ISSN 1870-4069

Algoritmos híbridos paralelos para el problema de coloreado de grafos

Jessica G. Urrea Guerrero[1], Xiomara P. Zaldivar Colado[1], Rolando Menchaca-Méndez[2]

[1] Universidad Autónoma de Sinaloa, Culiacán, México
[2] Instituto Politécnico Nacional, Centro de Investigación en Computación, Ciudad de México, México
jess.urreag@gmail.com, xiomara.zaldivar@uas.edu.mx, rmen@cic.ipn.mx

Resumen El problema de coloreado de grafos pertenece a la clase de problemas *NP-Difícil*, por lo que a menos que la clase de problemas P sea igual a la clase NP, no existe un algoritmo polinomial que lo resuelva. A pesar de su elevada complejidad, el desarrollo de algoritmos para este tipo de problemas son de suma importancia por el hecho de que los problemas *NP-Difícil* aparecen en casi todas las áreas de las ciencias de la computación como aprendizaje de máquina, las redes de computadoras, los sistemas operativos, sistemas de información, entre otras. En este trabajo proponemos una implementación paralela, basada en CUDA, de un conjunto de variantes de un algoritmo evolutivo híbrido para el problema de coloreado de grafos. Dicho algoritmo se caracteriza por mezclar técnicas de cómputo evolutivo con heurísticas de búsqueda local, con el objetivo de encontrar soluciones cercanas a la óptima. Los resultados experimentales muestran que nuestra propuesta de implementación paralela es capaz de consistentemente encontrar buenas soluciones y al mismo tiempo reducir el tiempo de ejecución de los algoritmos evolutivos tradicionales.

Keywords: NP-difícil, coloreado de grafos, optimización combinatoria, metaheurísticas, algoritmos paralelos, CUDA.

Parallel Hybrid Algorithms for the Problem of Graph Coloring

Abstract. The graph coloring problem belongs to the problem class *NP-Hard*, so unless the problem class P is equal to the class NP, there is no polynomial algorithm to solve it . Despite its high complexity, the development of algorithms for these types of problems are of the utmost importance due to the fact that the problems *NP-Hard* appear in almost all areas of computer science as machine learning , computer networks, operating systems, information systems, among others. In this paper, we propose a parallel implementation, based on CUDA, of a set of variants of a hybrid evolutionary algorithm for the problem of coloring graphs. This

algorithm is characterized by mixing evolutionary computation techniques with local search heuristics, with the aim of finding near-optimal solutions. Experimental results show that our parallel implementation proposal is able to consistently find good solutions and at the same time reduce the execution time of traditional evolutionary algorithms.

Keywords: NP-hard, colored graphs, combinatorial optimization, metaheuristics, parallel algorithms, CUDA.

1. Introducción

La versión de decisión del problema de coloreado de grafos es uno de los 21 problemas que Richard Karp [1] demostró que pertenecen a la clase NP-Completa. Los problemas que pertenecen a esta clase se caracterizan por su complejidad, y por el hecho de que un algoritmo que resuelva cualquiera de ellos, podría ser usado para resolver a todos los demás problemas que pertenecen a dicha clase [1].

Existe otra clase de problemas conocida como NP-Difícil [2,3] que está compuesta por problemas aún más difíciles que los problemas NP-Completos. La complejidad extra radica en el hecho de que a diferencia de los problemas NP-Completos, los problemas NP-Difícil no se pueden verificar en tiempo polinomial.

Dado que el problema de coloreado de grafos en su versión de búsqueda pertenece a la clase de problemas NP-Difícil, es muy poco probable que exista un algoritmo de complejidad polinomial que lo resuelva. Por lo anterior, el problema ha sido abordado utilizando una gran variedad de técnicas algorítmicas entre las que se encuentran los algoritmos metaheurísticos. Una alternativa particularmente prometedora consiste en utilizar *algoritmos evolutivos híbridos* [4,5,6,9,10] que combinan algoritmos evolutivos tradicionales con esquemas de búsqueda local. En la literatura se muestra que este tipo de herramientas han dado buenos resultados [7], sin embargo, estos algoritmos suelen ser extremadamente costosos en términos de su tiempo de ejecución, por lo que se propone desarrollar una versión paralela que utilice la tecnología de tarjetas CUDA, con las cuales se espera una mejora significativa.

El resto del presente artículo se organiza de la siguiente manera. En la Sección 2 presentamos una muestra de los trabajos que proponen algoritmos para el problema de coloreado de gráfos. En la Sección 3 formulamos el problema de coloreados de grafos como un problema de minimización del número de aristas monocrómaticas del grafo. En la Sección 4 presentamos la especificación del algoritmo híbrido paralelo propuesto. Los resultados experimentales que caracterizan el desempeño del algoritmo propuesto en términos de la calidad de las soluciones y el tiempo de ejecución se presentan en la Sección 5 y finalmente, en la Sección 6 se presentan las conclusiones finales y los trabajos futuros.

2. Algoritmos metaheurísticos para coloreado de grafos

2.1. Búsqueda de vecindario variable

La búsqueda de vecindario variable o *variable neighborhood search* (VNS), se basa en la idea de un cambio dinámico del vecindario dentro de la búsqueda local. Procede por una metodología descendente que a través del vecindario hacia la exploración del óptimo local. Este método se mueve de la solución actual hacia una nueva, si es que implica una mejora en ella. De esta manera, la variable óptima se mantiene obteniendo vecindarios prometedores [11].

