UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE FRAMEWORK DE EVALUACIÓN PARA LOS PROBLEMAS DEL MÁXIMO CONJUNTO INDEPENDIENTE Y MÁXIMO CORTE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

DANIEL ALEJANDRO BAEZ MIRANDA

PROFESOR GUÍA:
ANDRÉS ABELIUK KIMELMAN

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
SEBASTIÁN FERRADA ALIAGA

SANTIAGO DE CHILE
2024

Resumen

Las redes neuronales son cada vez más populares. La creación de redes neuronales que funcionan sobre grafos o GNNs (Graph Neural Network) se ha presentado como una nueva oportunidad de abordar la amplia gama de problemas que se pueden modelar con grafos, problemas que van desde el comercio virtual hasta la bioinformática. El área de optimización combinatorial ha sido una en la que ha habido interés particularmente debido a sus numerosos problemas importantes con muchas aplicaciones que son NP y difíciles de aproximar. Sin embargo, estos problemas tienen una larga historia de estudio, a lo largo de los años múltiples algoritmos se han desarrollado para tratar de obtener buenos resultados en estos problemas. También se han tratado de buscar instancias difíciles que puedan representar bien los desafíos que conllevan estos problemas. El problema que existe es que al momento de querer desarrollar una GNN es muy fácil pasar por alto alguno de los algoritmos importantes o no encontrar los casos de prueba indicados con los que evaluar la red, si un algoritmo nuevo no se evalúa apropiadamente se pueden llegar a conclusiones erróneas con respecto a su desempeño, es importante tener claro el estado del problema para poder desarrollar un algoritmo de forma correcta. Para solucionar este problema, este trabajo plantea un framework para organizar la evaluación de algoritmos y generar un benchmark apropiado para estos problemas. Además, se crean dos instancias del framework sobre los problemas de Máximo Conjunto Independiente y Máximo Corte, con esto se deja disponible una herramienta que permite comparar el desempeño de nuevos algoritmos para estos problemas con algoritmos clásicos que funcionan al estado del arte sobre instancias relevantes del problema.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Los grafos son una de las estructuras de datos más populares en computación. Un grafo es un conjunto de objetos llamados nodos conectados por enlaces llamados aristas [71]. Gracias a su simpleza, estos se pueden usar para modelar distintos problemas en áreas como el comercio virtual [84, 24], las redes sociales [75, 35] y la bioinformática [83, 21]. Con una larga historia de estudio en teoría de grafos, muchos problemas han sido estudiados ampliamente por cientos de años. Sin embargo, no porque hayan sido estudiados por tanto tiempo implica que hayamos encontrado formas eficientes de resolverlos. Una categoría de problemas que surgen frecuentemente en el estudio de grafos son los problemas de optimización combinatorial, esta área trabaja problemas donde se toman una gran cantidad de decisiones del tipo sí o no y cada conjunto de decisiones tiene un valor asociado, ya sea de costo o ganancia, y la idea es optimizar ese valor [68]. El problema del vendedor ambulante y el máximo conjunto independiente son ejemplos de problemas de esta categoría. A pesar de que existen problemas de optimización combinatorial para los cuales se conocen algoritmos polinomiales, hay bastantes problemas que están en NP debido a que a medida que va creciendo el tamaño de los grafos, el espacio de búsqueda sobre las decisiones crece de forma exponencial, esto hace que la búsqueda de soluciones exactas sea inviable y solo se puedan desarrollar métodos aproximados para encontrar una solución. De esta área han salido muchos problemas de alto interés tanto en la comunidad científica como en la industria debido a sus variadas aplicaciones [76].

Una forma de resolver estos problemas es usar redes neuronales artificiales, un modelo computacional para resolver problemas basado en un conjunto de unidades llamadas neuronas artificiales. Estas neuronas están conectadas a través de señales que se envían entre sí. Hay un grupo de neuronas iniciales que reciben el problema traducido como señales y un grupo de neuronas finales que entregan una solución en forma de señales. La fortaleza de este modelo es que las señales se van ajustando con ejemplos del problema, cambian sus parámetros con tal de acercarse lo más posible a la solución [7]. El uso de redes neuronales para resolver todo tipo de problemas va cada vez más en aumento con el incremento de la capacidad de cómputo que avanza año tras año y la disponibilidad de cada vez más información en la era digital. Para el trabajo de los grafos con redes neuronales se ha desarrollado el área de las redes neuronales de grafos (GNN por Graph Neural Network), redes neuronales específicamente diseñadas para trabajar con grafos. Las redes neuronales más típicas tienden a aprovecharse

de la estructura y distribución de los datos para conseguir más información, esto se puede ver por ejemplo en las imágenes, donde las redes neuronales convolucionales asocian información de sectores cercanos en las imágenes con el objetivo de no estudiar solo los píxeles de la imagen por separado, sino que también grupos de píxeles y su relación con su entorno [72]. En los grafos es difícil hacer esto ya que no existe un orden en las conexiones de un nodo, algo conocido como invarianza de permutaciones [80], por lo tanto las GNNs tienen que funcionar de una forma muy distinta para poder trabajar con estas estructuras. Se trabaja sobre los vecindarios de los nodos y las características de estos vecindarios para conseguir información que pueda ser útil para resolver el problema. Las GNN han generado gran interés en la comunidad científica debido a su alto potencial; la versatilidad de los grafos para modelar problemas hace que las GNNs sean capaces de resolver múltiples desafíos en muchas áreas.

Recientemente, el éxito de las redes neuronales han incrementado considerablemente su popularidad para tratar de resolver estos problemas [5, 52]. Sin embargo, debido a la larga historia de estos problemas, a pesar de que muchos de ellos sean NP [56] y también sean difíciles de aproximar [39], a lo largo del tiempo se han desarrollado muchas soluciones que pueden no ser óptimas pero funcionan de una u otra manera [73]. Entonces a la hora de desarrollar una GNN para resolver uno de estos problemas no basta con compararlas con las distintas GNNs que hay, es importante también que se compare con los mejores algoritmos que se han desarrollado hasta ahora en base a distintas heurísticas y métodos alternativos. En un reciente artículo de Angelini et al [1] se dejó en evidencia este problema cuando se demostró que el trabajo reciente de Schuetz et al. [68] que trataba de resolver varios problemas de optimización combinatorial con GNNs y afirmaba que su modelo funcionaba a la par del estado del arte en realidad era peor tanto en tiempo como en calidad de resultados con un algoritmo greedy, uno de los algoritmos más básicos que se pueden desarrollar. Para resolver este problema es importante que en las investigaciones se compare el desempeño de las GNNs con algoritmos clásicos a través de distintos casos de prueba que permitan dilucidar sus debilidades.

Con este problema en mente, se introduce el tema de la memoria: desarrollar un framework que funcione como benchmark para estos problemas, que permita comparar distintos algoritmos clásicos con GNNs en cuanto a tiempo y resultados con una base de datos de distintos grafos reales y sintéticos con los cuales se pueda explorar como funcionan los algoritmos con casos auténticos y en casos de borde. Además se proveerá una serie de implementaciones de algoritmos que resuelvan los problemas sin el uso de redes neuronales.

Los problemas que se decidieron abordar en este trabajo son el problema del Máximo Conjunto Independiente (MIS por Maximal Independant Set) y el problema del Máximo Corte (MaxCut). Se escogieron estos problemas específicamente porque son los problemas abordados en el trabajo de Schuetz et al. [68], el cual fue mencionado anteriormente como el trabajo que afirmaba estar a la par del estado del arte pero resultaba no estarlo. Con estas implementaciones se espera poder usar el framework para poder identificar como se desempeña con distintos tipos de entrada.

Además, estos dos problemas son importantes en el área de optimización combinatorial, tienen una larga historia en la cual se han desarrollado una gran cantidad de heurísticas para tratar de acercarse a soluciones mejores, una larga historia que es difícil de estudiar mientras se desarrolla una GNN, por lo tanto, sería ideal tener esto desde un principio para poder

compararlo sin tener que hacer una larga investigación al respecto.

Los objetivos a lo largo de este trabajo son:

1. Recopilar información sobre los algoritmos clásicos de resolución del Máximo Conjunto Independiente y el Máximo Corte para encontrar un grupo de algoritmos que representen al estado del arte tanto en eficiencia como calidad de resultados.
2. Generar y compilar dataset de grafos para la evaluación del Máximo Conjunto Independiente y el Máximo Corte tomando en cuenta casos reales y casos de borde.
3. Implementar y evaluar los algoritmos clásicos del Máximo Conjunto Independiente y el Máximo Corte para comprobar cómo se comportan con distintos casos.
4. Implementar y evaluar los distintos GNNs que ya existen para el Máximo Conjunto Independiente y el Máximo Corte para tener una base de comparación para comprobar lo que tenemos.
5. Organizar todas las partes para generar una interfaz de usuario que permita comparar su propio algoritmo con los algoritmos recopilados anteriormente.

Capítulo 2

Estado del Arte

2.1. Grafos

Un grafo $G = (V, E)$ se define como un conjunto de elementos V llamados vértices y un conjunto de conexiones E llamadas aristas, cada arista $e = (v_1, v_2) \in E$ conecta a dos vértices $v_1, v_2 \in V$ [77]. Un grafo se llama simple cuando todas las aristas son distintas entre sí y siempre conectan dos elementos distintos de V.

