**UNIVERSIDAD MAYOR DE SAN ANDRÉS**

**FACULTAD DE CIENCIAS PURAS Y NATURALES**

**CARRERA DE INFORMÁTICA**

****

**COMPARACIÓN Y REFINAMIENTO DE ALGORITMOS DE COLOREADO DE GRAFOS**

**Tesis de Grado para obtener el Título de Licenciatura en Informática**

**Mención Ciencias de la Computación**

**POR: OSCAR ANTONIO RONDO RODRIGUEZ**

**DOCENTE: LIC. GROVER ALEX RODRIGUEZ RAMIREZ**

**TUTOR: M.SC. JORGE HUMBERTO TERÁN POMIER**

**LA PAZ – BOLIVIA**

**Febrero, 2022**

ÍNDICE

[**Capítulo 1 Antecedentes Generales o Marco Referencial**](#_hiiv3bw7828s) **4**

[**Introducción**](#_xgy4euacky54) **4**

[**Problema**](#_1fob9te) **6**

[Antecedentes](#_38cmbpio8jwn) 6

[Planteamiento del Problema](#_r9ocif7xczar) 9

[Formulación del Problema](#_2et92p0) 10

[**Objetivos**](#_3dy6vkm) **10**

[Objetivo General](#_1t3h5sf) 10

[Objetivos Específicos](#_4d34og8) 10

[**Hipótesis**](#_2s8eyo1) **10**

[Operacionalización de Variables](#_17dp8vu) 10

[**Justificación**](#_yk6eanq82uki) **11**

[Justificación Tecnológica](#_26in1rg) 11

[Justificación Social](#_kk4bnpxp8tr5) 12

[Justificación Científica](#_lilb2dufk6ic) 12

[**Alcances y Límites**](#_lnxbz9) **13**

[Alcances](#_8x1z82v1axyi) 13

[Límites](#_1ksv4uv) 13

[**Metodología**](#_1y810tw) **13**

[**Capítulo 2 Marco Teórico**](#_4i7ojhp) **13**

[Definiciones básicas](#_ybt5ghbz8oe6) 14

[Grafo](#_7wfeneejcpu) 14

[Caminos](#_3umfa5r5uewm) 14

[Ciclos](#_ulenxtya6uls) 15

[Grafo completo](#_iks47mxoouzo) 15

[Subgrafos](#_9u1j9pgg5j4m) 15

[Distancia](#_pstto5t5pfup) 16

[Conjunto Independiente](#_nana7o1kde1) 16

[Isomorfismo](#_6k14h1m90ae3) 17

[Heurísticas](#_krk3pv2pedvy) 18

[Paralelismo](#_dmwzx6lcwwdq) 18

[Coloreado de Grafos](#_pax85odgmk3p) 18

[Algoritmos Heurísticos de coloreado de Grafos](#_xssns8jlvz5e) 20

[Algoritmos Paralelos de coloreado de Grafos](#_lgsrz9y5uj2c) 22

[¿Por qué el Procesamiento Paralelo?](#_q6ps9wvykejf) 23

[Modelo Paralelo](#_mour5z5ifzku) 23

[Técnicas Básicas de Paralelismo](#_z33kjeg72h4p) 25

[**Capítulo 3 Diseño Metodológico**](#_iqssd7wp8cw4) **28**

[Generación de Grafos Aleatorios](#_skcizlb7wsy9) 29

[Algoritmo DSatur](#_9a57dtplttgh) 33

[Algoritmo RFL](#_80yevcl73kx3) 35

[Algoritmo Jones-Plassmann](#_m0ddu2gzpwzz) 38

[**Capítulo 4 Evaluación de Resultados**](#_96fel1906gf2) **40**

[Resultados](#_kfyoaaxa8uu7) 40

[Demostración de la Hipótesis](#_nujxw5m1bxt6) 49

[Conclusiones](#_x0xgjbqf93ig) 51

[**Bibliografía**](#_3whwml4) **52**

# **Capítulo 1 Antecedentes Generales o Marco Referencial**

## **Introducción**

El presente trabajo se centra en la asignación de etiquetas llamadas colores a elementos del grafo, lo que es conocido en ciencias de la computación como coloreado de grafos. El problema, en su forma más general, es el de asignar colores a los vértices de un grafo que consiste en un conjunto de n vértices V y un conjunto de m aristas E de tal forma que puede tener cualquiera de las siguientes asignaciones donde:



(No puede haber 2 vértices adyacentes que compartan el mismo color)

* K es mínimo

El coloreado de grafos tiene varias aplicaciones en el mundo real, como ser la construcción horarios o partición de grupos entre muchas más, la mayoría de estos problemas requiere una partición de conjuntos sujeta a ciertas restricciones, y dicha partición debe ser lo más pequeña posible (Aslan y Akhan, 2016).

La primera forma de afrontar el problema, la más intuitiva, podría ser comprobar una a una todas las posibles asignaciones entre colores y vértices para finalmente escoger la mejor de todas ellas, el máximo número de colores a utilizar seria |V| = n. En este caso tendríamos un total de soluciones posibles, número que crece de forma notable a medida que el valor de n aumenta, algo que ni el ordenador más potente podría resolver.

Sin embargo, la tasa de crecimiento exponencial del espacio de soluciones no es la única razón que hace del problema de coloración de grafos un problema de dificultad extrema. El hecho de que problemas como estos se consideren “complejos” o “intratables” se debe al trabajo realizado por (Cook, 1971) introdujo los conceptos de “NP-completitud” y de “reducción en tiempo polinomico”. Ademas, demostró que el problema conocido como “problema de satisfacibilidad” es NP-completo. Dicho problema plantea si, dada una expresión booleana cualquiera, existe alguna asignación entre valores y variables tal que la expresión se evalúe como “verdadero” (Pena, 2016).

El problema de coloreado de grafos generaliza el problema el problema NP-completo de la 3-satisfacibilidad, es decir que el problema de la 3-satisfacibilidad es polinomicamente reducible al problema de coloración de grafos.

A pesar de lo mencionado, el hecho de que el problema de coloración de grafos sea NP-completo no significa que no podamos encontrar soluciones en tiempo polinómico, para esto utilizaremos 2 tipos de algoritmos. Los algoritmos heurísticos y los algoritmos aproximados, estos nos darán “soluciones de buena calidad” en un tiempo polinómico (Lewis, 2016).

En el presente trabajo se pretende realizar la implementación y refinado de diferentes algoritmos Heurísticos y Paralelos con la finalidad de evaluar el desempeño en términos de cantidad de colores usados, tiempo de ejecución y espacio de memoria ocupado.

## **Problema**

### **Antecedentes**

El problema de k-coloreado de un grafo consiste en la asignación en uno de los k colores a cada vértice en el grafo de tal forma que 2 vértices adyacentes no compartan en mismo color. El número cromático de un grafo es el k más pequeño tal que el grafo puede ser coloreado con k vértices.

En el caso de los algoritmos heurísticos. El algoritmo base, y a la vez el más simple es el algoritmo voraz. Este algoritmo opera tomando los vértices del grafo uno a uno, siguiendo un orden (aleatorio), y asignando a cada vértice el primer color disponible. Al tratarse de un algoritmo heurístico, la solución proporcionada por este algoritmo puede no ser la mejor, pero una correcta elección del orden de los vértices puede producir una solución óptima para cualquier grafo. Este tiene una complejidad de donde n es el número de nodos.

En los años 70 se propuso un algoritmo llamado RFL (“Recursive Largest First”), propuesto por (Leighton, 1979), este algoritmo funciona coloreando un grafo con un color por cada iteración en lugar de un vértice por iteración. En cada iteración, el algoritmo busca conjuntos de vértices independientes en el grafo, los cuales son asociados a un solo color, posteriormente este conjunto es eliminado.

DSATUR, propuesto por (Brelaz, 1979), es similar a RLF. En lugar del grado del vértice, el orden de los vértices de color está determinado por su saturación. La saturación de un vértice se define como el número de colores diferentes a los cuales el vértice está conectado. Si ninguno de los vértices vecinos ha sido coloreado todavía, es igual al grado del vértice. Inicialmente el vértice con el grado más alto (el número de vértices conectados, ya que aún no se ha coloreado ningún vértice) está coloreado. Luego los otros vértices se colorean con el color más bajo posible en orden de grado de saturación decreciente, donde se seleccionan los lazos seleccionando el vértice con el grado más alto Este método es un poco más refinado que RLF ya que prioriza vértices con pocos colores disponibles en lugar de tratar todos los vértices con el mismo grado igual Brelaz demostró que para los gráficos bipartitos el algoritmo es exacto, lo que significa que está garantizada para encontrar una solución óptima.

Para grafos con número cromático constante el primer caso de aproximación no trivial fue planteado por (Wigderson,1983) para el caso especial de los grafos 3-coloreables. Este algoritmo usa el hecho de que en un grafo 3-coloreable el vecindario de cualquier vértice de este grafo es 2-coloreable. Este colorea el un grafo 3-coloreable con O () colores, y en general colorea cualquier grafo k-coloreable con colores.

La Búsqueda Tabú o TabuCol, es una técnica metaheuristica propuesta por (Fred Glover, 1986). El método se centra en la búsqueda de óptimos locales, mediante la prohibición de ciertos movimientos. El objetivo de clasificar algunos movimientos como prohibidos (“tabú"), es evitar que el algoritmo entre en bucles cíclicos. Estos movimientos pierden su estatus de tabú para volver a ser accesibles después de un periodo relativamente corto de tiempo.

Más tarde (Berger y Rompel, 1988) mejoraron este limite a colores, el cual para el caso especial de los grafos 3-coloreables se podía realizar con colores.

El año (Halldorson, 1990) planteo un algoritmo que supero a los anteriores con una garantía de rendimiento que era una relación del número de colores utilizados para el número cromático que era de . Mas adelante (Blum, 1994) propuso un algoritmo para los grafos k-coloreables pero enfocándose en el caso específico de k = 3 donde mejora el límite a colores.