Avanthay [12] en 2003 introdujo una adaptación del método VNS para el problema de coloreado de grafos. Proponiendo una técnica que probó ser efectiva en determinar soluciones de alta calidad.

2.2. Algoritmos híbridos

Este tipo de algoritmos o técnicas son conocidos por combinar múltiples estrategias, ya sean del mismo tipo o diferentes para resolver un problema, también son llamados algoritmos *meméticos*.

Galinier [4] en 1999 realiza una propuesta utilizando un algoritmo híbrido para el problema de coloreado de grafos, combinando nuevas clases de operadores de cruza especializadas. Los resultados experimentales demostraron que el algoritmo de coloreado híbrido garantiza una mejora en los resultados dentro de su operador de búsqueda tabú, demostrando la importancia de la cruza dentro del proceso de la búsqueda. Además de realizar experimentos acerca del comportamiento del algoritmo referente a la evolución de la función de evaluación, así como la diversidad de la población. Al principio es evidente la necesidad de preservar la diversidad para que la búsqueda sea eficiente. Concluyendo con la afirmación del gran potencial que existe al utilizar este tipo de algoritmos para problemas combinatorios.

Mientras que en 2010 Zhipeng [13], propuso un algoritmo de metaheurística híbrida la cual integra búsqueda tabú dentro de un algoritmo evolutivo para resolver el problema de coloreado de grafos. En su trabajo destaca la importancia de la diversidad de los individuos. Utilizó una técnica *greedy* para cruzar a los individuos llamada *greedy partition crossover* (GPX), combinándolo con un operador de cruza multi-padre adaptativo, el cual demuestra obtener mejores resultados que con un operador de cruza de dos padres, además agregó una nueva función de "calificación de bondad"que considera la calidad y la diversidad de los individuos.

2.3. Algoritmos paralelos

El procesamiento paralelo es definido como ejecución concurrente o simultánea de instrucciones en una computadora. La motivación obvia de la paralelización es incrementar la eficiencia del procesamiento. Existen múltiples aplicaciones que requieren de grandes cantidades de procesamiento y análisis.

A pesar de que el uso del paralelismo es un tema reciente con tarjetas gráficas, en 1994 Lewandowski [14] introdujo el paralelismo en su trabajo por medio de una computadora paralela y un algoritmo híbrido para el coloreado de grafos, el cual combina una versión paralela del algoritmo Morgenstern *S-Impasse*, con una búsqueda exhaustiva *Branch and bound*. Lewandowski menciona que los resultados obtenidos pueden ser comparados con dos heurísticas secuenciales simples: el algoritmo de saturación de Brélaz (Dsatur) y el algoritmo de Leighton RFL (*Recursive Largest First*). En general el algoritmo propone que un gestionador de procesos empiece realizando la búsqueda exhaustiva paralelizada dentro de un grupo de procesadores, mientras que en otro grupo en paralelo se realice la heurística *S-Impasse*.

Por otro lado, en 2007 Szymon [15] propuso otro algoritmo paralelo con recocido simulado para el problema de coloreado de grafos, utilizando múltiples procesadores trabajando de forma concurrente en cadenas individuales con el objetivo de minimizar la función de costo y guardar la mejor solución encontrada. La coordinación del algoritmo es por medio de un modelo esclavo-maestro en el cual un procesador es es responsable de recolectar las soluciones, elegir la solución actual y distribuir la cantidad de unidades esclavas. Los experimentos revelaron que el rendimiento depende mucho en escoger adecuadamente el proceso de enfriamiento, así a la generación inicial con un coloreado que se ajuste al problema. Además de demuestra que el modelo utilizado dentro del algoritmo alcanza su punto óptimo, cuando el número de esclavos es relativamente pequeño.

3. Planteamiento del problema

En el contexto de la teoría de grafos, los objetos del problema de coloreado de grafos se representan por medio de un conjunto de vértices V, y las relaciones son modeladas por un conjunto de aristas E entre vértices. Para el problema de coloreado de grafos, dos nodos $u, v \in V$ tal que $(u, v) \in E$ no pueden pertenecer a la misma clase. Una forma de distinguir las clases es utilizando un conjunto de colores, y a la división en clases se le conoce como coloreado [16].

Formalmente, la versión de búsqueda del problema de coloreado de grafos se define por un grafo no dirigido $G = (V, E)$, un número de colores disponibles k y consiste en encontrar una función $f : V \rightarrow \{1, 2, 3, .., k\}$ tal que el tamaño del conjunto de *aristas monocromáticas* $|M|$ es minimizado, donde $M = \{(u, v) : (u, v) \in E \text{ y } f(u) = f(v)\}$, es decir, que el conjunto M contiene todos los pares de nodos adyacentes cuya función de asignación de color, sea la misma.

Existen dos variantes de este problema, en la primera el valor k (número de colores) es fijo y se desea minimizar el número de aristas monocromáticas. En la segunda versión, k es variable y se requiere encontrar el valor mínimo de k, tal que el número de aristas monocromáticas es igual a cero.