Un grafo también puede tener peso en sus aristas, esto significa que toda arista $e = (v_1, v_2, w)$ conecta a v_1 con v_2 con un peso w. No hay un significado global de este peso, puede significar distintas cosas para distintos problemas.

El complemento de un grafo G se define como $G' = (V', E')$ donde $V' = V$ y $e = (v_1, v_2) \in E' \iff e \notin E$, esto significa que una arista va a estar en el complemento si y solo si no está en el grafo original.

2.2. Máximo Conjunto Independiente

2.2.1. Definición del problema y problemas similares

El problema del Máximo Conjunto Independiente se define como: dado un grafo, encontrar un subconjunto de nodos con cardinalidad máxima tal que no exista un par de nodos del subconjunto que estén conectados por una arista.

Este problema está relacionado al problema del Clique Máximo que se define como: dado un grafo, encontrar un subconjunto de nodos con cardinalidad máxima tal que todos los nodos del subconjunto estén conectados por una arista. El problema del Clique Máximo en un grafo G es equivalente al problema del Máximo Conjunto Independiente en el complemento de G [79].

Este problema también está relacionado al problema de la Mínima Cobertura de Vértices que se define como: dado un grafo, encontrar un subconjunto de nodos con cardinalidad mínima tal que para todas las aristas del grafo, al menos uno de los nodos está en el subconjunto. Si uno obtiene un Máximo Conjunto Independiente I del conjunto de nodos V de G, entonces $V \backslash I$ es una Mínima Cobertura de Vértices de G [79].

2.2.2. Historia

Estos tres problemas han sido ampliamente estudiados debido a sus aplicaciones prácticas en numerosos dominios como lo son la bioinformática [33], teoría de códigos [18, 45], economía [8], entre otros. Además de estar involucrados en otros problemas importantes de optimización combinatorial como particionamiento de cliques, clustering de grafos, coloración de vértices, entre otros. Estos problemas son NP-hard [28] y también está demostrado que son difíciles de aproximar [14], esto significa que no se puede asegurar una respuesta ni siquiera aproximada con un algoritmo no exponencial. En Chalermsook et al [14] se demostró que para cualquier r más grande que una constante $\varepsilon > 0$, cualquier r-aproximación para el Máximo Conjunto Independiente tienen que correr en tiempo $2^{n^{1-\varepsilon}/r^{1+\varepsilon}}$ si la hipótesis de tiempo exponencial es cierta.

Dada su historia e importancia, muchos algoritmos se han desarrollado para resolver estos problemas. Por un lado, están los algoritmos exactos que exploran todo el espacio de búsqueda de la manera más eficiente posible garantizando que van a obtener la solución óptima. Por la naturaleza de estos problemas, sin importar lo mucho que acoten la búsqueda, sabemos que estos algoritmos van a estar limitados a tamaños pequeños si se quiere obtener una respuesta en un tiempo razonable. Por eso, también se ha intentado desarrollar algoritmos heurísticos que intentan acercarse lo más rápido posible a una buena respuesta, sin preocuparse por obtener la respuesta exacta.

Estos problemas han sido el foco de ciertas competencias en las que se ha buscado desarrollar nuevos algoritmos para resolver estos problemas. El segundo desafío DIMACS que se desarrolló entre 1992 y 1993 incluyó entre sus problemas el problema del Clique Máximo [40]. El dataset creado para el desafío todavía es usado frecuentemente como benchmark para comparar y evaluar algoritmos que resuelvan estos problemas debido a su variedad de instancias, incluyendo grafos en los que la respuesta se separa de las características que uno esperaría como serían nodos de alto grado, lo cual los hacen buenos casos de prueba para exponer a los algoritmos más simples. Otra competencia más reciente es el desafío PACE de 2019 que incluyó a la Mínima Cobertura de Vértices como un problema [23]. Esta competencia fomentó el desarrollo de nuevos algoritmos para resolver el problema, aunque su enfoque era sobre algoritmos exactos, no heurísticas.

Además del dataset de DIMACS, otro benchmark popular y más reciente es BHOSLIB (Benchmarks with Hidden Optimum Solutions for Graph Problems o Benchmarks con Soluciones Óptimas Escondidas para Problemas de Grafos) es un set de 36 grafos con soluciones óptimas escondidas que fueron obtenidas de instancias aleatorias difíciles para SAT [81].

Una de las áreas en las que estos problemas se aplican, teoría de código, mantiene una página [69] con instancias de grafos que representan problemas para los cuáles se está bus-

cando el Máximo Conjunto Independiente.

Wu et al [79] hace un review de los distintos algoritmos que existen para resolver el problema del Clique Máximo, tanto exactos como aproximados, explicando cómo funcionan y analizando sus resultados. Se toman en cuenta algoritmos para los tres distintos problemas, adaptándolos al problema del Clique Máximo. En este se identifica que los algoritmos de búsqueda local dominan en cuanto a tiempo y resultados. Además, se señalan siete algoritmos como los más dominantes: DLS, PLS, COVER, CLS, EWCC, AMTS y SBTS. Estos algoritmos se explican con más detalles en la siguiente sección.

2.2.3. Algoritmos

El algoritmo DLS (Dynamic Local Search), descrito por Pullan et al [59], es un algoritmo para el problema del Clique Máximo. En este se va construyendo un clique eligiendo vértices que se puedan agregar sin problemas al conjunto, estos se priorizan basados en penalties dinámicos que son ajustados durante la búsqueda relacionados al tiempo que ha pasado cada vértice dentro del conjunto solución. Cuando el clique ya no puede ser expandido se empiezan a intercambiar vértices que mantienen el tamaño de la solución pero cambian el espacio de búsqueda eligiendo vértices que solo tienen un vecino dentro del conjunto actual. Una vez que esas opciones de intercambio se acaban se hace una perturbación para cambiar de forma más drástica el espacio de búsqueda, eligiendo un nodo que agregar al azar o reduciendo el clique al último vértice agregado. Este algoritmo tiene un parámetro que se elige a mano, el penalty delay, que determina la frecuencia con que se reducen los penalties, este parámetro tiene que ser elegido manualmente de forma experimental dependiendo de la familia de grafos con la que se esté trabajando.

El algoritmo PLS (Phased Local Search), descrito por Pullan et al [60], es un algoritmo para el problema del Clique Máximo, basado en el algoritmo DLS. En este se pasan por varios sub-algoritmos similares a DLS, todos expanden la solución y cuando no pueden expandirla más, buscan vértices que intercambiar para mantener el tamaño de la respuesta, la diferencia es en cómo eligen los vértices que agregar. Uno elige los vértices al azar, otro elige los vértices con mayor grado y el último los elige según un penalty similar al de DLS. Con estos tres sub-algoritmos distintos es posible explorar el espacio de búsqueda de múltiples formas evitando que se enfoque demasiado en un solo tipo de solución. Además, se tiene una forma de perturbar el espacio de búsqueda igual que DLS. A diferencia de DLS, el algoritmo de pentalty no tiene el parámetro penalty delay, este se va ajustando dinámicamente a medida que corre el algoritmo.

El algoritmo COVER, descrito por Richter et al [63], es un algoritmo para el problema de la Mínima Cobertura de Vértices. Para este algoritmo se pide, además del grafo, un parámetro k que representa el tamaño de la Cobertura esperado, sin embargo, para convertir este algoritmo en uno que busca el mínimo basta con hacer una búsqueda lineal respecto al valor k que entregue resultados. Inicialmente, el algoritmo cubre la mayor cantidad de aristas que puede con k vértices y luego se procede a intercambiar vértices esperando que en algún momento se cubran todas las aristas. Para esto, se elige una arista al azar que no esté cubierta y se elige uno de sus dos vértices para agregar, para elegir al vértice se calcula un peso del

vértice basado en las aristas que sería capaz de cubrir en la solución y qué tan difíciles de cubrir son aquellas aristas, esto es un valor dinámico que se calcula a medida que se corre el algoritmo. Con este peso se intenta maximizar la ganancia que tendría el agregar el vértice. Esto también se toma en cuenta para ver que vértice eliminar de la solución actual.

El algoritmo CLS (Cooperating Local Search), descrito por Pullan et al [61], es un algoritmo para el problema del Clique Máximo. El algoritmo usa cuatro heurísticas de búsqueda local estocástica que trabajan en paralelo, compartiendo información entre sí para cubrir lo mejor posible el espacio de búsqueda. Son cuatro algoritmos de búsqueda similares a DLS, pasando por fases de expansión, búsqueda a través de intercambios y perturbaciones. La heurística GREEDY elige los vértices de mayor grado y por lo tanto, sirve para encontrar respuestas más evidentes. La heurística LEVEL también elige los de mayor grado pero hace una búsqueda basada en intercambios más extensa, lo cual permite explorar a mayor profundidad el espacio de búsqueda que encontraría GREEDY. La heurística FOCUS elige los vértices intentando conseguir que el grado promedio de los vértices dentro del conjunto solución sean iguales a un grado objetivo específico, este grado objetivo se elige intentando que se busquen espacios que no hayan sido buscados por las otras heurísticas. La heurística PENALTY se basa en principios de penalty similares a los de DLS para buscar en los espacios que ninguno de las otras heurísticas está buscando.

