                        TimeGMatrix = TimeGMatrix + (time_end-time_start);
                    }else{
                        TimeGMatrix = TimeGMatrix + 60000;
                    }

//==============================================================END GMatrix

                    // Cleaning Variables for next test
                    LGM = LG.toMatrix();
                    SGM = LG.getSmallGraph().toMatrix();
                    time_end=0;
                    time_start=0;
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                                        foundSubIsomorphism=false;
                                        // END


        //=================================================START test: GMatrixThread
                                        time_start = System.currentTimeMillis();
                                        MySignal mySignal = new MySignal();
                                        ArrayList<Matrix> ALM =
GMatrixThreads.getAPartOfM(GMatrixThreads.buildStartMatrix(SGM, LGM));
                                        ArrayList<GMatrixThreads> ALGMT = new ArrayList<GMatrixThreads>();
                                        ArrayList<Thread> ALThre = new ArrayList<Thread>();

                                        for(Matrix Ma : ALM){
                                                ALGMT.add(new
GMatrixThreads(Ma,SGM,LGM,mySignal,ALThre));
                                        }
                                        for(GMatrixThreads GMT : ALGMT){
                                                ALThre.add(new Thread(GMT));
                                        }

                                        for(Thread Th : ALThre){
                                                Th.start();
                                        }
                                        for (Thread Th : ALThre){
                                                try {
                                                        Th.join(60000);
                                                } catch (InterruptedException e) {
                                                        e.printStackTrace();
                                                }
                                        }

                                        if(mySignal.isFinished()){
                                                time_end = mySignal.getTimeEnd();
                                                foundSubIsomorphism = true;
                                        }else{
                                                time_end = System.currentTimeMillis();
                                                foundSubIsomorphism = false;
                                        }

                                        if (foundSubIsomorphism) {
                                                TimeGMatrixThread = TimeGMatrixThread + (time_end - time_start
);
                                        }else{
                                                TimeGMatrixThread = TimeGMatrixThread + 60000;
                                        }

        //=======================================================END GMatrixThreads

                                        //Wait for all the threads to finish to make sure they don't affect the next test
                                        while(ALThre.get(0).isAlive() || ALThre.get(1).isAlive() ||
ALThre.get(2).isAlive()){
                                                try {
                                                        Thread.sleep(10);
                                                } catch (InterruptedException e) {
                                                        e.printStackTrace();
                                                }
                                        }
                                        //END Waiting

                                        // Cleaning Variables for next test
                                        LGM = LG.toMatrix();
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
				SGM = LG.getSmallGraph().toMatrix();
				time_end=0;
				time_start=0;
				mySignal = null;
				foundSubIsomorphism=false;
				// END


	//=======================================================START test: UllmanThreads
				time_start = System.currentTimeMillis();
				mySignal = new MySignal();
				ArrayList<Matrix> ALM1 =
GMatrixThreads.getAPartOfM(UllmanThreads.buildStartMatrix(SGM, LGM));
				ArrayList<UllmanThreads> ALUT = new ArrayList<UllmanThreads>();
				ArrayList<Thread> ALThre1 = new ArrayList<Thread>();

				for(Matrix Ma : ALM1){
					ALUT.add(new UllmanThreads(Ma,SGM,LGM,mySignal,ALThre1));
				}
				for(UllmanThreads UT : ALUT){
					ALThre1.add(new Thread(UT));
				}

				for(Thread Th : ALThre1){
					Th.start();
				}
				for (Thread Th : ALThre1){
					try {
						Th.join(60000);
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}

				if(mySignal.isFinished()){
					time_end = mySignal.getTimeEnd();
					foundSubIsomorphism = true;
				}else{
					time_end = System.currentTimeMillis();
					foundSubIsomorphism = false;
				}

				if (foundSubIsomorphism) {
					TimeUllmanThread = TimeUllmanThread + (time_end-time_start);
				}else{
					TimeUllmanThread = TimeUllmanThread + 60000;
				}

	//===========================================================END UllmanThreads

				//Wait for all the threads to finish to make sure they don't affect the next test
				while(ALThre1.get(0).isAlive() || ALThre1.get(1).isAlive() ||
ALThre1.get(2).isAlive()){
					try {
						Thread.sleep(10);
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}
				//END Waiting
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                // Cleaning Variables for next test
                time_end=0;
                time_start=0;
                mySignal = null;
                foundSubIsomorphism=false;
                // END

                // Free RAM
                System.runFinalization();
                System.gc();
                // END free RAM
            }
            //promediate
            TimeUllman =(long) Math.floor( TimeUllman/10);
            TimeUllmanThread = (long) Math.floor(TimeUllmanThread/10);
            TimeGMatrix =(long) Math.floor(TimeGMatrix/10);
            TimeGMatrixThread =(long) Math.floor(TimeGMatrixThread/10);
            //END

            FP.write(PS.getSmallGraphVertexesNumber() + "\t"
                        + PS.getSmallGraphEdgesNumber() + "\t"
                        + df.format(PS.getSmallGraphDensity()) + "\t"
                        + PS.getLargeGraphVertexesNumber() + "\t"
                        + PS.getLargeGraphEdgesNumber() + "\t"
                        + df.format(PS.getLargeGraphDensity()) + "\t"
                        + TimeUllman + "\t"
                        + TimeUllmanThread + "\t"
                        + TimeGMatrix + "\t"
                        + TimeGMatrixThread + "\n");