Debido a que el problema de coloreado de grafos pertenece a la clase NP-Difícil, a menos que la clase de problemas P sea igual a la NP, no es posible diseñar un algoritmo que se ejecute en tiempo polinomial capaz de resolverlo.

Por lo anterior, se han utilizado técnicas metaheurísticas [8,17,18] no polinomales para intentar encontrar buenas soluciones.

En el caso particular de los algoritmos híbridos, el principal cuello de botella radica en que cada individuo realiza una etapa de búsqueda local de manera secuencial. Para resolver este problema, se propone paralelizar la búsqueda local para que todos los individuos puedan progresar al mismo tiempo. Reducir el tiempo de ejecución es en particular importante cuando las instancias son muy grandes, o en casos en los que se requieren resultados en el menor tiempo posible, como lo es en los problemas de planificación.

4. Algoritmo evolutivo híbrido paralelo

En este trabajo se propone paralelizar en la plataforma CUDA variantes del algoritmo implementado por Galinier [4], por lo anterior, se concluyó que debido a que la memoria disponible dentro de la tarjeta CUDA es limitada, el implementar la técnica de búsqueda tabú no es una opción viable debido al consumo excesivo de recursos de cómputo que esta implica. Por lo tanto se propone el utilizar dos técnicas distintas para implementar la búsqueda estocástica y caracterizar el impacto que puede llegar a tener el tipo de búsqueda dentro del algoritmo evolutivo. Las técnicas propuestas son: *ascenso de colina* y *metrópolis*.

La parte que se desea paralelizar dentro del algoritmo genético híbrido es la búsqueda local. En el artículo de Galinier realiza la búsqueda de cada uno de los individuos dentro de un número fijo de generaciones, causando un retardo considerable en el tiempo. Mientras que en este trabajo se realiza la búsqueda local de cada individuo de forma paralela (asíncrona). Mediante el uso de los hilos CUDA, cada hilo toma un individuo realizando su búsqueda independiente, mientras que para evitar que el algoritmo avance, se utilizó una barrera CUDA, la cual garantiza que el algoritmo continúe una vez que todos los hilos hayan terminado de ejecutarse y avanzar a la siguiente generación. La población resultante será compuesta por óptimos locales que conformarán la nueva generación de individuos. Lo que respecta a la condición de parada, se utilizan dos aspectos: un número fijo de generaciones o cuando tres generaciones den los mismos resultados.

Se puede observar en la Figura 1 la metodología propuesta del algoritmo genético híbrido paralelo, para resolver el problema de coloreado de grafos.

Además del algoritmo genético junto con las búsquedas locales, se propone analizar las distintas jerarquías de memoria de la plataforma CUDA (global, compartida y local) en conjunto con variaciones en el tamaño de la población, número de bloques, e hilos, para ambas estrategias de búsqueda local, mediante experimentos exhaustivos para caracterizar el desempeño de cada algoritmo en términos del número cromático, y el tiempo total de ejecución. Para esto se utilizarán tanto experimentos estandarizados (*benchmarks*), como instancias generadas de manera aleatoria.

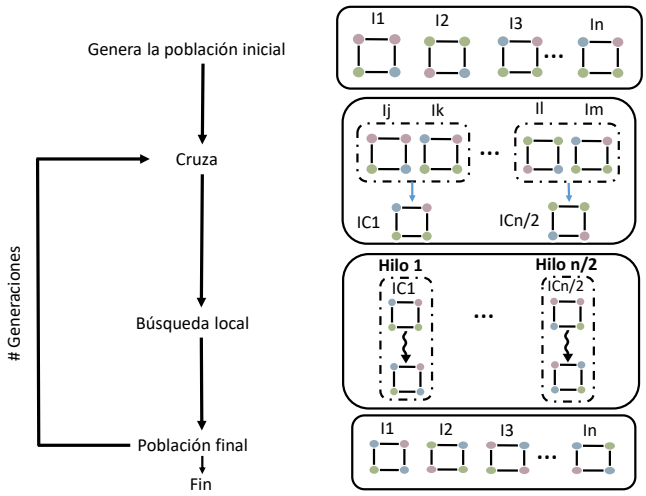

Fig. 1. Propuesta paralela para solucionar el problema de Coloreado de Grafos.

5. Resultados experimentales

Para la implementación de los experimentos se utilizó una computadora ASUS, con un procesador intel *core* i7 de 7_{ma} generación, la cual cuenta con una tarjeta gráfica *GEFORCE* GTX de NVIDIA, esta tarjeta contiene 768 núcleos CUDA, indispensables para la implementación en paralelo del problema de coloreado de grafos. Por otro lado, el algoritmo genético híbrido se codificó con *python* 3.6, junto con la extensión para CUDA en *python* llamada *numba*, con ella fue posible realizar la paralelización de *python* bajo ciertas restricciones, como lo fueron las estructuras aceptadas por numba dentro del *kernel*. Esto generó un cambio en la implementación del algoritmo paralelo respecto al secuencial. Lo que corresponde a las pruebas implementadas para el problema de coloreado de grafos, en este trabajo se realizó mediante el uso de dos tipos de experimentos: el primero fueron los *grafos* más conocidos para el problema de coloreado de grafos (*benchmarks*), y el segundo fueron *grafos aleatorios* generados con el modelo Gilbert G(n,p), en el que cada arista tiene una probabilidad independiente $0 < P < 1$ de aparecer en el grafo. Estos dos tipos de grafos fueron los utilizados en las pruebas bajo diferentes esquemas de configuraciones CUDA.