El algoritmo EWCC (Edge Weighting Configuration Checking), descrito por Cai et al [13], es un algoritmo para el problema de la Mínima Cobertura de Vértices. El algoritmo inicialmente crea una cobertura de vértices de acuerdo a un puntaje calculado con sus aristas, luego elimina un vértice, haciendo que el conjunto deje de ser una cobertura, entonces empieza a intercambiar vértices del conjunto solución con otros de acuerdo al mismo puntaje mencionado anteriormente, asegurándose de no caer en ciclos guardando un estado para cada nodo. Hace los intercambios hasta que este conjunto se vuelve una cobertura de vértices nuevamente, luego elimina otro vértice y repite el ciclo. El puntaje para cada vértice es calculado usando los pesos de las aristas que van cambiando dinámicamente a medida que corre el algoritmo.

El algoritmo AMTS (Adaptative Multistart Tabu Search), descito por Wu et al [78], es un algoritmo para el problema del Clique Máximo. En este, dado un k que es el tamaño del clique a buscar, se inicia un conjunto de nodos de forma greedy tomando los nodos que más aristas tienen dentro del conjunto y luego se busca intercambiar nodos tal que se maximice la función $f(S)$ que suma todas las aristas presentes en el conjunto, buscando que este número llegue a $\frac{k(k+1)}{2}$. Para estos intercambios, siempre se busca el par que haga la mayor diferencia y se usa una lista tabú para evitar ciclos. Además, para asegurarse de buscar en múltiples vecindades, el algoritmo se reinicia constantemente, tomando en cuenta un parámetro extra l que indica la profundidad de la búsqueda antes de detenerse.

El algoritmo SBTS (Swap Based Tabu Search), descrito por Jin et al [37], es un algoritmo para el problema del Máximo Conjunto Independiente. Este algoritmo se basa en movimientos de intercambio $(k, 1)$, en donde salen k nodos del conjunto solución y entra 1. Se toman decisiones de que movimiento ejecutar dependiendo de la situación actual, de forma similar a DLS, en fases de crecimiento, intercambio y perturbación. Además, se mantiene una lista de tabú para evitar ciclos en la búsqueda.

Todos los algoritmos mencionados mostraron un buen desempeño siendo probados con los datasets DIMACS y BHOSLIB, tanto en tiempo como en resultados [79]. Sin embargo, debido a que todos las heurísticas son estocásticas no existen garantías teóricas fuertes de su desempeño.

2.3. Máximo Corte

2.3.1. Definición del problema y problemas similares

El problema del Máximo Corte se define como: dado un grafo, dividir los vértices en dos conjuntos disjuntos A y B tal que cada nodo pertenezca a solo uno de los dos conjuntos y maximizar la suma de los pesos de las aristas que van entre nodos del conjunto A y el conjunto B.

Este problema frecuentemente se ve agrupado con el problema QUBO (Quadratic Unconstrained Binary Optimization u Optimización Cuadrática Binaria Sin Restricciones) donde se busca el vector $x \in \{0,1\}^n$ que maximice la ecuación $x^T Q x$ dado un $Q \in \mathbb{R}^{n \times n}$. Es posible reducir ambos problemas entre sí [20]. El problema QUBO se ha vuelto popular últimamente por su capacidad de expresar una gran cantidad de problemas de optimización combinatorial de forma sencilla.

2.3.2. Historia

El problema del Máximo Corte es un problema NP-Hard [29] que ha sido muy estudiado por su gran variedad de aplicaciones prácticas en áreas como diseño VLSI [3], diseño de redes [2] y segmentación de imágenes [17]. Este problema también se ha demostrado que es difícil de aproximar [42].

Para el trabajo de Martí et al [48], en el cual se desarrolló un algoritmo para resolver el problema del Máximo Corte usando scatter search o búsqueda esparcida, se recopiló un conjunto de instancias de prueba para el problema de distintas fuentes que habían generado instancias antes [34, 26] incluyendo instancias del séptimo desafío DIMACS.

Para el trabajo de Rendl et al [62], en el cual se desarrolló el algoritmo exacto *Biq Mac* para resolver los problemas QUBO y de Máximo Corte, se construyó la librería *Biq Mac*, que incluía un dataset con una variedad de instancias para ambos problemas, incluyendo algunas que representaban problemas reales en los que se aplican estos algoritmos [44].

Un trabajo muy importante que destacar es el de Dunning et al [20]. Este recopiló los algoritmos heurísticos más relevantes para los problemas QUBO y de Máximo Corte, incluyendo los dos mencionados anteriormente, programando los que no se encontraban disponibles y recopiló un dataset de más de 3000 instancias para los problemas resultando en la *MQLib*, una librería abierta con los algoritmos y las instancias. El trabajo también comparó todas las heurísticas, llegando a la conclusión de que ninguna dominaba en todos los aspectos, por

lo tanto desarrolló un modelo de machine learning que en base a las características del grafo elige la heurística más apropiada para resolver el problema. Los algoritmos para el problema del Máximo Corte que se implementaron fueron los descritos en Burer et al [12], Sousa et al [17], Duarte et al [19], Festa et al [26] y Laguna et al [43]. Estos algoritmos se describen en la siguiente sección.

2.3.3. Algoritmos

El algoritmo descrito en Burer et al [12] se basa en una solución de una relajación del problema del Máximo Corte usando programación semidefinida. En este trabajo se explica que el problema del Máximo Corte puede ser escrito como el siguiente problema de optimización de matrices: minimizar el valor de $\frac{1}{2}(W \cdot X)$ donde W es la matriz de los pesos de las aristas y se tiene que cumplir que la diagonal de X sean solamente unos, el rango de X sea 1 (que todos los valores sean unidades escalares) y que X sea una matriz simétrica semidefinida positiva. De la matriz X que minimiza la función, se puede obtener una respuesta al Máximo Corte sabiendo que $X_{i,j} = x_i x_j$, siendo $x_i \in \{-1, 1\}$ podemos agrupar los nodos dependiendo de su valor. En estudios anteriores [31] se había probado la efectividad de eliminar la condición de que el rango de X sea 1, es decir que x_i sean vectores de n dimensiones, para poder usar programación semidefinida y encontrar una solución aproximada. En este algoritmo, no se elimina la condición, si no que se relaja a que el rango de X sea 2, es decir, el valor asociado a cada nodo está en la circunferencia unitaria. Con esta relajación se calcula una respuesta aproximada que luego se traduce en una clasificación para el problema del Máximo Corte.

En el trabajo de Festa et al [26] se implementan distintos algoritmos basados en heurísticas aleatorias. Se describen tres técnicas de búsqueda importantes las cuales posteriormente se combinan para buscar cuál da mejores resultados. GRASP [25] es una heurística aleatoria de dos fases: construcción y búsqueda local. En la construcción se construye una solución inicial de forma greedy en la que se asignan los vértices al conjunto que maximice la suma de las aristas que se cruzan a medida que se van construyendo los dos conjuntos. Luego, en la búsqueda local se exploran movimientos que cambien un vértice de conjunto si es que el movimiento hace que el corte actual aumente, esto se repite hasta que no queden opciones y se encuentre un máximo local. Estas dos fases se reinician varias veces para partir de soluciones iniciales variadas y explorar distintos vecindarios. Path-relinking [30] es una técnica de búsqueda en la que se mantiene un conjunto de soluciones elite, las mejores encontradas hasta ahora, y para explorar el espacio después de encontrar un máximo local x, se elige una solución z de la soluciones elite y se pasa gradualmente de x a z buscando alternativas no exploradas. VNS [32] es una metaheurística en la que, dada una solución x, se van explorando distintos vecindarios de x: $N_1(x)$, $N_2(x)$, ... $N_k(x)$ de forma iterativa y en cada uno se hace una búsqueda local hasta que en una se encuentra un resultado mejor que x, entonces se toma ese resultado y se ejecuta toda esta búsqueda de nuevo sobre esa nueva solución. El trabajo testeó haciendo: un GRASP puro, un GRASP con path-relinking, un VNS puro, un VNS con path-relinking, un GRASP que usa VNS como búsqueda local y un GRASP con path-relinking que usa VNS como búsqueda local. Se concluye que GRASP con path-relinking era el algoritmo más rápido y VNS con path-relinking encontraba las mejores soluciones.

En el trabajo de Duarte et al [19] se describe un algoritmo híbrido entre VNS y un algoritmo memético para el máximo corte. En este se tienen una variedad de cortes posibles del grafo y en cada fase, se eligen algunos para ser mejorados con VNS y posteriormente se usa un algoritmo memético para mejorarlos, eligiendo dos cortes con buenos resultados se crea una mezcla de los dos donde los nodos que estén en el mismo conjunto se mantienen y cuando difieren se elige una asignación al azar.

En el trabajo de Laguna et al [43] se describe un algoritmo de entropía cruzada híbrido. El concepto de entropía cruzada para resolver un problema de optimización combinatorial es crear una distribución de probabilidad para generar respuestas al problema, generar una cantidad de respuestas posibles grande y en base a la calidad de las respuestas generadas, ajustar los parámetros de la distribución de probabilidad, este proceso se repite hasta que haya convergencia. En este estudio, se mezcla esta metodología con un proceso básico de búsqueda local para mejorar la calidad de las respuestas antes de ajustar los parámetros de la distribución para hacer que las respuestas lleguen a mejores resultados más rápido.