            System.out.print(PS.getSmallGraphVertexesNumber() + "\t"
                        + PS.getSmallGraphEdgesNumber() + "\t"
                        + df.format(PS.getSmallGraphDensity()) + "\t"
                        + PS.getLargeGraphVertexesNumber() + "\t"
                        + PS.getLargeGraphEdgesNumber() + "\t"
                        + df.format(PS.getLargeGraphDensity()) + "\t"
                        + TimeUllman + "\t"
                        + TimeUllmanThread + "\t"
                        + TimeGMatrix + "\t"
                        + TimeGMatrixThread + "\n");

            //reset variables
            foundSubIsomorphism = false;
            LC = null;
            LG = null;
            LGM = null;
            SGM = null;
            time_start=0;
            time_end=0;
            TimeUllman=0;
            TimeUllmanThread=0;
            TimeGMatrix=0;
            TimeGMatrixThread=0;
            //END reset

            // Free RAM
            System.runFinalization();
            System.gc();
            // END free RAM
        }
        System.out.println("***|||********END TEST********|||***");
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
	}

}
```

## VIII.3.2. Selector de parámetros

```
package tesis.graphs.tests;

import java.util.ArrayList;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ParameterSelector{

	private int largeGraphVertexesNumber;
	private int largeGraphEdgesNumber;
	private double largeGraphDensity;

	private int smallGraphVertexesNumber;
	private int smallGraphEdgesNumber;
	private double smallGraphDensity;

	private ArrayList<int[]> ParameterList;
	private int location;

	public ParameterSelector(){

		String csvFile = "/home/geronimo/Escritorio/Tesis/sub-isomorfismo/Tabla
MonteCarlo/tabla_montecarlo_valores.csv";
		BufferedReader br = null;
		String line = "";
		String cvsSplitBy = ";";

		this.ParameterList = new ArrayList<int[]>();
		this.location = 0;
		try {

			br = new BufferedReader(new FileReader(csvFile));
			while ((line = br.readLine()) != null) {
				int[] parameters = new int[4];
				String[] numero = line.split(cvsSplitBy);
				parameters[0] = Integer.valueOf(numero[3]);
				parameters[1] = Integer.valueOf(numero[5]);
				parameters[2] = Integer.valueOf(numero[0]);
				parameters[3] =Integer.valueOf(numero[2]);
				ParameterList.add(parameters);
			}

		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			if (br != null) {
				try {
					br.close();
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
    }

    public boolean hasNext(){
                boolean hasNext = false;
                if(ParameterList.size() >= location +1){
                    hasNext = true;
                }
                return hasNext;
    }
    public void next(){
                int[] parameters = ParameterList.get(location++);
                largeGraphVertexesNumber = parameters[0];
                largeGraphEdgesNumber = parameters[1];
                largeGraphDensity =(double) largeGraphEdgesNumber
/(largeGraphVertexesNumber*(largeGraphVertexesNumber-1)/2);

                smallGraphVertexesNumber = parameters[2];
                smallGraphEdgesNumber = parameters[3];
                smallGraphDensity =(double)
smallGraphEdgesNumber/(smallGraphVertexesNumber*(smallGraphVertexesNumber-1)/2);
    }

    public void setLocation(int location){
            this.location = location;
    }

    public int getLocation(){
            return this.location;
    }

    private double getEdgesNumber(double density, double vertexes){
            double edges = Math.floor((density*(vertexes*vertexes) - density*vertexes)/2);
            return edges;
    }

    @SuppressWarnings("unused")
    private double minDensityOf_LG(double sgd,double sgvn, double lgvn){
                double minDensity = (2*getEdgesNumber(sgd, sgvn) + 2*(lgvn - sgvn))/(lgvn*(lgvn-
1));

                return minDensity;
    }

    @SuppressWarnings("unused")
    private double maxDensityOf_LG(double lgvn,double sgvn){
            double maxDensity = (2*(lgvn - sgvn -1) + 2*sgvn*(lgvn-sgvn))/(lgvn*(lgvn-1));
            return maxDensity;
    }

    public int getLargeGraphVertexesNumber() {
            return largeGraphVertexesNumber;
    }

    public int getLargeGraphEdgesNumber() {
            return largeGraphEdgesNumber;
    }
```



Repositorio Digital Institucional "José María Rosa"
UNLa.

```
	public double getLargeGraphDensity() {
		return largeGraphDensity;
	}

	public int getSmallGraphVertexesNumber() {
		return smallGraphVertexesNumber;
	}

	public int getSmallGraphEdgesNumber() {
		return smallGraphEdgesNumber;
	}

	public double getSmallGraphDensity() {
		return smallGraphDensity;
	}
	public int getListSize(){
			return ParameterList.size();
	}

}
```

## VIII.3.3. Señal para detener ejecución

```
package tesis.graphs.tests;

public class MySignal{

	protected volatile boolean isFinished = false;
	protected long time_end;

	public synchronized boolean isFinished(){
	  return this.isFinished;
	}

	public synchronized void setFinished(){
	  this.isFinished = true;
	}
	public synchronized void setTimeEnd(long time_end){
		this.time_end = time_end;
	}
	public long getTimeEnd(){
		return this.time_end;
	}

}
```



Repositorio Digital Institucional "José María Rosa"
UNLa.