Las pruebas de los grafos aleatorios se realizaron en dos partes; la primera consistió en una comparación de las distintas configuraciones CUDA con ambas búsquedas locales, con los resultados obtenidos fue posible identificar a los representantes de cada búsqueda local. Una vez que se obtuvieron dichos representantes, en la segunda parte se analizó el comportamiento del problema dentro de diferentes esquemas, es decir, que se compararon los algoritmos greedy, el algoritmo genético híbrido secuencial y los representantes de las configuraciones

CUDA, todas ellas en base a los resultados de las pruebas con grafos aleatorios. Esto con el objetivo de analizar cual fue el algoritmo que brindó los mejores resultados.

5.1. Experimentos con grafos benchmarks

Los grafos más conocidos para el problema de coloreado (*benckmarks*) que fueron utilizados en las pruebas, son los que se muestran a continuación en la Tabla 1, así como los datos correspondientes de cada uno de ellos, como: el nombre del grafo, el número de nodos y el número cromático. Este último se refiere al menor número de colores encontrado hasta la fecha para colorear dichos grafos.

Tabla 1. Grafos *benchmarks* utilizados en pruebas.

Nombre	Num. Nodos	χ
myciel3	11	4
myciel4	23	5
$2-Fullins\_3$	52	5
Jean	80	10
Anna	138	11
DSJC250.5	250	28
DSJC500.5	500	48
$le450\_15c$	450	15
$le450\_25c$	450	25
$flat300\_28\_300$	300	31

El algoritmo genético híbrido paralelo diseñado para la plataforma CUDA, nos brindó la oportunidad de utilizar las diferentes jerarquías de memoria con las que 5cuenta la plataforma, dando la posibilidad de observar en que medida y como afecta la condición de competencia por los recursos, a medida que se va distanciando la memoria. Por lo anterior, se realizaron las pruebas correspondientes con cada búsqueda local tomando en cuenta el aspecto de la memoria, el número de hilos, y por último el número de bloques utilizados en el *kernel* CUDA, dependiendo del tamaño de la población (128 ó 256). En la Tabla 2 se observa como fueron organizadas estas configuraciones.

Tabla 2. Configuraciones del AGH-CUDA con respecto al tamaño de la población.

Población 128	Población 256
2 Bloques con 32 hilos	2 Bloques con 64 hilos
1 Bloque con 64 hilos	1 Bloque con 128 hilos

El motivo de utilizar diferentes tamaños de número de bloques y número de hilos, fue que en la literatura se menciona que la paralelización en CUDA tiende a dar buenos resultados si los hilos son múltiples de 32 en cada bloque, debido a

los warps (hilos en ejecución en el procesador). Experimentar con esto permitió analizar los diferentes resultados y el comportamiento del algoritmo. Por otro lado, lo que respecta al número de colores utilizado en el algoritmo paralelo, fue el mismo que con los otros algoritmos (número cromático χ), dando como resultado, la mejor configuración que fue capaz de encontrar, con el mínimo número de aristas monocromáticas.

Con el objetivo de poder realizar una comparación a profundidad sobre cada algoritmo, en las Tablas 3 y 4 se encuentran los resultados de los algoritmos *greedy*, el algoritmo genético híbrido secuencial, y los resultados del algoritmo genético híbrido paralelo con cada una de las diferentes configuraciones CUDA y búsquedas locales.

Tabla 3. Comparación de resultados de todas las diferentes estrategias para resolver el PCG.