En el trabajo de Sousa et al [17] se describe un algoritmo de estimación de distribución. En este algoritmo se tienen varias distribuciones de probabilidad para hacer soluciones al problema que se van modificando y afectando entre sí, informando cuales son los vértices más elegidos y cambiando de acuerdo a esto.

Se ha probado que estos algoritmos, más que dominar entre sí, son con frecuencia complementarios, algunos algoritmos funcionan mejor en ciertas instancias y son peores en otras [20].

2.4. GNNs en Optimización Combinatorial

Las GNNs son redes neuronales artificiales diseñadas para poder trabajar con grafos. Para lograr esto, se adopta un método en dónde el grafo es entregado como parámetro con datos cargados en sus nodos y la GNN usa la estructura del grafo para conectar las neuronas de la red y termina entregando el mismo grafo estructuralmente con los datos procesados de cada nodo [65].

En general, existen capas de la red neuronal en las que cada neurona representa un nodo v del grafo y esta neurona recibe y agrega información de todos las neuronas que representan a los vecinos del nodo v. Una neurona con un vector de datos x_u calcula su salida con la siguiente función:

$$h_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}_u} \psi(x_u, x_v)\right)$$

Donde $\mathcal{N}_u$ es el vecindario del nodo u, la función $\bigoplus$ es una función de agregación invariante a las permutaciones y ϕ, ψ son funciones que se pueden aprender [11].

Mucho trabajo se ha hecho para tratar de resolver problemas de optimización combina-

torial con GNNs. El trabajo de Peng et al [57] se dedica a recopilar los métodos que han sido desarrollados para resolver este tipo de problemas con aprendizaje en grafos.

En general, la resolución se puede dividir en dos etapas. La primera etapa es el aprendizaje de representación de grafos en donde se traducen los grafos a vectores de dimensiones más bajas para facilitar el trabajo de estos. La segunda etapa es usar machine learning para resolver el problema, usando las representaciones del grafo que se obtuvieron en la primera etapa.

Para la primera etapa, hay dos formas de hacerlo. Una son los métodos de embedding de grafos en los cuales la representación del grafo tiene un objetivo de por sí que puede no tener relación al problema a resolver, como por ejemplo los métodos basados en SkipGram [53] en donde se busca maximizar la similitud del encoding de cada nodo con su vecindario o los métodos basados en AutoEncoder [74] donde se construyen los encodings para que la función de decoding cumpla condiciones específicas. El otro método es un aprendizaje end-to-end donde la representación no es más que un paso intermedio de todo el proceso y se está entrenando en conjunto con la resolución del problema, métodos como usar el grafo en el entrenamiento y usar un AutoEncoder específico para el problema son ejemplos.

Para la segunda etapa, se pueden tener métodos no-autorregresivos en los que la respuesta se entrega toda de una vez o métodos autorregresivos en los que la respuesta se va construyendo iterativamente. Los métodos autorregresivos tienen mejores resultados en problemas secuenciales como TSP [41], aunque tomándose un tiempo considerable y escalando mal para grafos grandes, sin embargo en problemas sin características secuenciales los métodos no-autorregresivos funcionan mejor [36].

La implementación del trabajo en Schuetz et al [68] dejada como ejemplo[1] usa un embedding, pero no lo usa para conseguir la representación de un grafo, si no que lo usa como input para una GNN, por lo tanto, usa aprendizaje end-to-end. Para el problema del Máximo Conjunto Independiente genera un vector de probabilidades de que cada nodo esté en el conjunto solución. Para el problema del Máximo Corte genera un vector de probabilidades también y en base a eso se divide en dos conjuntos. Ambos métodos son no-autorregresivos.

2.5. Benchmarks de GNNs

Schuetz et al [68] describe como las GNNs han tenido una explosión de popularidad en los últimos años debido a su alto potencial para abordar una gran variedad de problemas como clasificación de usuarios en redes sociales [58], predicciones en sistemas de recomendación [82] y predicción de propiedades en grafos moleculares [70].

En un área de desarrollo tan activa como esta, las benchmarks son sumamente importantes para poder comparar el desempeño de los distintos trabajos y comprobar que efectivamente se estén creando mejores soluciones.

Se han creado benchmarks para GNNs de áreas específicas como la química [27] que al

[1] `https://github.com/amazon-science/co-with-gnns-example`

estar diseñadas para un uso específico, cumplen muy bien su propósito [54, 46].

Otros trabajos han creado benchmarks para GNNs que tratan de abarcar una gran cantidad de áreas [4, 22], algunas incluso incluyendo datasets para optimización combinatorial. Estas benchmarks dan herramientas para crear GNNs dentro de un framework que está diseñado para que las comparaciones entre distintas GNNs sean justas. Además, traen consigo un gran conjunto de datasets para muchos tipos de problemas. Son herramientas sumamente útiles para testear y comparar GNNs entre sí, pero no tienen la capacidad de comparar los desempeños con algoritmos clásicos debido a que los frameworks para crear soluciones que se proveen solo dan herramientas para crear GNNs.

El trabajo de Palowitch et al [55] se enfoca en ser capaz de crear datasets de grafos que abarquen las áreas que las benchmarks anteriores no pueden cubrir por ser estáticas, con esto afirma poder medir la calidad de una GNN con una cobertura más amplia del mundo de características de los grafos, sin embargo, esto aún deja en las manos del investigador qué grafos serían los más importantes de revisar y a qué características se tiene que estar atento en la evaluación de la GNN.

Capítulo 3

Solución

El problema que existe actualmente con el desarrollo de GNNs es que estas no están siendo evaluadas correctamente con respecto al estado del arte, la mayoría de los estudios solo se limitan a comparar redes neuronales entre sí dando conclusiones incompletas respecto a su desempeño. Sería conveniente tener una herramienta que permita la comparación del desempeño de las GNNs con algoritmos de otra índole, basados en otros paradigmas de resolución, para poder tener una mejor idea de qué tan útil es el modelo realmente.

3.1. Framework

Para resolver este problema se crea un framework que permite organizar distintas soluciones de un problema en específico para compararlas entre sí con distintos datasets[1] de prueba. La idea es que al crear una instancia de este framework se incorporen algoritmos clásicos que resuelvan el problema al nivel del estado del arte y un grupo de datasets que representen casos de prueba sintéticos y reales. La idea es que el framework haga la parte de recopilar los algoritmos y casos de prueba necesarios para poder evaluar como se compara una nueva solución con las mejores soluciones actuales.

El framework se estructura como una serie de archivos y carpetas que, al ser usadas de acuerdo a las especificaciones, permite la fácil ejecución de distintos conjuntos de instancias provistas para el problema que se quiera resolver junto a herramientas para analizar los resultados en comparación con una serie de algoritmos clásicos que también intentan resolver el problema. Está pensado para GNNs, pero es compatible con cualquier tipo de algoritmo que se pueda correr en la terminal.

Se construyeron dos instancias del framework, una llamada "mc_testing"[2] para el problema del Máximo Corte y otra llamada "mis_testing"[3] para el problema del Máximo Conjunto

[1] Un dataset es un conjunto de instancias del problema. En este caso, son varios grafos que correspondan a instancias válidas del problema.

[2] `https://github.com/dabaez/mc_testing`

[3] `https://github.com/dabaez/mis_testing`

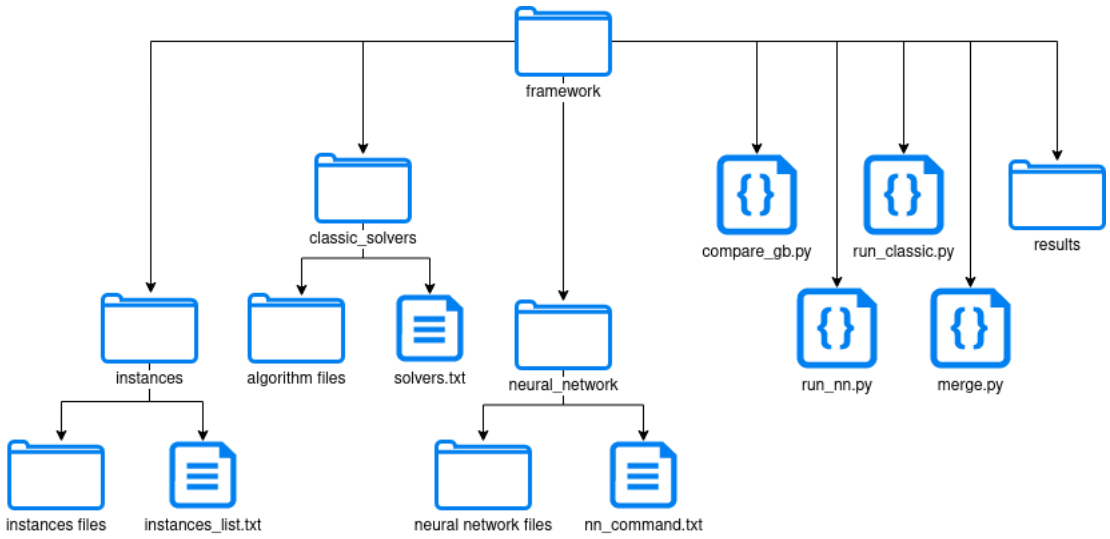

Figura 3.1: Estructura del framework. Se compone de distintas carpetas para estructurar el trabajo y códigos que facilitan la evaluación y el análisis de los resultados.