El año (J. R. Allwright and R. Bordawekar and P. D. Coddington and K. Dincer and C. L. Martin, 1995) realizaron y plantearon comparaciones sobre algoritmos paralelos, paralelizaron los algoritmos Heurísticos Degree-First y Smallest-Degree-Last para realizar las comparaciones.

Posteriormente (Karger, Motwani y sudan, 1998) plantearon un algoritmo que podía colorear grafos k-colorables y para el caso específico de k = 3 tenían colores.

(Galinier y Hao, 1999) propusieron un algoritmo llamado HEA (Hybrid Evolutionary Algorithm). La HEA opera manteniendo una población de soluciones candidatas que se desarrollan a través de una recombinación específica del problema y un método de búsqueda local.

Más recientemente (Blochliger y Zufferey, 2008) propusieron un algoritmo similar a la búsqueda Tabú y lo llamaron PartialCol. Este no considera soluciones impropias, en su lugar los vértices que no pueden ser asignados a ninguno de los k colores sin causar un conflicto se colocan en un conjunto de vértices incoloros. El objetivo del algoritmo es hacer alteraciones en la solución parcial hasta que el conjunto de vértices incoloros quede vacío.

(Thompson y Dowsland, 2008) propusieron el algoritmo AntCol que es otro método metaheuristico que combina la búsqueda local y global, en este caso utilizando la optimización de la colonia de hormigas. La idea detrás del algoritmo es la de usar “hormigas” que produzcan soluciones candidatas. Cada “hormiga” producirá su solución de una manera no determinística, utilizando probabilidades basadas en heurísticas y también en la calidad de las soluciones producidas por hormigas anteriores. En particular, si las hormigas anteriores han identificado características que pueden llevar a soluciones mejores que el promedio, es más probable que la hormiga actual incluya estas características en su propia solución, lo que generalmente lleva a una reducción en el número de colores durante el curso de la ejecución.

En trabajos posteriores (Murat y Nurdan, 2016) se realizaron la comparación de varios algoritmos heurísticos como First Fit (FF), Largest Degree Ordering (LDO), Incidence Degree Ordering (IDO), Recursive Largest First (RLF) y Degrees of Saturation (DSatur) donde el desempeño de cada algoritmo fue comparado con diferentes casos de prueba en términos de colores utilizados y tiempos de ejecución.

### **Planteamiento del Problema**

La incorporación se algoritmos paralelos sobre grafos k-coloreables se ha realizado en

algunos trabajos (J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer y C. L. Martin), (Assefaw Hadish Gebremedhin y Fredrik Manne) los cuales se enfocan específicamente en la comparación de algoritmos paralelos.

En el caso de los algoritmos heurísticos existen algunos trabajos (Aslan y Akhan, 2016),

(Pena, 2017) donde se utilizan distintos tipos de grafos k-coloreables.

Dichos trabajos se centran en algoritmos particulares ya sean paralelos y/o heurísticos.

Dichos trabajos se enfocan en un solo tipo algoritmos ya sea heurístico y/o paralelo por lo que no existe una aplicación ni implementaciones de manera extensa mucho menos un

análisis del desempeño entre dichos algoritmos.

### **Formulación del Problema**

¿Cuál es el desempeño en términos de tiempo, memoria y colores utilizados de los algoritmos Heurísticos y Paralelos para el coloreado de grafos sobre diferentes implementaciones y casos de prueba?

## **Objetivos**

### **Objetivo General**

Estudiar y realizar la evaluación, análisis y refinado de distintos algoritmos heurísticos y paralelos en distintos casos de prueba, comparando el tiempo de ejecución y cantidad de colores utilizados.

### **Objetivos Específicos**

* Utilizar grafos aleatorios para la evaluación de desempeño, que sean aptos para el coloreado.
* Evaluar el desempeño en términos de tiempo y cantidad de colores usados.
* Utilizar grafos de DIMACS como casos de prueba
* Utilizar la notación Big-O para el análisis de complejidad.
* Realizar el refinado de los diferentes algoritmos

## **Hipótesis**

Dado el estudio, análisis y refinamiento de los algoritmos paralelos y heurísticos, los algoritmos paralelos tienen un mejor desempeño en términos de tiempo y los algoritmos heurísticos tienen un mejor desempeño en términos de colores utilizados.

### **Operacionalización de Variables**

Refinado de Algoritmos (variable independiente)

| DIMENSIONES | INDICADORES |
| --- | --- |
| Estudio y análisis de los algoritmos | Estudio de la naturaleza de los algoritmos y su desempeño sobre diferentes implementaciones y casos de prueba. |
| Tunning de los algoritmos | Mejoras sobre las implementaciones de algoritmos heuristicos y paralelos de coloreado de grafos. |

Desempeño del algoritmo (variable dependiente)

| Dimensiones | Indicadores |
| --- | --- |
| Tiempo | Cantidad de segundos empleados |
| Número de Colores | Cantidad de colores usados |

## **Justificación**

### **Justificación Tecnológica**

Actualmente el problema de Coloreado de Grafos es un problemas de complejidad extrema, suponiendo que nuestro grafo tiene n vértices, necesitamos a lo sumo n colores y en este caso tendríamos un total de n^n soluciones, número que crece notablemente a medida que la cantidad de vértices aumenta, algo que ni el ordenador mas potente podria resolver.

Debido a que actualmente los ordenadores no pueden resolver el problema de coloreado cuando la cantidad de vértices del grafo en muy grande, optamos por soluciones aproximadas.

Los algoritmos heurísticos y/o paralelos reducen el tiempo computacional, esto hace que el problema sea tratable con soluciones aproximadas para grafos con una gran cantidad de vértices, estas opciones son las más utilizadas en la práctica.

### **Justificación Social**

En la actualidad con el avance de la tecnología a pasos agigantados, las soluciones a cierto tipo de problemas como Horarios con tablas de tiempo, registro de asignaciones y sudokus entre otros requieren de optimizaciones para grandes cantidades de datos, es necesario brindar soluciones e información para poder aplicar el algoritmo correcto para cada tipo de problema.

### **Justificación Científica**

El coloreado de grafos es una de las áreas más estudiadas de la teoría de grafos, pero debido a la intratabilidad del problema se han planteado alternativas por medio de algoritmos paralelos, heurísticos y/o exactos, existen ciertos trabajos relacionados a estos (Blum, 1994), (Olav, 2010), (Kawarabayashi y Oseki, 2010), (J. R. Allwright y R. Bordawekar y P. D. Coddington y K. Dincer and C. L. Martin, 1995) pero dichos trabajos están enfocados a algoritmos o grafos particulares donde dichos algoritmos no son comparados ni analizados entre ellos.

Por otro lado, podemos ver ciertos trabajos donde se describen algoritmos de coloreado de grafos (Pena, 2017), (Aslan y Nurdan, 2016) los cuales cuentan con algoritmos Heurísticos.

Por lo tanto, el trabajo se justifica, no solo por los resultados obtenidos en el coloreado de distintos tipos de grafos, sino también puede dar origen a optimizaciones de los diferentes algoritmos secuenciales y paralelos ya sea en términos de tiempo, memoria y cantidad de colores usados con la finalidad de aportar al área de las ciencias de la computación.

## **Alcances y Límites**

### **Alcances**

* Para la comparación de los diferentes algoritmos se tomarán en cuenta grafos aleatorios generados con diferentes probabilidades de aristas entre nodos
* Se realizará un análisis teórico mediante una función que limitará el tiempo de cálculo del algoritmo y una prueba experimental de dónde se recogerán estadísticas de tiempo consumido por los diferentes algoritmos.

### **Límites**

* No se tomará en cuenta el coloreado parcial.
* No se realizarán coloreados impropios.
* Todo coloreado será factible.
* Se realizarán las comparaciones únicamente con algoritmos heurísticos y/o paralelos
* solo se tomarán en cuenta grafos aleatorios no direccionados y sin pesos

## **Metodología**

Para el desarrollo del presente trabajo se utilizará el método lógico inductivo ya que partiremos de casos particulares de coloreado de grafos para posteriormente llegar a una conclusión respecto a cada uno de los algoritmos que serán comparados y refinados.

# **Capítulo 2 Marco Teórico**

Ahora veremos algunas definiciones y algoritmos que usaremos en este trabajo. Este incluye notación básica sobre grafos y su coloreado. Las siguientes definiciones son recopiladas principalmente de (Lewis, 2016), (Blum, 1994), (Guangping, 2016), (Olav, 2010), (Husfeldt, 2013), (Demaine y Karger, 2003), (Pena, 2016), (Williams, 2017), (Moss, 2017), (Per Normann, 2014)

## **Definiciones básicas**

### **Grafo**

Un grafo es un conjunto de vértices V y aristas . En este trabajo utilizaremos únicamente grafos no dirigidos, donde . El número de vértices y aristas está dado por respectivamente. Por cada vértice el vecindario de v está definido por , donde . El grado de un vértice está definido por . Se dice que un vértice w es adyacente de un vértice v si . El grado máximo de un grafo es . Un camino de tamaño n es una secuencia de vértices

### **Caminos**

Un grafo cuyos vértices están ordenados en una fila es llamado grafo camino o generalmente llamado camino como lo vemos en la figura 1.

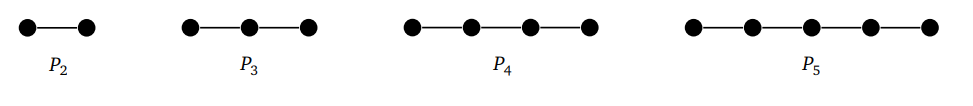


Figura 1. Grafos Camino

Fuente: Brian Heinold, 2018

Formalmente el camino Pn tiene un conjunto de vertices {V1, V2, ….. Vn} y un conjunto de aristas {Vi Vi+1: i = 1, 2, … n}

### **Ciclos**

Si ordenamos los vértices y las aristas de un grafo G alrededor de un ciclo o polígono tendremos un grafo ciclo, generalmente llamado ciclo como podemos verlo en la figura 2.