	myciel3		myciel4		2-Fullin-3		Jean		Anna	
	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM
smallest last	0.00082774	0	0.00141325	0	0.00390942	0	0.00752094	0	.01270669	0
Independent set	0.0015182	0	0.00352569	0	0.01781993	2.5	0.04130497	3.2	0.1002692	2.8
Secuencial-Metro-128	0.01187162	0	0.03661599	0	0.12785444	0	0.23786497	0	0.74481041	0
Secuencial-Metro-256	0.02434862	0	0.07471681	0	0.26140914	0	0.47922232	0	1.51863251	0
Secuencial-AC-128	0.0118885	0	0.03630571	0	0.13015926	0	0.24174876	0	0.757967	0
Secuencial-AC-256	0.02403321	0	0.07381628	0	0.26248481	0	0.47692354	0	1.50307961	0
Global,128,2,32 AC	0.01186512	0	0.03740296	0	0.13064239	0	0.24175212	0	0.75179014	0
Global,128,1,64 AC	0.01216662	0	0.03750648	0	0.12895861	0	0.23986328	0	0.75629468	0
Global,256,2,64 AC	0.02403724	0	0.07341001	0	0.2646903	0	0.48102677	0	1.50167754	0
Global,256,1,128 AC	0.02499306	0	0.07186005	0	0.24993515	0	0.45770319	0	1.4683805	0
Compartido,128,2,32 AC	0.01167412	0	0.03760276	0	0.13293967	0	0.24412897	0	0.75360692	0
Compartido,128,1,64 AC	0.01166618	0	0.03751254	0	0.13184466	0	0.24494262	0	0.74899721	0
Compartido,256,2,64 AC	0.02343147	0	0.07029381	0	0.25618696	0	0.45926509	0	1.44184403	0
Compartido,256,1,128 AC	0.02343309	0	0.06873243	0	0.24993711	0	0.48649487	0	1.4972954	0
Local,128,2,32 AC	0.01405582	0	0.03124785	0	0.12654028	0	0.22807615	0	0.7185735	0
Local,128,1,64 AC	0.01249394	0	0.03593137	0	0.13434041	0	0.23275118	0	0.72014003	0
Local,256,2,64 AC	0.02394104	0	0.07450573	0	0.25841138	0	0.47673178	0	1.51734428	0
Local,256,1,128 AC	0.02343206	0	0.07030921	0	0.25619063	0	0.46083097	0	1.52664618	0
Compartido,128,1,64 AC	0.01166618	0	0.03751254	0	0.13184466	0	0.24494262	0	0.74899721	0
Global,128,2,32 Metro	0.0124676	0	0.03668928	0	0.1294549	0	0.24205575	0	0.76824338	0
Global,128,1,64 Metro	0.01208477	0	0.03680036	0	0.13353307	0	0.24884129	0	0.75777636	0
Global,256,2,64 Metro	0.02383668	0	0.0747021	0	0.25820274	0	0.47802968	0	1.49112074	0
Global,256,1,128 Metro	0.02343116	0	0.06561277	0	0.2483758	0	0.5311223	0	1.44964225	0
Compartido,128,2,32 Metro	0.01245883	0	0.03904974	0	0.12496932	0	0.23276341	0	0.72482629	0
Compartido,128,1,64 Metro	0.01217918	0	0.03811331	0	0.13223476	0	0.24504087	0	0.76275992	0
Compartido,256,2,64 Metro	0.0241441	0	0.07400806	0	0.26549172	0	0.47712281	0	1.51675532	0
Compartido,256,1,128 Metro	0.02424293	0	0.07539182	0	0.25961161	0	0.47422667	0	1.44497166	0
Local,128,2,32 Metro	0.01562073	0	0.03428657	0	0.12965605	0	0.23118894	0	0.72170324	0
Local,128,1,64 Metro	0.01249373	0	0.03593302	0	0.12805967	0	0.23275688	0	0.72170436	0
Local,256,2,64 Metro	0.02342687	0	0.06873183	0	0.25306246	0	0.45769961	0	1.43717144	0
Local,256,1,128 Metro	0.02414665	0	0.07439873	0	0.27027705	0	0.47712581	0	1.5648118	0

Gracias a las Tablas 3 y 4, es posible observar desde un panorama general cual fue el algoritmo que brindó los mejores resultados en ambos aspectos de evaluación (tiempo y NAM). El algoritmo genético secuencial con la búsqueda local de *gradient descent* y una población de 128, es el que demuestra sobresalir en ambos aspectos, se puede observar que hay una pequeña mejora en el número de aristas cuando la población aumenta a 256, pero los resultados en el NAM no representa una mejora en proporción al tamaño de la población, además de ser ejecutado en el doble de tiempo.

Tabla 4. Comparación de resultados de todas las diferentes estrategias para resolver el PCG (cont.).