Independiente. Ambas están hosteados en GitHub y son de libre acceso público. Estructuralmente son idénticas y solo se diferencian en los distintos algoritmos clásicos y datasets que tienen.

3.1.1. Carpetas

El framework se organiza en carpetas que separan las distintas partes de la evaluación: los casos de prueba en la carpeta "instances", los algoritmos para comparar en la carpeta "classic_solvers", el algoritmo a desarrollar en "neural_network" y los resultados en la carpeta "results". La organización en carpetas se realizó para estructurar el trabajo en diferentes secciones y poder ubicar fácilmente las distintas herramientas para el momento de las evaluaciones. Esta organización se puede ver en la Figura 3.1.

La carpeta "instances" es donde se guardan todos los distintos datasets elegidos que sean relevantes para el problema que se aborde en el framework respectivamente. Cada dataset tiene su propia carpeta en donde se guardan todas sus instancias, cada grafo es guardado como una lista de aristas, este formato es popular por su simplicidad y compactibilidad, siendo elegido para trabajos como el de Dunning et al [20]. Junto a los datasets está el archivo "instances_list.txt" que se usa como una lista para poder iterar a lo largo de los distintos datasets que se quieran evaluar, siendo posible comentar los que no se quiera usar, además de proveer una descripción de los datasets presentes.

La carpeta "classic_solvers" es donde se guardan las implementaciones de los algoritmos clásicos para resolver el problema en cuestión, algoritmos basados en métodos distintos a las redes neuronales. Todos los algoritmos deben ser programados tal que reciban un archivo con el formato de grafo de los datasets y un tiempo límite. Se decide dar un tiempo límite porque la mayor parte de estos algoritmos heurísticos tienden a encontrar una solución y tratar de mejorarla por la mayor cantidad de tiempo que puedan, entonces se decidió que sería

mejor darle este límite de tiempo a todos los algoritmos para que lo usaran internamente. Se espera que la última linea impresa en consola por los algoritmos sea el resultado encontrado y el tiempo que se tomó en encontrarlo separado por una coma. Se pide el tiempo que se tomó en encontrarlo ya que el tiempo límite no es algo que se imponga externamente y con frecuencia estos algoritmos pueden tener formas de terminar su ejecución de forma temprana, por lo tanto, se considera como una métrica útil de guardar. Se decidió usar el texto de la consola como canal de información para recibir los resultados por su facilidad de uso y por su compatibilidad con cualquier lenguaje que pueda imprimir texto a la terminal. Junto a los algoritmos está el archivo "solvers.txt" que se usa como una lista para poder iterar a lo largo de los distintos algoritmos que se quieran evaluar, siendo posible comentar los que no se quiera usar. En esta lista se tiene que entregar un comando de terminal que pueda ser usado desde esta carpeta para correr el algoritmo, con argumentos para poder insertar el camino al archivo y el tiempo límite.

La carpeta "neural_network" es donde se espera que se desarrolle el trabajo de la red neuronal a evaluar. Para que sea testeado, se tiene que escribir el comando para correr la red neuronal sobre un grafo en el archivo "nn_command", donde se tiene que tener un lugar donde poner el nombre del archivo que tiene el grafo y un tiempo límite. El tiempo límite es para que se pueda hacer una comparación similar a los algoritmos clásicos, pero como este es un argumento que recibe la red neuronal y no es algo que se imponga por el programa, puede ser recibido e ignorado. Este comando debe funcionar igual que los de los algoritmos clásicos, se espera que la última línea impresa en la consola sea el resultado y el tiempo que se tomó separado por una coma. En el archivo también se pueden poner múltiples comandos para redes neuronales, en caso de que se quieran probar distintos hiperparámetros o configuraciones.

En la carpeta "results" es donde los scripts "run_classic.py" y "run_nn.py" van dejando todos los resultados de cada test, organizados primero en subcarpetas con el nombre del test, luego en subcarpetas de acuerdo al algoritmo y finalmente en archivos de formato csv para cada dataset.

3.1.2. Scripts

Los scripts que están en la raíz del framework son utilidades para ejecutar la evaluación y facilitar el análisis de los resultados.

Para hacer posible múltiples sesiones de evaluación o distintas agrupaciones de resultados, se usa el parámetro "testname" (nombre del test) que agrupa múltiples resultados.

Los scripts "run_classic.py" y "run_nn.py" sirven para correr los algoritmos clásicos y la red neuronal sobre las instancias escogidas respectivamente. Ambos códigos están hechos para ser corridos en la terminal con los comandos:

```
$ run_classic.py [testname] [timelimit]
$ run_nn.py [testname] [timelimit]
```

El "timelimit" o tiempo límite es lo que se le va a entregar a los algoritmos al correr. El "testname" es para organizar todos los resultados, tanto de los algoritmos clásicos como los

Tabla 3.1: Tabla detallando los tamaños y densidades de los datasets **gb**.

	Tamaño	Densidad
Máximo Conjunto Independiente	{100,250,500,750,1000}	{0.001,0.005,0.01, 0.05,0.1,0.5,0.9}
Máximo Corte	{100,250,500,750,1000}	{0.01,0.05,0.1,0.5,0.9}

de las redes neuronales. Se espera que para comparar un grupo de algoritmos clásicos con un grupo de redes neuronales se use el mismo nombre de test.

gb Datasets

Los datasets de tipo **gb** son grafos aleatorios que varían en tamaño y densidad. Los distintos valores para el tamaño y la densidad fueron escogidos buscando ser similares en órdenes de magnitud a los valores que se encontraban en los datasets que se habían encontrado durante la investigación del problema, pensados para que tomen demasiado tiempo en ser resueltos de forma exacta pero que sea posible obtener una buena aproximación con heurísticas en un tiempo razonable. Los valores escogidos se encuentran en la Tabla 3.1. Para cada combinación de tamaño y densidad se generaron cinco grafos. Se escogieron estas características ya que son las que más claramente afectan a la complejidad de una solución, el tamaño siendo un impedimento para realizar búsquedas de mayor profundidad en un tiempo razonable ya que el espacio de búsqueda crece de forma exponencial con este y la densidad agregando directamente más condiciones al problema, restringiendo la solución en el caso del Máximo Conjunto Independiente e incluyendo más factores en el caso del Máximo Corte. Al comparar las densidades con los valores de las Tablas 3.2 y 3.3, se puede notar una discordancia con las densidades mínimas y máximas. Esto se debe a que los grafos se construyeron usando la densidad como la probabilidad de tener una arista entre dos nodos, esto causa una gran desviación estándar en los grafos pequeños. Todos los códigos usados para crear los grafos están incluidos en su instancia del framework correspondiente.

Se escogieron el tamaño y la densidad como los valores que determinan el dataset y se usan para construir los gráficos y analizar los resultados porque son las características que más evidentemente determinan la complejidad del problema a resolver. A pesar de que existen muchos otros factores que determinan la dificultad de un problema, el tamaño y la densidad son dos factores claves que afectan en la cantidad de información que tiene el problema. Se decidió no abordar más características por la dificultad de visualizar esta información de una manera útil.

"compare_gb.py" es un código que sirve para analizar los resultados de los datasets de tipo **gb** que fueron creados para este trabajo. Está hecho para ser corrido en la terminal con el siguiente comando:

```
$ compare_gb.py [dataset] [testname] [main-alg]
```

El argumento de "dataset" es para decir de que dataset se quieren sacar los resultados, "testname" para saber de qué ejecuciones sacar los resultados y "main-alg" es para elegir un algoritmo que utilizar para el segundo gráfico. Primero, entrega un gráfico como el que se

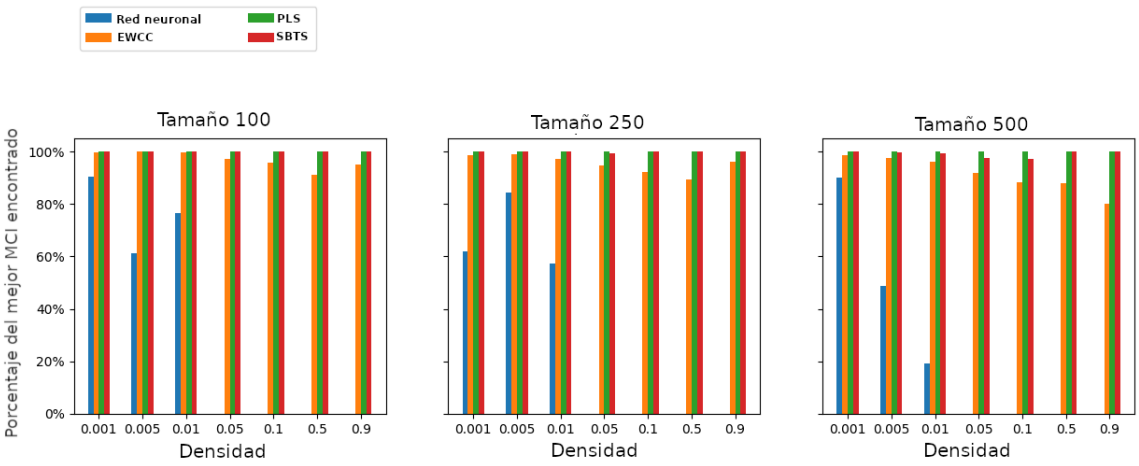

Figura 3.2: Muestra de como se ven algunos gráficos de barra al usar "compare_gb.py" para el problema del Máximo Conjunto Independiente. En este caso se están comparando cuatro algoritmos: Red neuronal, PLS, EWCC y SBTS, cada uno representado por un color distinto. El eje Y representa el porcentaje de la mejor respuesta obtenida entre todos los algoritmos, promediado entre 5 instancias distintas. El eje X representa la densidad de los grafos de cada caso.