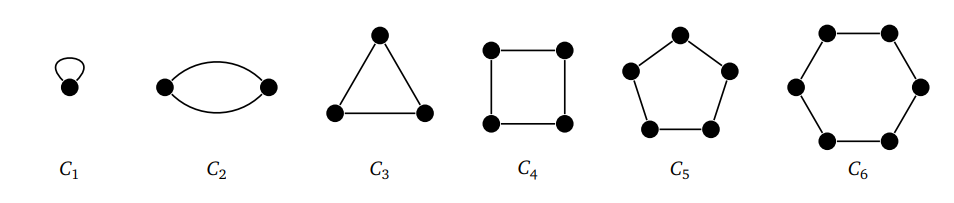


Figura 2. Grafos Ciclo

Fuente: Brian Heinold, 2018

### **Grafo completo**

Un grafo completo es un grafo en el cual cada vértice es adyacente con los demás vértices del grafo como lo vemos en la figura 3.

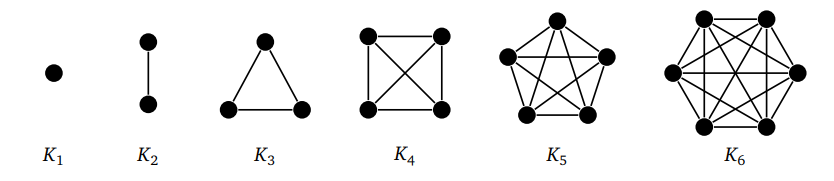


Figura 3. Grafos Completo

Fuente: Brian Heinold, 2018

### **Subgrafos**

Un subgrafo es parte de un grafo donde tomamos un subconjunto de sus vértices y las aristas asociados a estos como lo vemos en la figura 4.

formalmente podemos decir que un grafo H is un subgrafo del grafo G donde los vértices de H son un subconjunto de los vértices de G y las aristas de H son un subconjunto de de las aristas de G como vemos en la figura 4.

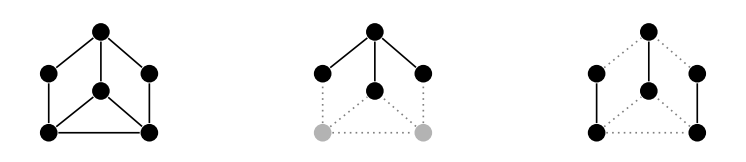


Figura 4. Grafos Completo

Fuente: Brian Heinold, 2018

### **Distancia**

En teoría de grafos la distancia entre 2 vértices es la medida de que tan separados están estos

La distancia entre 2 vértices denotada como d(u,v) es la longitud del camino más corto entre u y v en el grafo

Como vemos en la figura 5. b,c y d están a una distancia de 1 de a. El vértices e está a una distancia de 2 de a y el vértice f esta a una distancia 3 de a

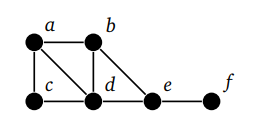


Figura 5. Ejemplo de distancias

Fuente: Brian Heinold, 2018

### **Conjunto Independiente**

Un conjunto independiente o más conocido como independent set (IS) es un subconjunto de vértices en un grafo tal que ninguno de sus vértices es adyacente a otro. El problema de decidir si un grafo contiene un conjunto independiente de tamaño k es NP-completo.

### **Isomorfismo**

El termino isomorfismo significa que 2 estructuras tienen la misma forma, viene de diferentes ramas de la matematica en teoria de grafos cuando decimos que dos grafos son isomorfos nos referimos a que en realidad son el mismo grafo, dibujados o presentados de manera diferente. Por ejemplo como vemos en la FIgura 6 vemos dos grafos isomorfos



Figura 6. Grafos Isomorfos

Fuente: Brian Heinold, 2018

Formalmente podemos decir que los grafos G y H son isomorfos probando que existe una relación de uno a uno sobre la función f: V(G) -> V(H) tal que v es adyacente a w en G si y sólo si f(v) es adyacente a f(w) en H

Usando la definicion podemos ver que los siguientes grafos en la figura 7 son isomorfos

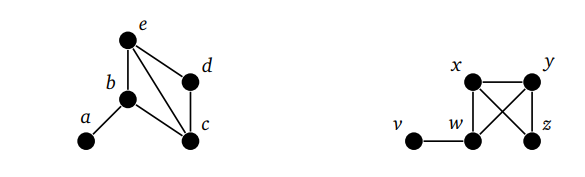


Figura 7. Grafos Isomorfos

Fuente: Brian Heinold, 2018

Asociando los vértices de la siguiente manera, (a,v), (b,w), (c,x), (d,s) y (c,y)

## 

## **Heurísticas**

Es una técnica diseñada para resolver problemas rápidamente con una solución aproximada cuando los métodos clásicos son demasiado lentos. Esta se logra a partir del sacrificio o pérdida de la precisión.

## **Paralelismo**

El paralelismo es una forma de [computación](https://es.wikipedia.org/wiki/Computaci%C3%B3n) en la cual varios cálculos pueden realizarse simultáneamente,[1](https://es.wikipedia.org/wiki/Paralelismo_(inform%C3%A1tica)#cite_note-1)​ basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo. Hay varios tipos diferentes de paralelismo: [nivel de bit](https://es.wikipedia.org/w/index.php?title=Paralelismo_a_nivel_de_bit&action=edit&redlink=1), [nivel de instrucción](https://es.wikipedia.org/w/index.php?title=Paralelismo_a_nivel_de_instrucci%C3%B3n&action=edit&redlink=1), [de datos](https://es.wikipedia.org/wiki/Paralelismo_de_datos) y [de tarea](https://es.wikipedia.org/w/index.php?title=Pararelismo_de_tarea&action=edit&redlink=1). El paralelismo ha sido empleado durante muchos años, sobre todo para la [Computación de alto rendimiento](https://es.wikipedia.org/wiki/Computaci%C3%B3n_de_alto_rendimiento).

## **Coloreado de Grafos**

El coloreado de grafos es el etiquetado de elementos del grafo, usualmente llamamos a estas etiquetas colores. Ya sean aristas, vértices o ambos. En este trabajo tomaremos en cuenta únicamente el coloreado de vértices. Para un grafo , una asignación de k-colores está dada por la función . Decimos que c es propia si para todos los pares . Decimos que G es k-coloreable si existe una asignación propia en G. El numero cromático de G es es el k más pequeño tal que el grafo G es k-coloreable. Dado un coloreado c, la clase de un color para el color i es en conjunto . El resultado más importante en el coloreado de grafos es el teorema de los 4 colores. Debido a la intratabilidad del problema podemos ver grafos en los cuales se pueden encontrar algoritmos que realicen el coloreado en un tiempo polinomial. Por ejemplo. Grafos completos, grafos bipartitos y grafos planos entre otros.

**Teorema de los 4 colores.** El teorema de los 4 colores consiste básicamente, en que cualquier mapa o grafo planar puede ser coloreado únicamente con 4 colores de tal forma que 2 regiones adyacentes no comparten el mismo color.

**Grafos completos**. Un grafo completo con n vértices es un grafo en el cual existe una arista entre cualquier par de vértices con un conjunto aristas. Esto es obvio ya que todos los vértices en un grafo completo son mutuamente adyacentes, todos los vértices deben tener asignados un color. Por lo tanto, el número cromático de un grafo completo es n.

**Grafos bipartitos**. Los grafos bipartitos denotados por , son grafos cuyos vértices pueden ser particionados en 2 conjuntos y tales que solo existen aristas entre los vértices de y los vértices de . En consecuencia, y son conjuntos independientes, por lo que cualquier grafo bipartito puede ser coloreado con 2 colores, asignando a todos los vértices de el primer color y todos los vértices de el segundo color. Resulta evidente si y sólo si G es bipartito.

**Grafos Planos**. Los grafos planos son aquellos que pueden ser representados en el plano de tal forma que ninguna de sus aristas corta a otra. Según el teorema de los 4 colores un grafo plano puede ser coloreado con 4 colores o menos.

## **Algoritmos Heurísticos de coloreado de Grafos**

**Algoritmo voraz**. Es uno de los más simples y fundamentales entre los algoritmos heurísticos para el coloreado de grafos. El algoritmo opera tomando vértices uno por uno acorde a algún orden, este puede ser arbitrario y asigna a cada vértice algún color que esté disponible en ese momento, la cantidad de colores usados puede variar dependiendo del orden de los vértices utilizados, el algoritmo voraz siempre da soluciones optimas escogiendo el orden correcto de los vértices.

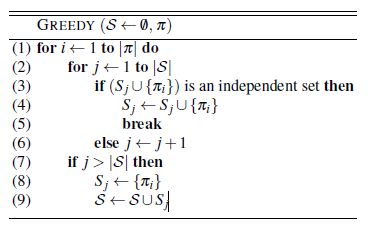


Figura 8. Algoritmo voraz para el coloreado de grafos

Fuente: Lewis, 2016

El el pseudocodigo de la Figura 8. Inicialmente, el algoritmo toma la solución vacia y una permutación arbitraria de los vértices. En cada iteración del bucle exterior el algoritmo toma el i-esimo vértice de la permutación y trata de encontrar una clase de color en la que este pueda ser insertado, si es que dicha clase de color existe, entonces el vértice es añadido a esta y nos movemos al siguiente vértice . Caso contrario, las líneas 8-9 del algoritmo son usados para crear una nueva clase de color para el vértice.

**Algoritmo DSatur.** El algoritmo DSatur es una abreviación de “degree of saturation”. En esencia es muy similar al comportamiento del algoritmo voraz, en este último tomamos un vértice de una permutación inicial arbitraria y asignamos un color a este creando nuevas clases de colores. La diferencia entre estos algoritmos viene dada por la manera de generar el orden inicial de los vértices. El algoritmo DSatur la elección del siguiente vértice a colorear es decidido heurísticamente basada en las características del coloreado parcial actual del grafo. Esta selección se basa principalmente en el grado de saturación de los vértices.