	DSJC250.5		DSJC500.5		le450_15c		le450_25c		flat300_28	
	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM	Tiempo	NAM
smallest last	0.144186	6042.8	0.57685297	25722.2	0.15067627	9309.2	0.16575511	3342.9	0.19865119	8394.2
Independent set	0.67416017	3264.3	4.24918582	13733.8	1.69596386	7320.3	2.27669196	3277	1.28648229	4735.8
Secuencial-Metro-128	34.0076456	127.6	211.857908	367.6	73.8724133	436.5	71.5198407	84.2	51.0609503	142.3
Secuencial-Metro-256	68.2377067	123.4	424.012502	365.3	144.784495	431.6	139.28404	81.8	97.6175751	138.6
Secuencial-AC-128	33.8509865	109	207.811258	346.7	73.6341539	412.1	73.747267	65.7	53.1880474	124.8
Secuencial-AC-256	70.7350266	107	443.754476	344.4	153.223014	411.5	145.053515	64.5	107.512961	123.7
Global,128,2,32 AC	34.1717166	126.5	213.940944	367.6	72.9817449	433.4	73.1647449	83.3	51.7450323	139.3
Global,128,1,64 AC	34.3502736	126.5	214.641463	365.1	73.133162	437.3	70.9769254	83.7	51.8578098	140.4
Global,256,2,64 AC	64.92534	123.9	413.151073	362.5	139.837441	437.2	139.158007	81.4	99.3718972	138.8
Global,256,1,128 AC	64.9549603	124	411.359283	364.6	139.243994	431.9	138.284741	81.6	97.7830749	137.3
Compartido,128,2,32 AC	34.288105	126	216.863896	363.2	74.5060439	438	72.5581412	82.9	56.5781646	138.3
Compartido,128,1,64 AC	34.5847636	124.2	213.416294	365.5	73.0850838	436.6	72.7328963	83.4	52.5269308	140.1
Compartido,256,2,64 AC	68.1067633	124	427.984928	365	144.212103	433	138.417273	82.6	98.2642722	138.1
Compartido,256,1,128 AC	67.4484771	121.7	429.068003	361.4	139.036223	434.1	142.060453	82.1	100.116905	136.3
Local,128,2,32 AC	32.9921419	125	207.696505	364.8	70.207198	434.6	70.5585664	82.7	50.3506656	142.5
Local,128,1,64 AC	33.0202579	125.2	204.747229	365.8	70.6045974	436.5	72.2004636	82	49.8178983	141.3
Local,256,2,64 AC	67.0653446	123.4	430.594769	362.3	145.049938	431.2	148.267676	82.8	101.732809	138.4
Local,256,1,128 AC	67.0821071	122.8	412.174914	363.9	141.39629	431.6	137.739474	82	98.5904665	138
Compartido,128,1,64 AC	34.5847636	124.2	213.416294	365.5	73.0850838	436.6	72.7328963	83.4	52.5269308	140.1
Global,128,2,32	33.7784859	126.2	211.159676	366.7	71.5544534	434.9	72.8595934	82.3	51.8439337	141.2
Global,128,1,64	33.7664801	123.7	214.284681	365.6	70.9570144	437.9	71.6842514	82.8	52.8520553	140.2
Global,256,2,64	66.5663851	123.5	421.985079	363.5	145.053013	437.5	149.404148	82.5	101.116071	140
Global,256,1,128	65.5345762	123.7	411.801365	366	138.462901	433.1	139.13256	83.6	102.435103	138.6
Compartido,128,2,32	32.2797088	125.5	205.553325	367.1	69.78218	434.8	69.8541036	81.8	54.7158808	139.6
Compartido,128,1,64	33.6483535	125.1	213.496964	366.8	71.8928846	435.7	71.8097772	83.2	51.7925991	141
Compartido,256,2,64	66.5423517	122	430.630632	360.9	141.605158	432.6	142.871373	82.6	100.438336	138.9
Compartido,256,1,128	63.3256643	125.3	421.142926	362.3	142.19439	437	141.475995	80	100.910794	140.2
Local,128,2,32	32.3939242	127	206.529582	363.1	70.2711495	434.9	69.1058066	82.8	48.9696568	139.9
Local,128,1,64	32.3908168	127	207.107711	364.2	68.9168411	439.2	69.597785	82.9	48.9541056	141.6
Local,256,2,64	64.4675533	123.4	415.066547	362	142.391495	432.7	136.831269	80.7	99.8263281	138.3
Local,256,1,128	67.2900504	122.6	444.546514	361.6	144.741161	436.1	141.839243	82.2	99.9947874	139

Respecto a los resultados con el algoritmo genético paralelo, es posible observar que la configuración más cercana al algoritmo secuencial, es aquel que utiliza memoria local, con resultados que muestran una ligera mejora en el tiempo, pero con un mayor número de aristas monocromáticas. Por lo que se puede suponer que la memoria local es que es muy pequeña, y por lo tanto, es muy probable que realice varias copias de diferentes datos de la matriz, generando una competencia por los recursos al pasar por el bus, es decir que al existir esa competencia, esto hace que los distintos hilos se formen en el bus para obtener los datos a la memoria global, perdiendo por completo la paralelización del algoritmo.

Con el objetivo de medir la estabilidad de los algoritmos, se realizó el calculo de la desviación estándar de los 10 resultados obtenidos con cada una de las diferentes semillas, además del calculo de la desviación estándar de cada grafo, utilizando múltiples semillas.

5.2. Experimentos con grafos aleatorios

Además de los grafos *benchmark* también fueron creados 160 grafos aleatorios distintos. Para crear dichos grafos se propuso utilizar el modelo Gilbert G(n,p), el cual funciona por medio del número de nodos del grafo (n) y una probabilidad (p), en la que cada arista tiene una probabilidad de que exista o no dentro del

grafo. En la Tabla 5 muestra el número de grafos que fueron creados, cada uno con probabilidades de 0.4 a 0.7 con variaciones en el número de nodos:

Tabla 5. Grafos aleatorios utilizados en pruebas.

Grafo	Probabilidad	Número de nodos
10 Grafos	0.4 - 0.7	20
10 Grafos	0.4 - 0.7	40
10 Grafos	0.4 - 0.7	60
10 Grafos	0.4 - 0.7	80
10 Grafos	0.4 - 0.7	100

Así como en los grafos *benchmark*, en los grafos aleatorios fueron utilizados las mismas estrategias de búsqueda local (*smallest last* e *independent set*), además de los mismos criterios de evaluación: el tiempo en segundos, y el número de aristas monocromáticas, éste último como con los grafos *benchmarks*, una vez que se fija el número de colores (k) en el algoritmo genético híbrido, los nodos cuyas etiquetas (colores) sean mayores a k, son sumados como aristas monocromáticas.

Los experimentos con grafos aleatorios se dividió en dos partes; la primera consistió en comparar las distintas configuraciones CUDA, para obtener representantes para cada una de las estrategias de búsqueda local. Para realizarlo, se tomó los resultados de los grafos de 20, 60 y 100 nodos, los resultados ganadores fueron aquellas que ganaban en dos de tres casos para cada estrategia, además de que fue dividida en las dos categorías de evaluación: tiempo y número de aristas monocromáticas. En la Tabla 6 muestra a los representantes ganadores.