puede ver en la Figura 3.2 en el que se muestran varios gráficos de barra; para cada algoritmo se muestra qué porcentaje de la mejor respuesta obtenida entre todos los algoritmos obtuvo en comparación con la densidad del grafo, esto en varios gráficos que representan varios grafos de distintos tamaños. Se escogió el porcentaje de la mejor respuesta obtenida entre todos los algoritmos como medida para representar la calidad del algoritmo porque era posible estandarizarla a través de varios grafos sin importar cuál fuera la respuesta para cada uno respectivamente. Es de suma importancia probar más de un grafo por categoría porque siempre es posible que con uno solo se prueben características específicas de la instancia más que una generalización del problema. Con este gráfico se puede comparar a simple vista el desempeño de los algoritmos clásicos con las redes neuronales, ver en qué momento la diferencia es notable o si está a la par de algún algoritmo en específico. Después de esto, se entrega un segundo gráfico como el que se puede ver en la Figura 3.3, donde se usan los mismos datos del gráfico anterior pero de solo un algoritmo y se puede ver en un mapa de calor como varía su porcentaje de acuerdo a las características del grafo.

En caso de que el dataset no sea del tipo **gb**, al no tener una forma general de graficar los datos, se hizo el código "merge.py" al cual, dado un nombre de test y un dataset, acopla todos los datos en un archivo de formato csv que se deja en la carpeta "results" junto a los resultados de los demás tests para poder comparar el desempeño de los grafos a la medida.

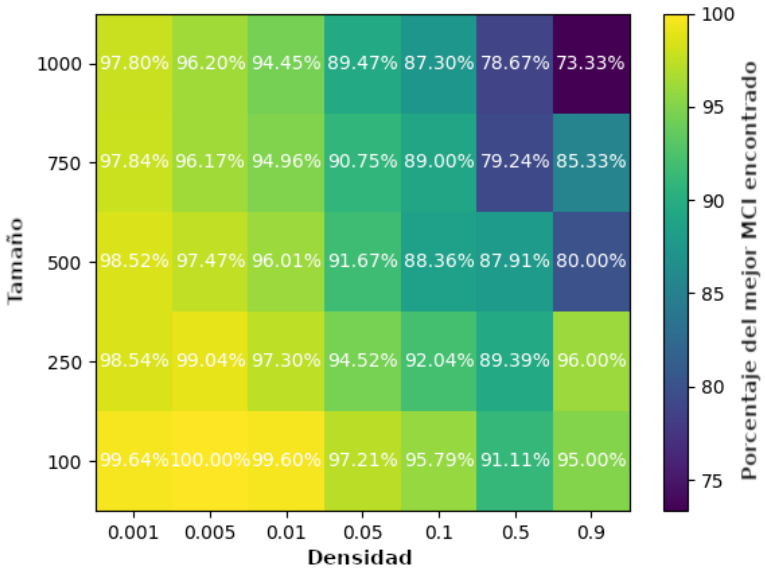

Figura 3.3: Ejemplo de gráfico de mapa de calor de "compare_gb.py". El eje X representa la densidad del grafo y el eje Y representa el tamaño del grafo. El color de cada celda y el texto escrito en ellas es el porcentaje de la mejor respuesta obtenida entre todos los algoritmos, promediado entre cinco instancias distintas.

3.2. Máximo Conjunto Independiente

3.2.1. Algoritmos

Para el problema del Máximo Conjunto Independiente, se eligieron tres de los siete algoritmos dominantes mencionados en Wu et al [79] para ser implementados y utilizados como las heurísticas base con las que comparar a las redes neuronales.

Se descartaron los algoritmos que requerían de parámetros extra como DLS y AMTS ya que estos requieren una calibración manual donde se busca el mejor valor para determinadas características del grafo y esto no se podía automatizar. Se descartó CLS por su dependencia en un sistema de múltiples núcleos. Como los algoritmos COVER y EWCC eran muy similares, se decidió escoger solo uno de ellos, EWCC por ser el más reciente. Entonces se terminó escogiendo los algoritmos PLS, EWCC y SBTS.

A pesar de que algunos de estos algoritmos afirmaban estar disponibles para descargar, no se pudo encontrar ninguna implementación de ellos así que fueron programados para ser parte de este framework y su código quedará disponible en la instancia del framework "mis_testing" para cualquier uso que se le quiera dar a futuro.

Los algoritmos fueron programados en C++ ya que es reconocido por ser un lenguaje rápido y preciso con el cuál se puede asegurar eficiencia a la hora de realizar algoritmos intrincados como estos. Un lenguaje de más alto nivel hubiera podido afectar la eficacia de los problemas de forma considerable. No se usó más que la librería estándar de C++.

En los artículos originales todos estos algoritmos traían una tabla de resultados con respecto al dataset DIMACS y se dedicó gran parte del trabajo a tratar de que las implementaciones desarrolladas obtuvieran los mismos resultados, sin embargo esta meta no se pudo cumplir. En términos de velocidad, los algoritmos tenían velocidades similares a las descritas en los artículos, pero las soluciones tendían a ser más pequeñas con una diferencia de uno a cuatro elementos. Se obtuvieron algoritmos que cumplen su cometido, ser un punto de comparación de un algoritmo eficaz que resuelva el problema del Máximo Conjunto Independiente, pero no al nivel que se hubiera deseado.

Los tres algoritmos fueron programados de acuerdo a las especificaciones de los distintos artículos, sin embargo se hicieron pequeñas modificaciones para hacer que los algoritmos corrieran con un límite de tiempo, en vez del límite de iteraciones que se mencionaba.

3.2.2. Datasets

Para el problema del Máximo Conjunto Independiente se eligieron cuatro datasets:

- El dataset "gb_mis" (MIS por Maximum Independant Set) que fue creado para análisis más superficiales, con grafos aleatorios que varían en tamaño y densidad como se muestra en la Tabla 3.1. Estos están para ser analizados con la herramienta "compare_gb.py".
- El dataset "DIMACS", creado para la Segunda Competencia de Implementación DIMACS en la que el desafío era encontrar soluciones novedosas para el problema del Clique Máximo [40]. Múltiples familias de grafos están incluidas en este dataset. La familia brock de grafos aleatorios con cliques escondidos en nodos de bajo grado [10]. La familia C con grafos aleatorios que van desde los 125 nodos hasta los 4000 nodos y densidades que van desde 0,5 hasta 0,9. La familia c-fat con grafos que parecen anillos gruesos, ciclos en los que múltiples nodos están en cada punto [6]. La familia de DSJC de grafos aleatorios usados en Johnson et al [38]. La familia gen de grafos aleatorios con cliques grandes insertados. La familia hamming con grafos que usan nodos que representan palabras y aristas que representan si están a cierta distancia de Hamming entre sí. La familia johnson que también usa la distancia de hamming en vectores con peso. La familia keller de grafos que representan la teselación de un hipercubo, problema en el que el Clique Máximo ha sido estudiado [15]. La familia MANN que son formulaciones de sistemas triples de Steiner como problemas de Clique Máximo. La familia p-hat de grafos generados con el generador de grafos p-hat, estos tienen una distribución más amplia de grados de nodos y cliques más grandes que grafos uniformes. La familia san con grafos generados para el problema de Cobertura de Vértices [66]. Como estas instancias estaban diseñadas para el problema del Clique Máximo, se tomó el complemento de todos los grafos para esta librería. Los grafos y la información se recolectaron del trabajo mantenido por Mascia [50].
- El dataset "BHOSLIB" de Benchmarks con Soluciones Óptimas Escondidas para Problemas de Grafos, diseñado a través de un método para codificar instancias con soluciones difíciles de encontrar traducidas de instancias difíciles aleatorias de SAT [81].

Tabla 3.2: Tabla resumen de datasets para el Máximo Conjunto Independiente.

Dataset	Número de grafos	Número de nodos mínimos	Número de aristas mínimas	Número de nodos máximos	Número de aristas máximas
gb_mis	175	100	101	1000	450102
DIMACS	80	28	72	4000	3997732
BHOSLIB	36	450	17794	4000	572774
CODE	31	8	6	2048	504451

Dataset	Densidad mínima	Densidad máxima	Densidad promedio
gb_mis	0.0029	0.8993	0.2276
DIMACS	0.0012	0.9604	0.3402
BHOSLIB	0.0716	0.1761	0.1343
CODE	0.0090	0.6266	0.1452

Estos grafos eran originalmente para el problema del Clique Máximo, para la librería se tomó el complemento de ellos. Los grafos y la información se recolectaron del trabajo mantenido por Mascia [49].