**Grado de saturación.** Decimos que para todo vértice que actualmente no tiene una clase de color. Dado un vértice v, el grado de saturación de v, esta denotado como sat(v), este es el número de colores diferentes asignados a los vértices adyacentes, esto es,

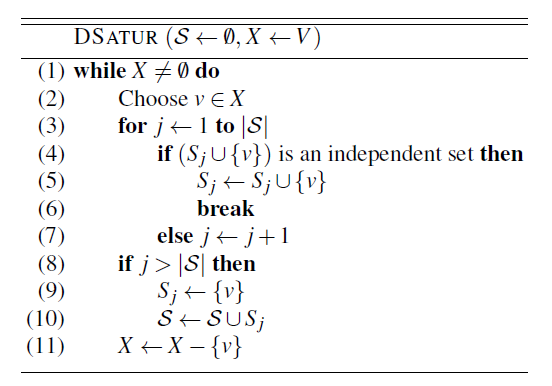


Figura 9. Algoritmo DSatur

Fuente: Lewis, 2016

Podemos ver que la mayoría del algoritmo es similar al algoritmo voraz. La mayor diferencia entre el algoritmo voraz y el algoritmo DSatur se puede ver en las líneas 1, 2 y 11 del pseudocodigo. Aquí, un conjunto X es usado para definir el conjunto de vértices que actualmente no tienen color, al inicio de la ejecución X=V. En cada iteración del algoritmo el siguiente vértice a ser coloreado es seleccionado de X acorde a la línea 2, y una vez coloreado, removemos este vértice de X en la línea 11. El algoritmo termina cuando

**Algoritmo de Wigderson.** Es un algoritmo de coloreado de grafos aproximado que colorea cualquier grafo 3-coloreable con colores, y en general puede colorear cualquier grafo k-coloreable con .

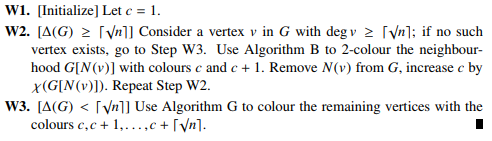
****

Figura 10. Algoritmo de Wigderson

Fuente: Lewis, 2016

Podemos ver que en el paso W2 se menciona al algoritmo B, se supone que este debería bicolorear el subgrafo en un tiempo polinomial. En el paso W3 se menciona al algoritmo G, dicho algoritmo es el algoritmo voraz.

## **Algoritmos Paralelos de coloreado de Grafos**

**Algoritmo de Luby Jones.** Es un algoritmo paralelo de coloreado de grafos que colorea cualquier grafo k-coloreable. La idea básica es iterar sobre un conjunto de vértices numerados y asignarles colores, a cada vértice se le asigna un número randómico, los mínimos locales son coloreados con el mismo color en cada iteración, esta tarea se realiza en paralelo

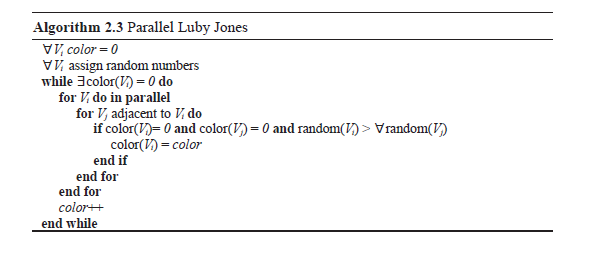


Figura 11. Algoritmo de Luby Jones

Fuente: Per Normann, 2014

## **¿Por qué el Procesamiento Paralelo?**

El propósito principal del procesamiento en paralelo es lograr una mayor computación

velocidad convencional mediante el uso de varios procesadores al mismo tiempo. La búsqueda de este objetivo ha tenido una significativa en influencia en muchas actividades relacionadas con la informática. La necesidad de soluciones más rápidas y de resolver problemas de mayor procesamiento.

Surge en una amplia gama de aplicaciones. Estos incluyen, entre otros,

predicción meteorológica, procesamiento y extracción de información, procesamiento de imágenes, inteligencia artificial y fabricación automatizada.

Los siguientes tres factores principales han contribuido a la fuerte tendencia a favor del procesamiento paralelo:

La constante disminución del costo del hardware, avances en tecnología de hardware y software, y la limitación física en el tiempo de ciclo más rápido de un solo procesador.

Estos factores han llevado a los investigadores a explorar el paralelismo y su potencial.

utilizar en aplicaciones importantes.

## **Modelo Paralelo**

La RAM (Random Access Machine) es un modelo utilizado con éxito para predecir

el rendimiento de los programas en computadoras con un solo procesador (secuencial).

La extensión natural de este modelo para computadoras paralelas es la memoria compartida. Este modelo consta de varios procesadores, cada uno de los cuales tiene su propia memoria local y puede ejecutar su propio programa local, y todos de los cuales se comunican mediante el intercambio de datos a través de una unidad de memoria compartida, también llamada memoria global.

Hay dos modos básicos de funcionamiento de un modelo de memoria compartida: sincrono y asincrono. En el primer modo, todos los procesadores operan sincrónicamente bajo el control de un reloj común, mientras que en el segundo modo cada procesador opera bajo un reloj separado.

Dado que cada procesador puede ejecutar su propio programa local, el modelo de memoria es un tipo de datos múltiples de instrucciones múltiples (MIMD). Donde cada procesador puede ejecutar una instrucción u operar con datos diferentes de aquellos ejecutados u operados por cualquier otro procesador durante cualquier unidad de tiempo.

El modelo de memoria compartida síncrona es un modelo idealizado utilizado para el desarrollo y análisis de algoritmos paralelos. Un nombre estándar para este modelo es el modelo de máquina de acceso aleatorio paralelo (PRAM).

Los algoritmos desarrollados para el modelo PRAM han sido del tipo de única instrucción de datos múltiples (SIMD). Es decir, todos los procesadores ejecutan lo mismo

programa tal que, durante cada unidad de tiempo, todos los procesadores activos estén procesando la misma instrucción, pero con datos diferentes en general. Sin embargo, tal como está el modelo, se pueden cargar diferentes programas en la memoria local de los procesadores, siempre que los procesadores puedan funcionar sincrónicamente.

Por lo tanto, se pueden ejecutar diferentes tipos de instrucciones dentro de la unidad de tiempo

asignado para un paso. En otras palabras, el modelo PRAM no excluye la desarrollo de algoritmos MIMD.

Hay varias variaciones del modelo PRAM basadas en el supuesto sobre el manejo del acceso simultáneo de varios procesadores a la misma ubicación de la memoria global. La escritura exclusiva de lectura exclusiva.

(EREW) PRAM no permite ningún acceso simultáneo a una sola localización de memoria. La PRAM de escritura exclusiva de lectura simultánea (CREW) permite acceso instantáneo para una instrucción de lectura solamente. La lectura concurrente PRAM de escritura concurrente (CRCW) permite el acceso simultáneo tanto para lectura y escritura instrucciones. El CRCW PRAM está especializado en diferentes campos, depende de cómo se rompan los lazos en el acceso simultáneo a la memoria para cada instrucción de escritura. Los tres modelos PRAM (EREW, CREW, CRCW) no difieren sustancialmente en su poder computacional. Procesamiento realizado en un p-procesador, una variante del modelo PRAM, se puede simular en otra variante con una desaceleración de un factor no mayor que O (log p).

## **Técnicas Básicas de Paralelismo**

La tarea de diseñar algoritmos paralelos presenta desafíos que son considerablemente más difíciles que los encontrados en el dominio secuencial.

La falta de una metodología bien definida se compensa con una colección de

técnicas y paradigmas que se han encontrado efectivos en el manejo de una amplia

variedad de problemas.

La estrategia de partición consiste en romper el problema dado p subproblemas independientes de tamaños casi iguales, y luego solución de los subproblemas p al mismo tiempo, donde p es el número de procesadores disponible.

* 1. . **Modelos de Programación Paralela**

El modelo PRAM es un modelo idealizado de una computadora paralela.

Podemos identificar dos categorías arquitectónicas principales: arquitectura de memoria distribuida y arquitectura de memoria compartida.

En la arquitectura de memoria distribuida, los procesadores se conectan mediante un

red de interconexión de paso de mensajes. Cada procesador tiene su propia memoria local, accesible sólo a sí misma. Los procesadores pueden interactuar solo pasando mensajes.

En la arquitectura de memoria compartida, hay soporte de hardware para lectura y

acceso de escritura de todos los procesadores a un espacio de direcciones compartido. Los procesadores interactúan modificando los objetos de datos almacenados en el espacio de direcciones compartido.

Modelo de programación de paso de mensajes. En este modelo, el programador ve sus programas como una colección de procesos con variables locales privadas con la capacidad de enviar y recibir datos entre procesos pasando

mensajes. En este modelo, no se comparten variables entre procesadores. Cada procesador usa sus variables locales y ocasionalmente envía o recibe datos de otros procesadores. El estilo de programación de transmisión de mensajes es, naturalmente, adecuado para computadoras con memoria distribuida.

Modelo de programación de memoria compartida. En este modelo, el programador

ven sus programas como una colección de procesos que acceden a un grupo central de

variables compartidas. El estilo de programación de memoria compartida se adapta naturalmente a computadoras con memoria compartida.

El ordenador comparte datos almacenándolos en una memoria accesible a nivel global. Cada proceso accede a los datos compartidos leyendo o escribiendo en variables compartidas.

Sin embargo, más de un procesador puede acceder a la misma variable compartida en

un momento. Esto puede causar problemas. Lenguajes de programación de memoria compartida

* 1. **Algoritmos Exactos de coloreado de Grafos**

Los algoritmos exactos son aquellos que siempre determinarán la solución óptima a un problema, dado el exceso de tiempo. Una forma de resolver exactamente NP-completo

problemas tales como el problema de coloración de gráficos es buscar exhaustivamente la solución

Sin embargo, a medida que aumenta el tamaño de los problemas, el tiempo de ejecución

para una búsqueda exhaustiva pronto se vuelve restrictivamente grande. Dicho esto, todavía es posible diseñar algoritmos exactos que son significativamente más rápidos que la búsqueda exhaustiva, aunque todavía no opera en tiempo polinomial.