Tabla 6. Representantes de las configuraciones CUDA.

Ascenso de colina		**Metrópolis**	
Criterio	**Configuración CUDA**	**Criterio**	**Configuración CUDA**
Tiempo	P128,1B,64H memoria local	*Tiempo*	P128,2B,32H memoria local
NAM	P256,1B,128H memoria global	*NAM*	P256,1B,128H memoria global

Una vez que se obtuvieron a los representantes de las configuraciones CUDA para cada búsqueda local, fue posible avanzar a la segunda parte de la experimentación, en la cual, se crearon gráficas que comparan los resultados del algoritmo *greedy*, secuencial, y los representantes de las configuraciones CUDA.

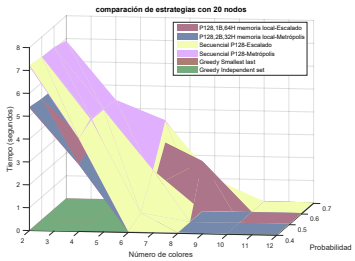

Fig. 2. Gráfica comparativa del tiempo de los algoritmos con 20 nodos.

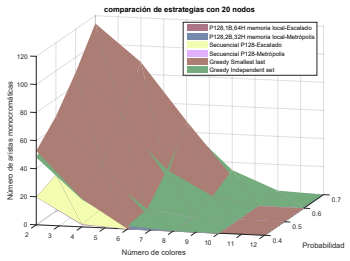

Fig. 3. Gráfica comparativa del NAM de los algoritmos con 20 nodos.

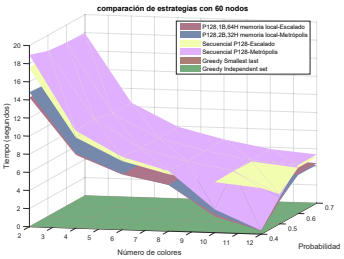

Fig. 4. Gráfica comparativa del tiempo de los algoritmos con 60 nodos.

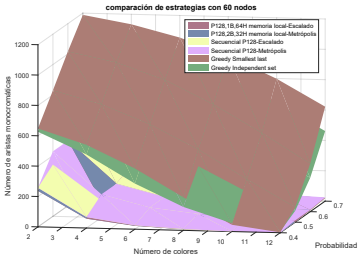

Fig. 5. Gráfica comparativa del NAM de los algoritmos con 60 nodos.

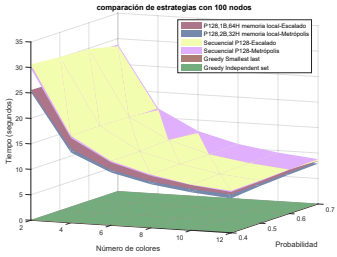

Fig. 6. Gráfica comparativa del tiempo de los algoritmos con 100 nodos.

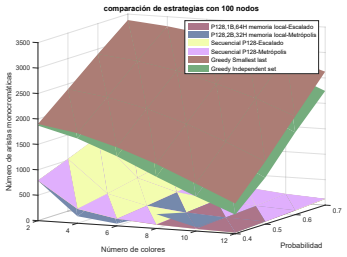

Fig. 7. Gráfica comparativa del NAM de los algoritmos con 100 nodos.

6. Conclusiones y trabajos a futuro

6.1. Conclusiones

Los resultados demostraron que con grafos *benchmarks*, el algoritmo secuencial es el que brinda el menor tiempo y menor número de aristas monocromáticas. La hipótesis resultante del análisis de estos resultados, es que existe un bajo

desempeño en la calidad de números aleatorios paralelos, a comparación con los generadores de números aleatorios secuenciales, por lo cual esto afecta la calidad de resultados de las estrategias de búsqueda local dentro del espacio de soluciones. Mientras que en los experimentos con grafos aleatorios, en la mayoría de los casos existe una clara superioridad de los algoritmos genéticos híbridos paralelos, demostrando la hipótesis inicial, que el paralelizar los algoritmos reduce el tiempo de ejecución.

Además se demuestra que los algoritmos evolutivos híbridos paralelos son capaces, en general, de encontrar mejores soluciones que los algoritmos voraces. Esto se debe, principalmente, a la capacidad de los algoritmos evolutivos de realizar una exploración más extensa del espacio de posibles soluciones, y de explotar las diferentes regiones del espacio de soluciones a partir de las búsquedas locales. Los algoritmos voraces, por su parte, tienden a quedar atrapados en óptimos locales que pueden ser arbitrariamente malos.

Con el propósito de caracterizar el impacto de incrementar el número de hilos, el tiempo de ejecución del algoritmo, así como la calidad de las soluciones al incrementar el tamaño de la población, se realizó un conjunto de experimentos donde varió el número de hilos desde 64 hasta 128.