- El dataset "CODE" con grafos que vienen de teoría de código, para problemas en los que se está buscando el Máximo Conjunto Independiente. Para los grafos más grandes, aún hay óptimos que se desconocen. Los grafos y la información se recolectaron del trabajo mantenido por Sloane [69].

Un resumen de estadísticas de los datasets se puede encontrar en la Tabla 3.2.

3.3. Máximo Corte

3.3.1. Algoritmos

Para el problema del Máximo Corte se usaron los algoritmos implementados en Dunning et al [20] ya que eran una buena selección de algoritmos verificados con múltiples casos de prueba, además de ofrecer una amplia variedad de soluciones lo cual es beneficioso cuando se quiere comparar un algoritmo con métodos alternativos para tener una buena muestra de las opciones que existen. Solo se programó una interfaz para pasar entre el código programado y nuestro framework.

Con esto se tienen once algoritmos en el framework, lo cual puede hacer difícil trabajar la información debido a que los gráficos empiezan a volverse difíciles de leer con tantos datos, sin embargo, como es posible desactivar los algoritmos para no ser corridos en las instancias, es posible dejar solamente una selección más exclusiva para trabajar. Además, es importante que en un problema como el Máximo Corte se dejen disponibles todos los algoritmos, ya que como se estudió en el trabajo que los implementó, todos tienen situaciones en los que se desempeñan mejor a los demás.

3.3.2. Datasets

Para el problema del Máximo Corte se escogieron nueve datasets:

- Cinco datasets son del tipo **gb**: "gb_1_10f", "gb_1_100i", "gb_m1_1i", "gb_m10_10f" y "gb_m100_100i". El primer y segundo número representan el peso mínimo y máximo de las aristas y el último caracter puede ser i para números enteros y f para números de punto flotante. Por ejemplo, en el dataset "gb_1_100i" se incluyen aristas cuyos pesos siguen una distribución uniforme con valores enteros del 1 al 100. Se decidió tener estos datasets para poder captar como funcionan los algoritmos con distintos rangos de pesos. Estos datasets están ahí para ser analizados con la herramienta "compare_gb.py" y varían en tamaño y densidad de acuerdo a los valores de la Tabla 3.1.
- El dataset "BiqMac" que tiene una variedad de grafos sintéticos para la librería BiqMaq [62] de distintos tipos de familias generados con el generador de grafos Rudy [64]. La familia g05 de grafos sin peso, donde cada arista tiene probabilidad 0,5 de aparecer. La familia pm1s de grafos de densidad 0,1 con pesos elegidos al azar entre {-1,0,1}. La familia pm1d con grafos de densidad 0,99 con pesos elegidos al azar entre {-1,0,1}. La familia wd100 de grafos con pesos elegidos al azar entre $[-10, 10]$ y densidad $d = \{0{,}1, 0{,}5, 0{,}9\}$. La familia pw$d$100 de grafos con pesos elegidos al azar entre $[0, 10]$ y densidad $d = \{0{,}1, 0{,}5, 0{,}9\}$. Los grafos y la información se recolectaron del trabajo mantenido por Mallach [47].
- "Liers" que tiene distintos grafos de problemas de física estadística en donde se aplica el problema en cuestión. Estos fueron generados por Frauke Liers [44]. Los grafos y la información se recolectaron del trabajo mantenido por Mallach [47].
- El dataset "Mannino" que tiene instancias recolectadas del mundo real por interferencias de frecuencias de radio en un problema de asignación de frecuencias. Se ha investigado el Máximo Corte en estas instancias [9]. Los grafos y la información se recolectaron del trabajo mantenido por Mallach [47].
- El dataset "DIMACS" que tiene cuatro instancias de problemas que aparecieron en el séptimo desafío de implementación DIMACS [16]. Estos grafos están relacionados a problemas de física estadística que se resuelve con el máximo corte. Los grafos y la información se recolectaron del trabajo mantenido por Schmieta [67].

Un resumen de estadísticas de los datasets se puede encontrar en la Tabla 3.3.

En algún momento se pensó ocupar las instancias recopiladas en Dunning et al [20], pero al ser sobre 2000 grafos, esto tomaba mucho tiempo de evaluación y además era sumamente difícil organizar los datos para mostrarlos de alguna forma que fuera útil para la persona que quiere usar el framework.

Tabla 3.3: Tabla resumen de datasets para el Máximo Corte.

Dataset	**Número de grafos**	**Número de nodos mínimos**	**Número de aristas mínimas**	**Número de nodos máximos**	**Número de aristas máximas**
gb_1_10f	125	100	40	1000	449968
gb_1_100i	125	100	43	1000	449726
gb_m1_1i	125	100	26	1000	449801
gb_m10_10f	125	100	39	1000	450039
gb_m100_100i	125	100	43	1000	449678
BiqMac	130	60	316	100	4901
Liers	48	100	200	400	34753
Mannino	4	48	1128	487	8511
DIMACS	4	512	1536	3375	10125

Dataset	**Densidad mínima**	**Densidad máxima**	**Densidad promedio**
gb_1_10f	0.0079	0.8990	0.3096
gb_1_100i	0.0085	0.8986	0.3094
gb_m1_1i	0.0051	0.8987	0.3094
gb_m10_10f	0.0077	0.8992	0.3098
gb_m100_100i	0.0085	0.8985	0.3099
BiqMac	0.0911	0.9705	0.4972
Liers	0.0100	0.9665	0.4657
Mannino	0.0121	0.9592	0.2721
DIMACS	0.0018	0.0117	0.0067

Capítulo 4

Evaluación

Para probar el framework se probó el modelo de GNN desarrollado en Schuetz et al [68] ya que este podía abordar tanto el problema del Máximo Conjunto Independiente como el problema del Máximo Corte y se sabía de antemano que estas soluciones no estaban a la par de los algoritmos clásicos, el análisis de Angelini et al [1] dejó en claro que esta solución era más lenta y daba peores resultados que un algoritmo simple. El trabajo original tenía una implementación disponible como ejemplo para el problema del Máximo Conjunto Independiente la cual fue usada para realizar las comparaciones de este ejemplo, modificándola ligeramente para resolver el problema del Máximo Corte de acuerdo a las indicaciones del artículo. También se hizo una modificación para que el algoritmo se detuviera con un tiempo límite determinado.

4.1. Máximo Conjunto Independiente

Para este problema, se trató de evaluar el algoritmo con las instancias del dataset de DIMACS pero estaba consistentemente dando 0 como respuesta, no eligiendo ningún nodo para ser parte del conjunto lo cual es un problema, ya que no estaba tomando ni siquiera un nodo al azar que siempre es una solución válida. El artículo original decía que la red neuronal fue testeada con grafos de tamaños hasta 10^6, pero todos los grafos de DIMACS le estaban dando problemas al punto que no entregaba ni siquiera un nodo. El problema del testing original es que solo probaron con grafos d-regulares con $d = 3$ y $d = 5$. Un grafo d-regular se caracteriza porque cada nodo tiene d vecinos. Estos son solo casos de grafos muy esparsos.

Se hizo un test con el dataset "gb_mis", dándole 100 segundos a los algoritmos clásicos y a la red neuronal. Los resultados finales indican que el porcentaje de la mejor respuesta encontrada en promedio fue 24,45 %, con una mediana del 0 % y una desviación estándar de 34,81 %. Estos resultados se pueden ver comparados a los de los algoritmos clásicos en la Tabla 4.1.

Se puede apreciar en la Figura 4.1 que PLS y SBTS tienen un funcionamiento similar, EWCC no funciona tan bien con altas densidades y la red neuronal no presenta resultados

Tabla 4.1: Resumen de los resultados en el dataset "gb_mis". Se calculan el promedio, la mediana y la desviación estándar del porcentaje de la mejor respuesta que se consigue para cada caso de prueba con los resultados de cada algoritmo.

Algoritmo	**Promedio**	**Mediana**	**Desviación Estándar**
GNN	24.45 %	0 %	34.81 %
EWCC	92.71 %	94.22 %	6.14 %
PLS	100 %	100 %	0 %
SBTS	99.26 %	100 %	1.40 %