* 1. **Backtracking**

En el problema de Coloreado de Grafos es posible ampliar algoritmos constructivos para formar lo que se conoce como algoritmos de retroceso. Retroceder es un

metodología general que se puede utilizar para determinar una solución óptima (o posiblemente todas las soluciones óptimas) a un problema computacional como el coloreado de grafos.

Los algoritmos de retroceso de esencia funcionan construyendo sistemáticamente soluciones parciales en soluciones completas. Sin embargo, durante este proceso de construcción tan pronto como se obtiene evidencia que nos dice que no hay forma de completar la actual solución para obtener una solución óptima, el algoritmo retrocede para tratar de encontrar

formas adecuadas de ajustar la solución parcial actual.

Un algoritmo de retroceso simple para colorear gráficos podría funcionar de la siguiente manera.

Dado un grafo G = (V, E), los vértices se ordenan primero de alguna manera para que el vértice vi (1 ≤ i ≤ n) corresponde al i-ésimo vértice en este orden. También tenemos que elegir un valor k que indica el número de colores disponibles. Inicialmente, esto podría ser simplemente k = ∞. El algoritmo de retroceso luego realiza una serie de pasos hacia adelante y hacia atrás.

Los pasos hacia adelante colorean los vértices en el orden dado hasta que se identifica un vértice que no se puede colorear con uno de los k colores disponibles. Pasos hacia atrás

por otro lado, retrocede por los vértices de colores en orden inverso e identifique

puntos donde se pueden realizar diferentes asignaciones de color a los vértices. Los pasos hacia adelante se continúan desde estos puntos. Si se encuentra una coloración completa factible, entonces k puede establecerse en el número de colores utilizados en esta coloración menos 1, con el algoritmo luego continuando. En última instancia, el algoritmo termina cuando un paso hacia atrás alcanza el vértice raíz v1, o cuando algún otro criterio de detención.

# **Capítulo 3 Diseño Metodológico**

Para el presente capítulo es necesaria la lectura de los capítulos anteriores por los conceptos que se manejan los cuales son nombrados y fueron introducidos y demostrados anteriormente. El capítulo se centra en el desarrollo de los algoritmos junto a su respectivo análisis de complejidad y espacio utilizado.

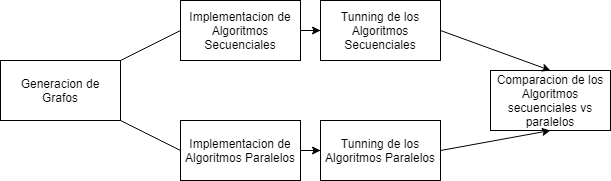
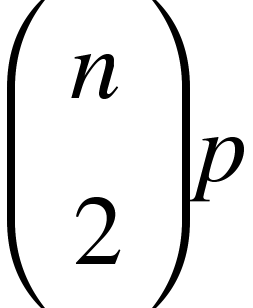


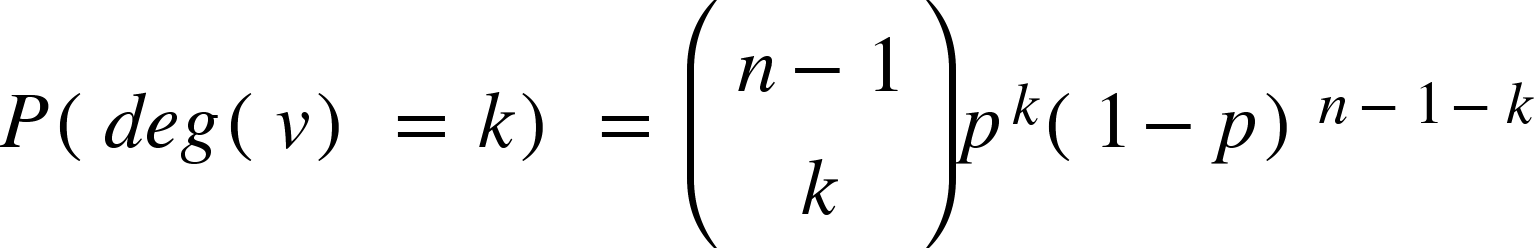
Figura 12.

Fuente: propia

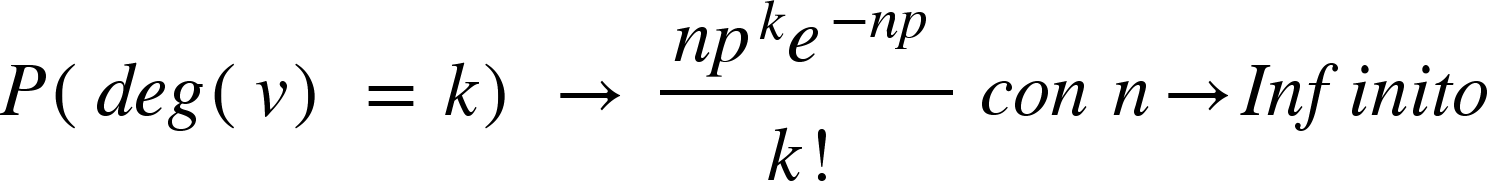
## **Generación de Grafos Aleatorios**

Para medir el tiempo y la calidad de las respuestas utilizaremos el modelo de Erdos Renyi G(n,p) donde definiremos n como la cantidad de nodos y p la probabilidad con la que se añade una arista al grafo.

Para G (n, p) tiene en promedio  aristas. La distribución del grado de cualquier vértice en particular es binomial:



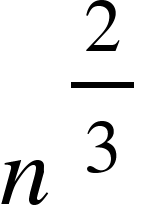
Donde n es el número total de vértices en el grafo. Desde

 y np = constante

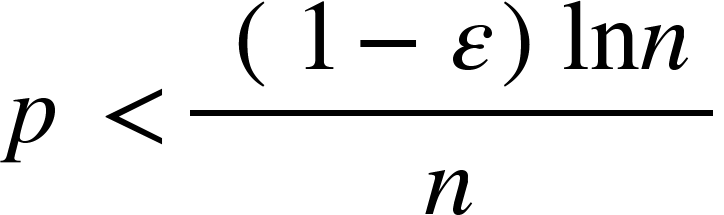
Esta distribución es de Poisson para n grandes y np = constante.

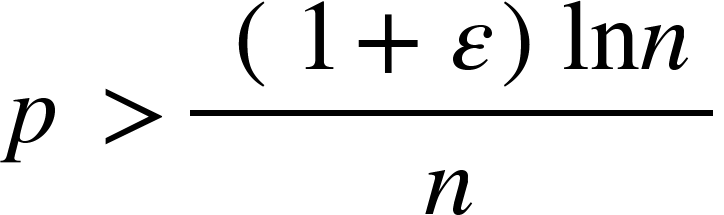
Erdos y Rényi (1960) describieron el comportamiento de G (n, p) con mucha precisión para varios valores de p. Sus resultados concluyeron que:

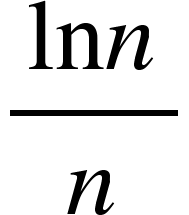
Si np < 1, entonces un grafo G (n, p) casi seguramente no tendrá componentes conectados de tamaño mayor que O (log (n)).

Si np = 1, entonces una grafo G (n, p) casi seguramente tendrá un componente más grande cuyo tamaño sea de orden .

Si np <math xmlns="http://www.w3.org/1998/Math/MathML"><mo>&#x2192;</mo></math> c> 1, donde c es una constante, entonces un grafo G (n, p) casi seguramente tendrá un componente gigante único que contiene una fracción positiva de los vértices. Ningún otro componente contendrá más de O (log (n)) vértices.

Si  , entonces un grafo G (n, p) casi seguramente contendrá vértices aislados y, por lo tanto, estará desconectado.

Si , entonces es casi seguro que un grafo G (n, p) esté conectado.

Por tanto,  es un umbral agudo para la conectividad de G (n, p).

Otras propiedades del gráfico se pueden describir casi con precisión ya que n tiende a infinito. Por ejemplo, hay un k (n) (aproximadamente igual a 2log2 (n)) tal que el clique más grande en G (n, 0.5) tiene casi seguramente tamaño k (n) o k (n) + 1.

Por lo tanto, aunque encontrar el tamaño del clique más grande en un gráfico es NP-completo, el tamaño del clique más grande en un grafo “típico” (según este modelo) se comprende muy bien.

Para generar los grafos aleatorios binomiales se utilizo la libreria networkx del lenguaje de programación Python

Erdos\_renyi\_graph (n, p, seed = Ninguno, directed = False)

Devuelve un Grafo aleatorio G (n, p), también conocido como gráfico de Erdos-Rényi o gráfico binomial.

El modelo G (n, p) elige cada uno de los posibles vertices con probabilidad p.

Las funciones binomial\_graph () y erdos\_renyi\_graph () son alias de esta función.

Parámetros: n (int): el número de nodos.

p (flotante): probabilidad de creación de bordes.

seed (int, opcional): seed para el generador de números aleatorios (predeterminado = Ninguno).

Dirigido (bool, opcional (predeterminado = Falso)): si es Verdadero, esta función devuelve un gráfico dirigido.

Por ejemplo en la Figura 13 y 14 podemos ver grafos generados con el modelo G(n, p) con 10 vértices con una probabilidad de 0.5 y 1 de arista respectivamente.