Además de caracterizar el número de hilos, se analizó el impacto de utilizar diferentes esquemas de memoria de la tarjeta gráfica en el tiempo de ejecución del algoritmo. Los resultados muestran que la configuración con una *población de 256,1 bloque, 128 hilos con memoria global y metrópolis* fue la que presentó un mejor desempeño. Esto se debe principalmente a que una población grande asegura contener más puntos dentro del espacio de búsqueda, mientras que la búsqueda local metrópolis da la oportunidad de repartir los puntos, además de que la nobleza que brinda esta búsqueda permite no quedar estancados en óptimos locales, como encontrar nuevos caminos a posibles soluciones potenciales.

Es importante resaltar que debido a las condiciones de competencia generadas dependiendo de la distancia y limitación de la memoria, así como el uso constante de los buses de comunicación, no permitió observar una disminución del tiempo de ejecución del algoritmo propuesto de manera proporcional al número de hilos. Por esto último, es importante seguir investigando sobre nuevas implementaciones que hagan un uso más eficiente de la arquitectura CUDA con el fin de mejorar, aún más, el tiempo de ejecución.

6.2. Trabajo a futuro

Como trabajo a futuro se pretende aplicar la herramienta de software desarrollada para resolver problemas reales que pueden ser modelados como el problema de coloreado de grafos. En particular problemas en el contexto de planificación de tareas [19], planificación de grupos de trabajo [20], diseño de compiladores [21], reconocimiento de patrones [22], redes de computadoras [23,24].

Diseñar, implementar y caracterizar experimentalmente nuevas estrategias de búsqueda local específicamente diseñadas para plataformas de hardware con memoria restringida como las tarjetas CUDA. En este mismo sentido, es importante seguir investigando sobre algoritmos concurrentes que reduzcan las condiciones

de competencia que se presentan cuando varios núcleos de ejecución intentan acceder a la memoria compartida.

Extender la investigación reportada en la presente a otros problemas de optimización combinatoria que pertenezcan a la clase NP-difícil como cobertura de vértices, conjunto independiente, conjunto dominante, ciclo Hamiltoneano, entre otros.

Referencias

1. Karp, R.M., et al.: Reducibility among combinatorial problems. Complexity of computer computations, Springer, pp. 85–103 (1972)
2. Cormen, T.H.: Introduction to algorithms. MIT press (2009)
3. Kleinberg, J., Tardos, E.: Algorithm design. Pearson Education (2006)
4. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. Journal of combinatorial optimization 3(4), 379–397 (1999)
5. Glover, F., Kelly, J.P., Laguna, M.: Genetic algorithms and tabu search: hybrids for optimization. Computers & Operations Research 22(1), 111–134 (1995)
6. Chiarandini, M., Stützle, T.: An application of iterated local search to graph coloring problem. In: Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, pp. 112–125 (2002)
7. Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation 9(5), 474–488 (2005)
8. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR) 35(3), 268–308 (2003)
9. Beasley, D., Bull, D.R., Martin, R.R.: An overview of genetic algorithms: Part 1, fundamentals. University computing 15(2), 56–69 (1993)
10. Mühlenbein, H.: Parallel genetic algorithms, population genetics and combinatorial optimization. In: Workshop on Parallel Processing: Logic, Organization, and Technology, pp. 398–406. Springer, Berlin, Heidelberg (1989)
11. Baghel, M., Agrawal, S., Silakari, S.: Survey of metaheuristic algorithms for combinatorial optimization. International Journal of Computer Applications 58(19) (2012)
12. Avanthay, C., Hertz, A., Zufferey, N.: A Variable Neighborhood Search for Graph Coloring. Eur. J. Oper. Res., vol. 151, pp. 379–388 (2003)
13. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. European Journal of Operational Research 203(1), 241–250 (2010)
14. Lewandowski, G., Condon, A.: Experiments with parallel graph coloring heuristics and applications of graph coloring. DIMACS Series in Discrete Mathematics (1994)
15. Łukasik, S., Kokosiński, Z., Świętoń, G.: Parallel simulated annealing algorithm for graph coloring problem. In: International Conference on Parallel Processing and Applied Mathematics. Springer, Berlin, Heidelberg, pp. 229–238 (2007)
16. Jensen, T.R., Toft, B.: Graph coloring problems (Vol. 39). John Wiley & Sons (2011)
17. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: International Work-Conference on the Interplay Between Natural and Artificial Computation, Springer, Berlin, Heidelberg, pp. 41–53 (2005)

18. Voß, S., Martello, S., Osman, I.H., Roucairol, C. (eds.): Meta-heuristics: Advances and trends in local search paradigms for optimization. Springer Science & Business Media (2012)
19. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards 84(6), 489–506 (1979)
20. Gamache, M., Hertz, A., Ouellet, J.O.: A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. Computers and operations research 34(8), 2384–2395 (2007)
21. Chaitin, G.J.: Register allocation and spilling via graph coloring. ACM Sigplan Notices 17(6) (1982)
22. Conte, D., et al.: Thirty years of graph matching in pattern recognition. International journal of pattern recognition and artificial intelligence 18(03), 265–298 (2004)
23. Zhu, X., Linglong, D., Zhaocheng. W.: Graph coloring based pilot allocation to mitigate pilot contamination for multi-cell massive MIMO systems. IEEE Communications Letters 19(10), 1842–1845 (2015)
24. Checco, A., Leith, D.J.: Fast, Responsive Decentralized Graph Coloring. IEEE/ACM Transactions on Networking 25(6), 3628–3640 (2017)