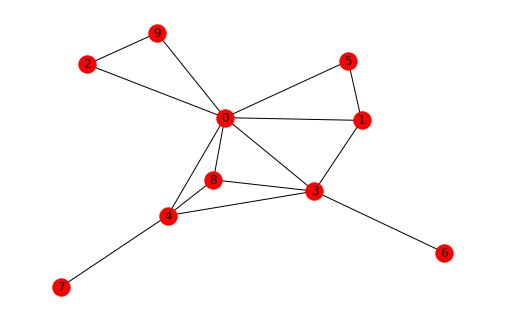


Figura 13. Grafo G(10, 0.5)

Fuente: Propia

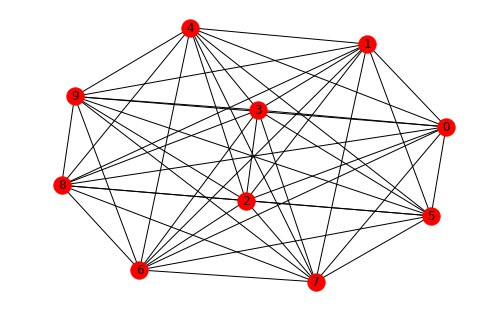


Figura 14. Grafo G(10, 1) (Clique)

Fuente: Propia

## **Algoritmo DSatur**

Dado el Grafo G(v, e) donde V es el conjunto de vértices y E el conjunto de aristas, iniciaremos asignando a todos los vertices un color Nulo es decir que inicialmente c(v) = NULL para todo vertice v que pertenece a V.

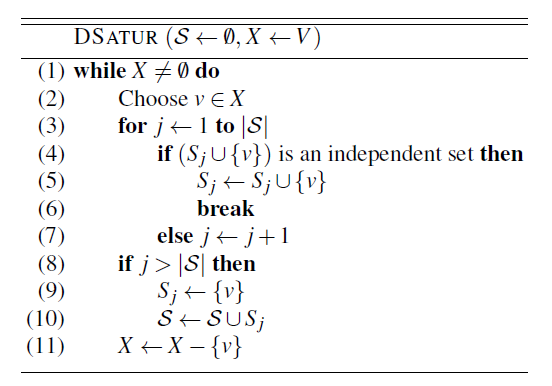


Figura 15. Algoritmo DSatur

Fuente: Lewis, 2016

Podemos ver que la mayoría del algoritmo es similar al algoritmo voraz. La mayor diferencia entre el algoritmo voraz y el algoritmo DSatur se puede ver en las líneas 1, 2 y 11 del pseudocodigo. Aquí, un conjunto X es usado para definir el conjunto de vértices que actualmente no tienen color, al inicio de la ejecución X=V. En cada iteración del algoritmo el siguiente vértice a ser coloreado es seleccionado de X acorde a la línea 2, y una vez coloreado, removemos este vértice de X en la línea 11. El algoritmo termina cuando

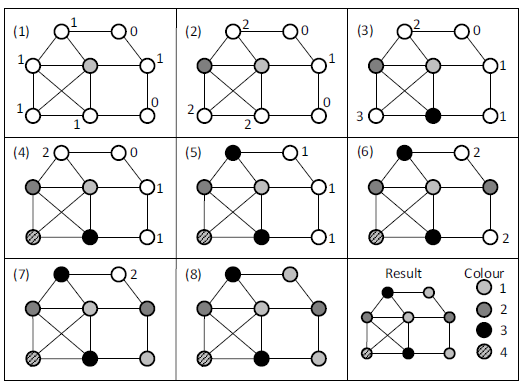


Figura 16. Ejecución Algoritmo DSatur

Fuente: Lewis, 2016

## 

En la figura 16 podemos ver un ejemplo de ejecución del algoritmo en un pequeño gráfico.

Para empezar, todos los vértices tienen un grado de saturación igual a 0, por lo que el primer vértice a ser

coloreado es el que tiene el grado más alto. Como se muestra en el Paso (1), esto se asigna al color 1. Esto también conduce a cinco vértices con un grado de saturación de 1, por lo que el próximo vértice a elegir es el que tiene el grado más alto entre estos. Esto es

luego asignado al color 2 como se muestra en el Paso (2). A continuación, tres vértices tienen saturación de grado 2, por lo que volvemos a elegir el vértice entre estos con el grado más alto.

Dado que los colores 1 y 2 no son factibles para este vértice, se asigna al color 3.

Este proceso continúa como se muestra en la figura 7 hasta que se haya obtenido una coloración factible.

En el capítulo 2 vimos que la cantidad de colores utilizados en las soluciones producidas por

El algoritmo GREEDY depende del orden en que se introducen los vértices en el procedimiento con resultados (en términos de la cantidad de colores utilizados en la solución producida) potencialmente variando mucho. Por otro lado, el algoritmo DSATUR reduce esta varianza generando el orden de vértices durante una ejecución de acuerdo con sus heurísticas.

## **Algoritmo RFL**

Si bien el algoritmo DSatur para colorear grafos es similar en comportamiento y complejidad

Para el enfoque clásico de Greedy, el algoritmo Recursive Largest First (RLF), sigue una estrategia ligeramente diferente.

El algoritmo RLF fue diseñado originalmente por Leighton (1979), en parte para

utilizar en la construcción de soluciones a grandes problemas de horarios. El método funciona por colorear un grafo de un color a la vez, en lugar de un vértice a la vez. En cada

paso el algoritmo utiliza heurística para identificar un conjunto independiente de vértices en el grafo que luego se asocian con el mismo color. Este conjunto independiente es entonces

eliminado del gráfico, y el proceso se repite en el subgrafo resultante.

Este proceso continúa hasta que el subgrafo está vacío, momento en el que todos los vértices han sido coloreados dejando con una solución factible. Leighton (1979) ha probado

la complejidad del peor caso de RLF es O (n^3), lo que le da un mayor costo computacional

que los algoritmos O (n^2) GREEDY y DSATUR; sin embargo, este algoritmo sigue siendo acotado polinomialmente

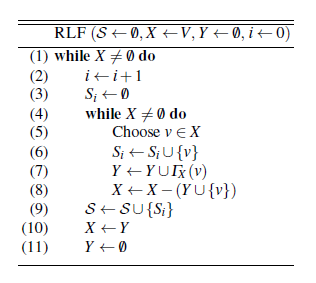


Figura 17. Algoritmo RFL

Fuente: Lewis, 2016

El pseudocódigo para el algoritmo RLF se muestra en la Figura 17. En cada bucle exterior

del proceso, se construye la i-ésima clase de color Si. El algoritmo también utiliza dos conjuntos:

X, que contiene vértices sin colorear que actualmente se pueden agregar a Si sin

sin causar choque.

Y, que contiene los vértices incoloros que no pueden ser factibles añadidos a Si. Al inicio de la ejecución X = V e Y = 0.

Las líneas (4) a (8) de la Figura 8 dan los pasos que se encargan de construir la i-ésima

clase de color Si. Para comenzar, se selecciona un vértice v de X y se agrega a Si (es decir, v es coloreado con el color i). A continuación, todos los vértices vecinos v en el subgrafo inducidos por X se transfieren a Y, para significar que ahora no pueden asignarse de manera factible a Si. Finalmente, v y sus vecinos también se eliminan de X, ya que ahora no son

considerados candidatos para su inclusión en la clase de color Si.

Una vez que X = 0, no se pueden agregar más vértices a la clase de color actual Si.

Por tanto, las líneas (9) a (11) del algoritmo Si se añaden a la solución S y, si

necesario, el algoritmo pasa a construir la clase de color Si + 1. Para hacer esto todos

los vértices en el conjunto de vértices incoloros Y se mueven a X, y Y se vacía.

Obviamente, una vez que tanto X como Y están vacíos, se han coloreado todos los vértices.

Las heurísticas sugeridas por Leighton (1979) para seleccionar el siguiente vértice v ∈ X para

color en la línea (5) siguen una lógica similar a las del algoritmo DSATUR en que

se priorizan los vértices más "restringidos". En consecuencia, el primer vértice elegido

para la inserción en cada clase de color Si es el miembro de X que tiene el grado más alto

en el subgrafo inducido por X. Los vértices restantes v para Si se seleccionan como

el miembro de X que tiene el mayor grado en el subgrafo inducido por Y ∪ {v} (que

es decir, el vértice de X adyacente al mayor número de vértices de Y). Al igual que con

DSATUR, cualquier vínculo en estas heurísticas se puede romper al azar.

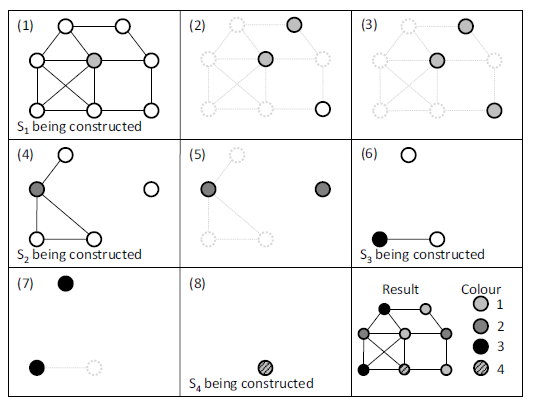


Figura 18. Ejecución del Algoritmo RFL

Fuente: Lewis, 2016

La figura 18 muestra un ejemplo paso a paso del algoritmo RLF.

Se indican los pasos que implican la creación de una nueva clase de color Si. En el Paso (1) el

el vértice v con el grado más alto en el gráfico se agrega a la clase de color S1. En el paso (2),

todos los vértices adyacentes a v ahora se han movido a Y, dejando un subgrafo inducido por

el conjunto X que contiene sólo dos vértices, los cuales se agregan posteriormente a

clase de color S1 en los Pasos (2) y (3). En el paso (4) se crea una nueva clase de color y el

El proceso se repite en el subgrafo inducido por los vértices incoloros restantes.

Esto continúa hasta que se hayan coloreado todos los vértices.

## **Algoritmo Jones-Plassmann**

A diferencia del algoritmos de Plassmann donde se generaban pesos aleatorios después de cada coloreado para obtener un conjunto independiente máximo en esta variante se puede construir un único conjunto de pesos aleatorios únicos al principio y usarlo en el algoritmo de coloreado. Esto se puede hacer fácilmente asignando números aleatorios a cada

de los vértices y usando el número de vértice único para resolver los problemas de conflictos que tienen los algoritmos paralelos.

El algoritmo de Jones-Plassmann procede entonces de manera muy similar al algoritmo MIS, excepto que no encuentra un conjunto independiente máximo en cada paso. Simplemente encuentra un conjunto independiente en paralelo usando el método de Luby para elegir vértices cuyos pesos son locales son máximos. La otra diferencia es que los vértices del conjunto independiente no son

asignaron el mismo color nuevo, ya que el comportamiento es similar al algoritmo MIS. En cambio, los vértices son coloreado individualmente utilizando el color más pequeño disponible, es decir, el color más pequeño que no ha ya ha sido asignado a un vértice vecino.

Este procedimiento se repite, una descripción del algoritmo de Jones Plassmann se muestra en la Figura 19.

Un ejemplo de coloración de Jones Plassmann se puede ver en la Figura 20. La coloración final es lo mismo que produciría el algoritmo codicioso si eligiera los vértices en

orden de sus pesos, el mayor peso primero.

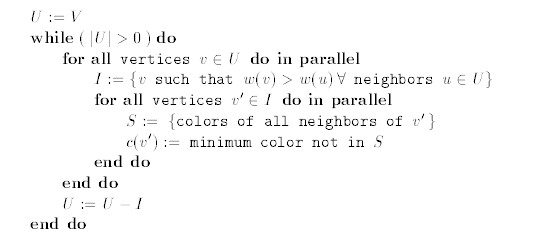


Figura 19. Algoritmo Jones-Plassmann

J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, C. L. Martin, 1995

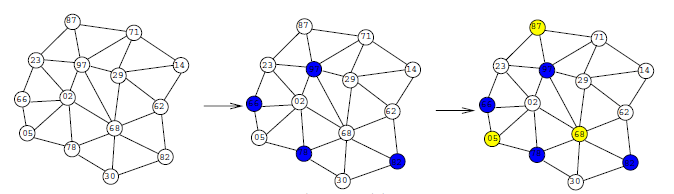


Figura 20. Ejecución del Algoritmo Jones-Plassmann

J. R. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, C. L. Martin, 1995

Inicialmente asignamos números aleatorios, luego cada vértice busca a sus vecinos y si tiene el número local más grande asignado entonces se le asigna el color más pequeño disponible.

Cada vértice que aún no tenga color busca a sus vecinos sin color asignado y es coloreado con el color más pequeño disponible.

# **Capítulo 4 Evaluación de Resultados**

En esta sección presentamos ahora una comparación de los algorimtos Jonnes-Plassmann, DSatur y RLF para ver particularmente el tiempo de ejecución y la calidad de soluciones que tienden a producir.

La plataforma de prueba fue una Notebook HP Intel® Core™ i7-4700MQ CPU 2.40GHz, RAM 8 GB. Los experimentos son evaluados en grafos de 1 a 19 MB con distintas probabilidades de existencia de aristas y estos mismos son usados para todos los algoritmos.

## **Resultados**

A continuación se exponen los resultados obtenidos en el coloreado de diferentes grafos con 100, 1000, 2000 y 4000 vértices y probabilidades de aristas de 0.1 - 1

Dsatur con color Negro

RFL con color Rojo

Jones-Plassmann con color Verde

La comparación de colores utilizados de los algoritmos descritos en el capítulo anterior usando Grafos con 100 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 21.

| **P** | **X(G) DSatur** | **X(G) RFL** | **X(G) Jones-Passmann** |
| --- | --- | --- | --- |
| 0.1 | 7 | 6 | 7 |
| 0.2 | 10 | 10 | 10 |
| 0.3 | 14 | 13 | 13 |
| 0.4 | 17 | 16 | 18 |
| 0.5 | 21 | 22 | 23 |
| 0.6 | 25 | 25 | 26 |
| 0.7 | 28 | 27 | 29 |
| 0.8 | 35 | 35 | 36 |
| 0.9 | 44 | 43 | 48 |
| 1 | 100 | 100 | 100 |

Tabla 1. Comparativa de Colores Con grafos de 100 vértices

Fuente: Propia

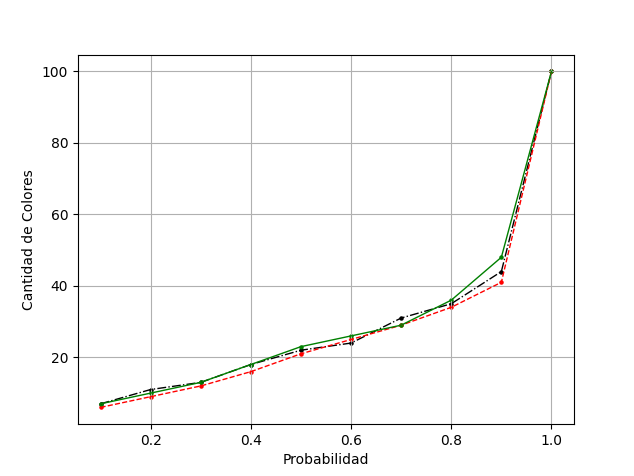


Figura 21. Comparativa de Colores Con grafos de 100 vértices

Fuente: Propia

La comparación de colores utilizados de los algoritmos descritos en el capítulo anterior usando Grafos con 1000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 22

| **P** | **X(G) DSatur** | **X(G) RFL** | **X(G) Jones-Passmann** |
| --- | --- | --- | --- |
| 0.1 | 31 | 30 | 31 |
| 0.2 | 54 | 51 | 54 |
| 0.3 | 74 | 72 | 76 |
| 0.4 | 100 | 96 | 101 |
| 0.5 | 127 | 123 | 124 |
| 0.6 | 152 | 151 | 155 |
| 0.7 | 192 | 187 | 193 |
| 0.8 | 242 | 236 | 244 |
| 0.9 | 317 | 319 | 318 |
| 1 | 1000 | 1000 | 1000 |

Tabla 2. Comparativa de Colores Con grafos de 1000 vértices

Fuente: Propia

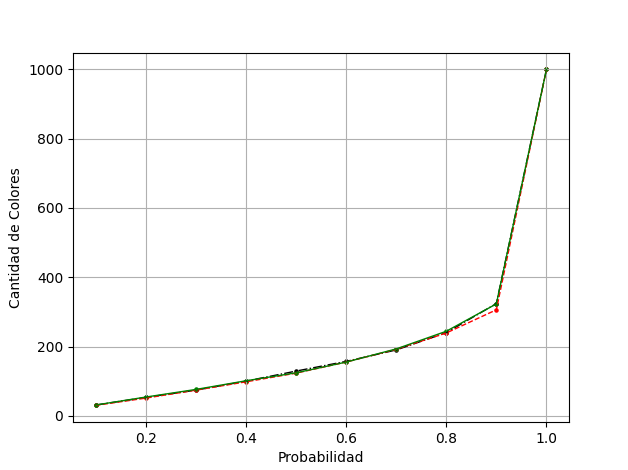


Figura 22. Comparativa de Colores Con grafos de 1000 vértices

Fuente: Propia

La comparación de colores utilizados de los algoritmos descritos en el capítulo anterior usando Grafos con 2000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 23.

| **P** | **X(G) DSatur** | **X(G) RFL** | **X(G) Jones-Passmann** |
| --- | --- | --- | --- |
| 0.1 | 52 | 50 | 54 |
| 0.2 | 91 | 89 | 92 |
| 0.3 | 131 | 130 | 132 |
| 0.4 | 173 | 171 | 178 |
| 0.5 | 224 | 219 | 224 |
| 0.6 | 279 | 275 | 279 |
| 0.7 | 346 | 343 | 346 |
| 0.8 | 435 | 431 | 437 |
| 0.9 | 583 | 579 | 586 |
| 1 | 2000 | 2000 | 2000 |

Tabla 3. Comparativa de Colores Con grafos de 2000 vértices

Fuente: Propia

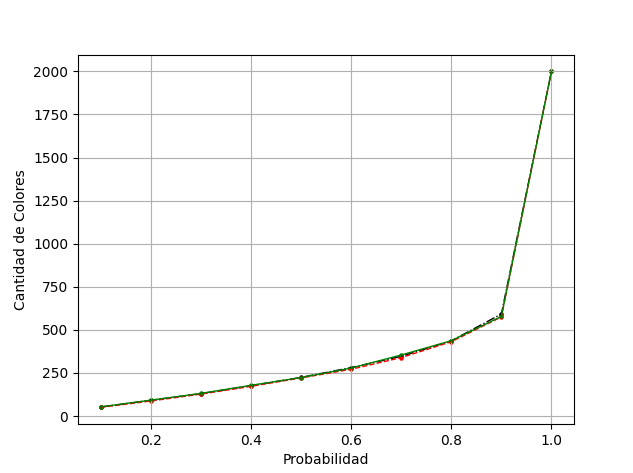
****

Figura 23. Comparativa de Colores Con grafos de 2000 vertices

Fuente: Propia

La comparación de colores utilizados de los algoritmos descritos en el capítulo anterior usando Grafos con 4000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 24.

| **P** | **X(G) DSatur** | **X(G) RFL** | **X(G) Jones-Passmann** |
| --- | --- | --- | --- |
| 0.1 | 89 | 85 | 89 |
| 0.2 | 161 | 155 | 159 |
| 0.3 | 234 | 230 | 235 |
| 0.4 | 315 | 307 | 312 |
| 0.5 | 399 | 398 | 401 |
| 0.6 | 505 | 499 | 505 |
| 0.7 | 627 | 627 | 626 |
| 0.8 | 804 | 793 | 802 |
| 0.9 | 1071 | 1069 | 1071 |
| 1 | 4000 | 4000 | 4000 |

Tabla 3. Comparativa de Colores Con grafos de 2000 vértices

Fuente: Propia

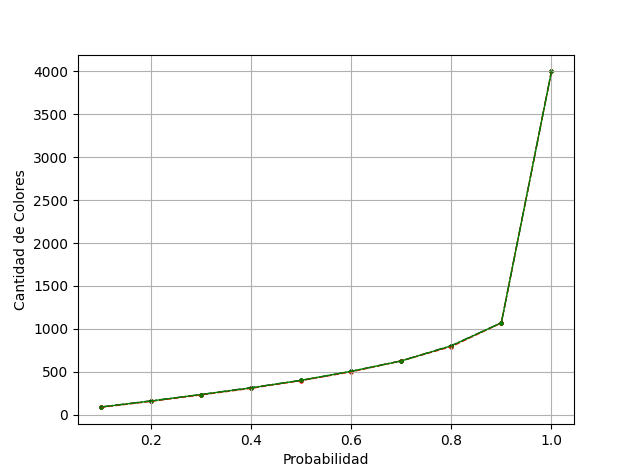


Figura 24. Comparativa de Colores Con grafos de 4000 vértices

Fuente: Propia

La comparación de tiempo de ejecución de los algoritmos descritos en el capítulo anterior usando Grafos con 100 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 25

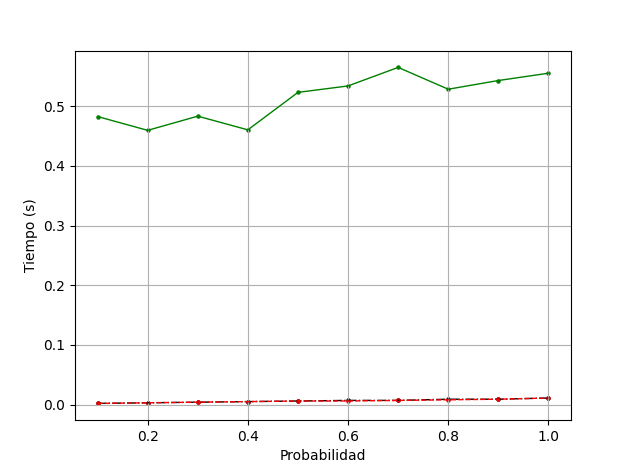


Figura 25. Comparativa de Tiempo de Ejecución Con grafos de 100 vértices

Fuente: Propia

La comparación de tiempo de ejecución de los algoritmos descritos en el capítulo anterior usando Grafos con 1000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 26

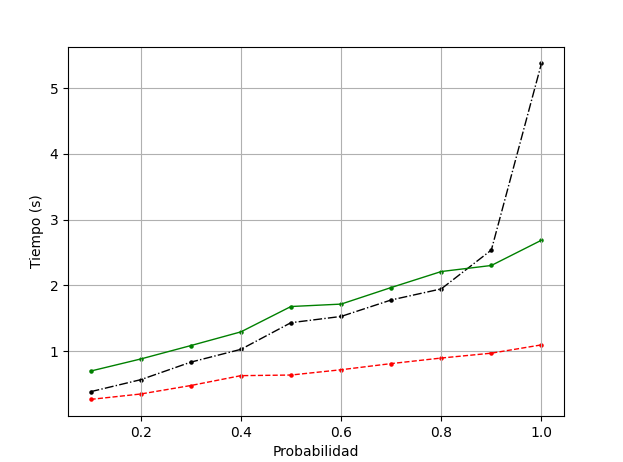


Figura 26. Comparativa de Tiempo de Ejecución Con grafos de 1000 vértices

Fuente: Propia

La comparación de tiempo de ejecución de los algoritmos descritos en el capítulo anterior usando Grafos con 2000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 27

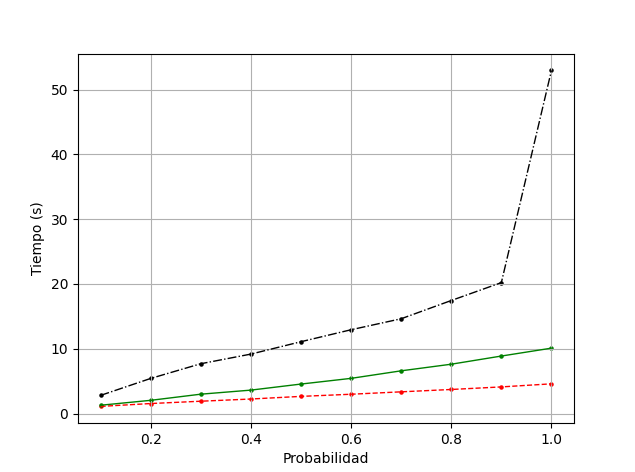


Figura 27. Comparativa de Tiempo de Ejecución Con grafos de 2000 vértices

Fuente: Propia

La comparación de tiempo de ejecución de los algoritmos descritos en el capítulo anterior usando Grafos con 4000 vértices y diferentes probabilidades de aristas que van de 0.1 a 1 podemos verlo en la Figura 28

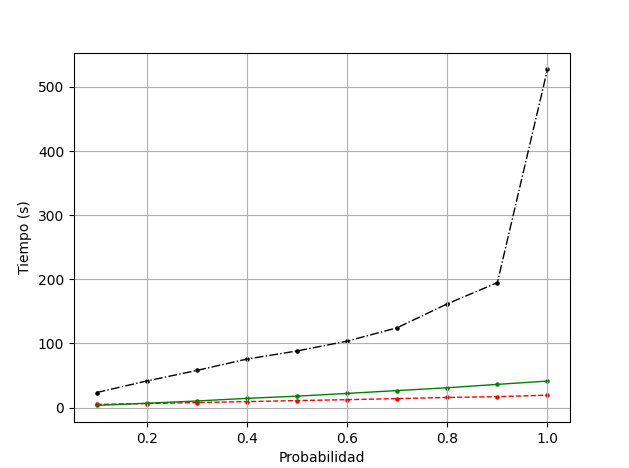


Figura 28. Comparativa de Tiempo de Ejecución Con grafos de 4000 vértices

Fuente: Propia

## **Demostración de la Hipótesis**

Para la demostración de la hipótesis se hará uso de los resultados obtenidos en las comparaciones, observando el número de colores y el tiempo empleado para colorear los diferentes casos de prueba.

En las comparaciones para todos los grafos aleatorios de 100, 1000 y 2000 vértices y generados con probabilidades de 0,1 a 1 para las aristas los algoritmos Heurísticos tuvieron un mejor desempeño en términos de colores utilizados Tabla 1, Tabla 2, Tabla 3 y Tabla 4.

En las comparaciones de tiempo para los grafos más pequeños, los de 100 aristas podemos notar que el algoritmo paralelo Jones-Plassmann tarda un poco más que los heurísticos Figura 25 pero a medida que tenemos que procesar grafos más grandes, 1000, 2000 y 4000 aristas este se torna mucho más veloz debido a que se utilizan más recursos para poder colorear los grafos Figura 26, Figura 27.

Podemos ver que el algoritmo RFL que basa su coloreado en el grado de los vértices es mucho más veloz que el algoritmo paralelo en términos de tiempo y colores utilizados, especialmente cuando la probabilidad de arista es mayor, por ejemplo cuando la probabilidad es 1 significa que el grafo es un clique ya que existe una arista de todos con todos, y como el algoritmo está basado en el grado de los vértices no será necesario preguntar por cada vecino de el vértice candidato como se describe en la implementación del algoritmo en el Capítulo 3.

Tenemos las poblaciones de 40 casos de prueba, para la cantidad de colores

H0: Nos indica que la cantidad de colores utilizados en los algoritmos Heurísticos es menor a la cantidad utilizada en los algoritmos paralelos

H1: Nos indica que la cantidad de colores utilizados son semejantes

n1 = 40

n2 = 40

Reemplazando los valores tenemos

Z = 0.0242

Si el nivel de significancia el del 5% entonces y Z = 1.65

Como |Z| < |Z| aceptamos la hipótesis nula

H0: Nos indica que el tiempo empleado para el coloreado en los algoritmos Paralelos es menor al tiempo de los algoritmos Heurísticos

H1: Nos indica que el tiempo en ambos casos es semejante

n1 = 40

n2 = 40

Reemplazando los valores tenemos

Z = -1.80

Si el nivel de significancia el del 5% entonces y Z = 1.65

Como |Z| < |Z| aceptamos la hipótesis nula

## **Conclusiones**

El algoritmo paralelo Jones-Plassmann tiene un mejor desempeño en términos de tiempo que el algoritmo DSatur cuando la cantidad de vértices y aristas es grande que viene a ser una mejora del algoritmo Greedy.

El algoritmo DSatur tiene un mejor desempeño en términos de colores utilizados que el algoritmo Jones-Plassman.

El algoritmo RFL (Recursive Large First) tiene mejor desempeño en términos de tiempo y colores que el algoritmo paralelo Jones-Plassmann en grafos con probabilidades de arista muy grandes, especialmente en los cliques.

# **Bibliografía**

BLUM A. 1991. Algorithms for Approximate Graph Coloring. Cambridge

BLUM A. 1994. New Approximation Algorithms for Graph Coloring. MIT

HALPERIN E., NATHANIEL R. y ZWICK U. 2018. Coloring k-coloreable graphs using relatively small paletter

DEMAINE E. y KARGER D. 2003. Approximation Algorithms

LI G. 2016. Solving the Grahp Coloring Problem With cooperative Local Search. karlsruhe institute of technology

REBOLLO I. y GRAÑA M. 2012. An empirical comparison of some approximate methods for Graph Coloring. University of the Basque Country

OLAV J. 2010. An analysis of tabu search for the graph coloring problem. Utrecht University

ASLAN M. y AKHAN N. 2016. A Performance Comparison of Graph Coloring Algorithms. Selcuk University

LEWIS. 2016. A Guide of Graph Coloring. Cardiff University

O’DONELL R.. 2008. Coloring 3-coloreable Graphs using SDP

PENA S. 2016. El Problema de Coloración de Grafos. Universidad de Vigo

HUSFELTD T. 2013. Graph Colouring Algorithms

J. R. ALLWRIGHT y R. BORDAWEKAR y P. D. CODDINGTON y K. DINCER and C. L. MARTIN. 1995. A Comparison of Parallel Graph Coloring Algorithms

PER NORMAN. 2014. Parallel Graph Colori