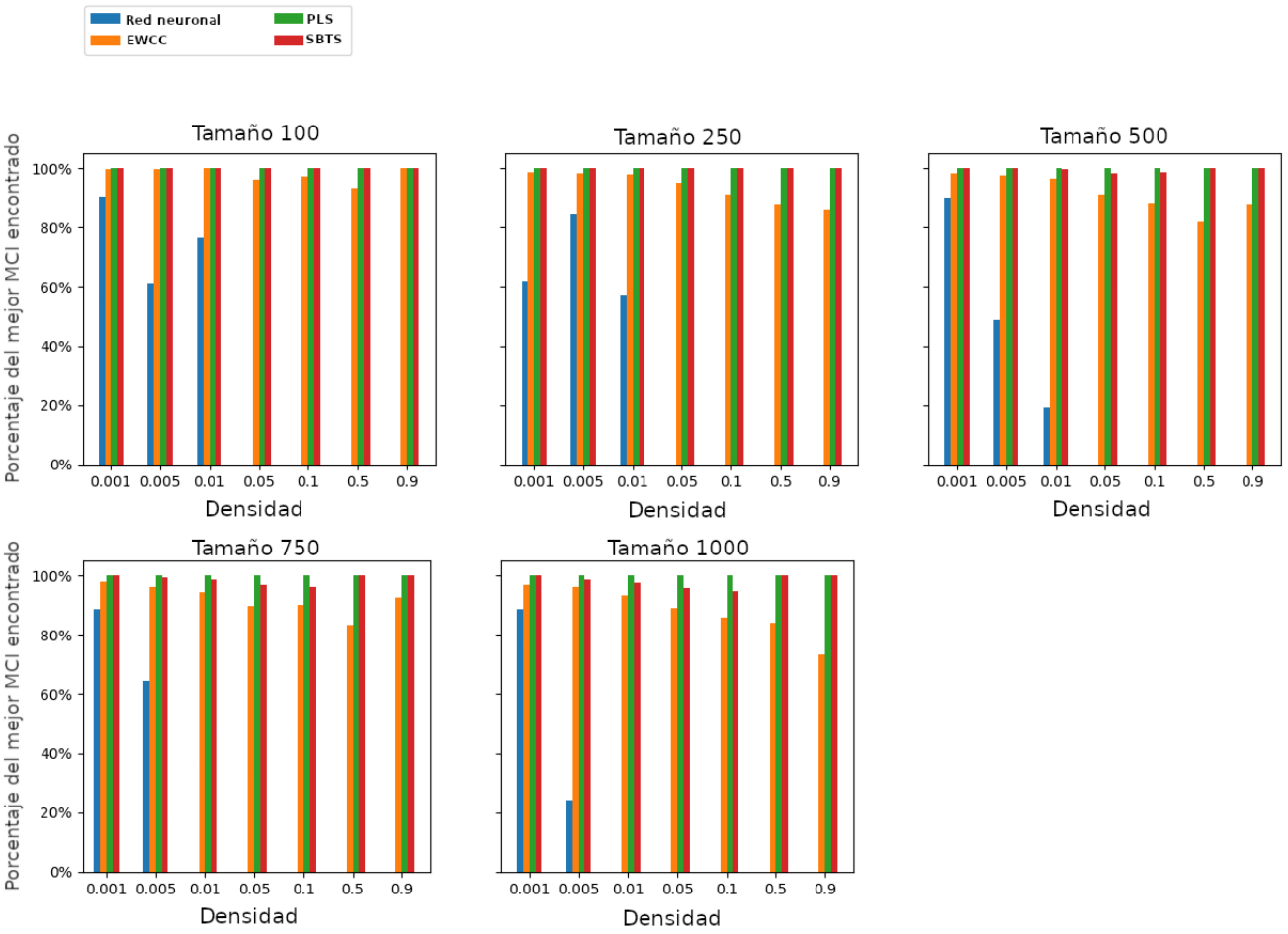

Figura 4.1: Gráficos de barra del problema del Máximo Conjunto Independiente con la red neuronal. Se compara la red neuronal del trabajo de Schuetz et al [68] con las tres heurísticas programadas para el problema. El eje X es la densidad de los grafos y el eje Y es el porcentaje de la máxima respuesta encontrada entre todos los algoritmos para el Máximo Conjunto Independiente, agrupados entre múltiples casos. Los múltiples graficos representan distintos tamaños de grafos.

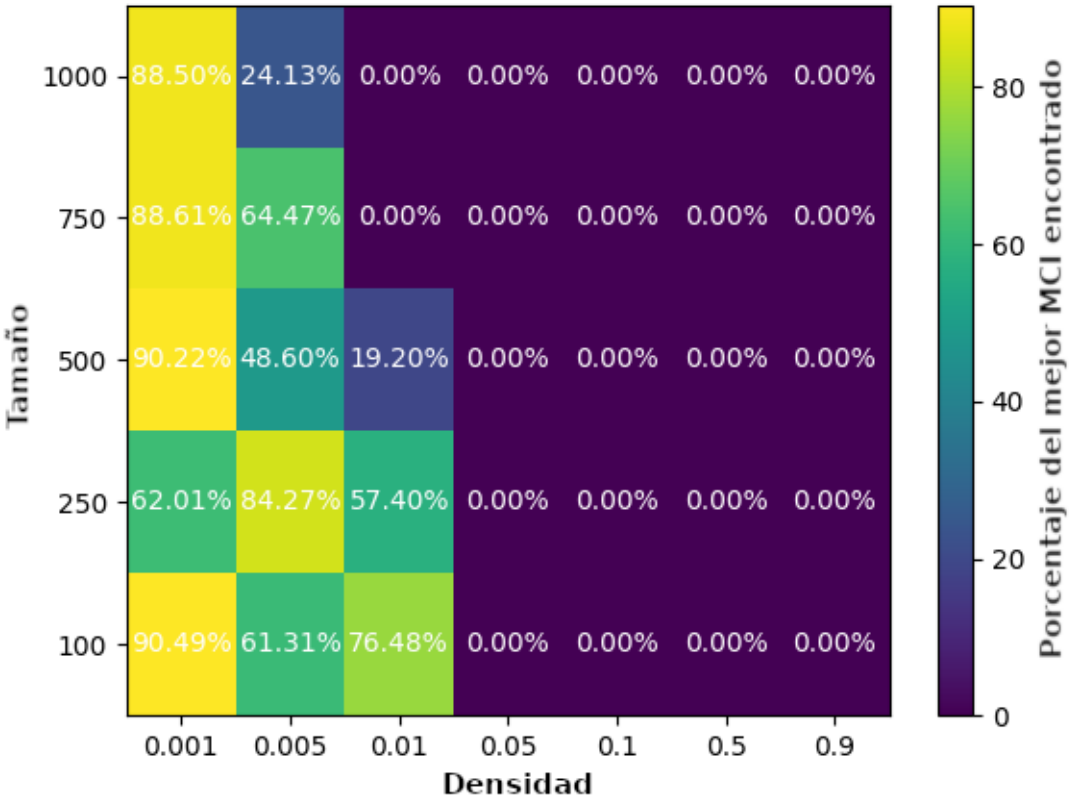

Figura 4.2: Mapa de calor del problema del Máximo Conjunto Independiente con la red neuronal. El eje X es la densidad de los grafos, el eje Y es el tamaño de los grafos medido en cantidad de nodos y el color de cada celda es el porcentaje que fue su respuesta en comparación con el conjunto más grande alcanzado entre todos los algoritmos.

en general para densidades mayores a 0,01. En la Figura 4.2 se puede ver a más detalle el comportamiento de la red neuronal, un 0 % significa que su respuesta no incluía ningún nodo. Nunca muestra respuestas para más de la mitad de las instancias. Se revisó más a fondo y resulta que la red neuronal se queda trabada en un óptimo local que es no tomar ningún nodo. Cuando la densidad es muy grande, la función de pérdida fácilmente llega al punto estable de no tomar ningún nodo, pero es muy difícil que tome cualquier grupo de nodos ya que, como la respuesta es pequeña, no hay suficiente incentivo para que la respuesta se mueva a cualquier otra configuración.

Esta red neuronal calcula una probabilidad de que cada nodo sea parte del conjunto final que sería la respuesta, pero ninguno de los nodos es suficientemente crucial para que su probabilidad termine siendo suficiente para ser incluido en la respuesta.

Hubo varios intentos de cambiar la red neuronal para tratar de que funcionara mejor. Se intentó cambiar la estructura de la red, agregando capas, cambiando los tamaños de las capas de embedding y las capas escondidas, pero esto no causó ningún cambio en el comportamiento con densidades altas, donde la red seguía sin tener ningún incentivo mayor para agregar nodos. Se esperaba que cambiando el learning rate hubiera algún cambio mayor, ya que era posible que haciendo que la red se moviera más lento en la búsqueda de un óptimo local, pudiera encontrar óptimos locales con más precisión, pero sin importar el valor que se usara la búsqueda siempre se quedaba estancada en 0. También se probó cambiando los valores de puntaje que daba agregar un nodo a la solución y el penalty que daba agregar dos nodos que estuvieran conectados, pero cambiar esto tenía un efecto muy similar al de cambiar el learning rate, solamente se demoraba más tiempo en llegar a 0 y se quedaba estancado ahí. Hubo un esfuerzo durante el trabajo para tratar de hacer que esta red funcionara pero sin cambios importantes a la lógica detrás de la red, no parece haber forma de mejorar los resultados.

Con el uso del dataset más simple se es posible encontrar los puntos débiles del algoritmo para saber en qué es necesario trabajar. Estos resultados implican también que la red neuronal no debería encontrar respuestas relevantes para ninguno de los demás datasets, ya que hay muy pocos grafos en los que la densidad es menor a la cota que se encontró.

4.2. Máximo Corte

Para este problema, en el artículo original [68] se hizo testing en grafos regulares, que como se pudo apreciar, no son una buena opción para ver qué tan bueno es el algoritmo. Sin embargo, también se hizo testing sobre instancias públicamente conocidas y, a pesar de ser escaso, se pudo comprobar que la red neuronal no daba los resultados óptimos pero lograba acercarse. Sin embargo, para ese testing se eligió la estructura de la red neuronal por cada instancia de forma manual, cambiaban parámetros como el embedding y la cantidad de capas para cada instancia sin detallar como se decidía, por lo que parecían números escogidos al azar.

Para probar la red se escogieron tres configuraciones inspiradas en las estructuras usadas para grafos pequeños, medianos y grandes del testing mencionado anteriormente del trabajo

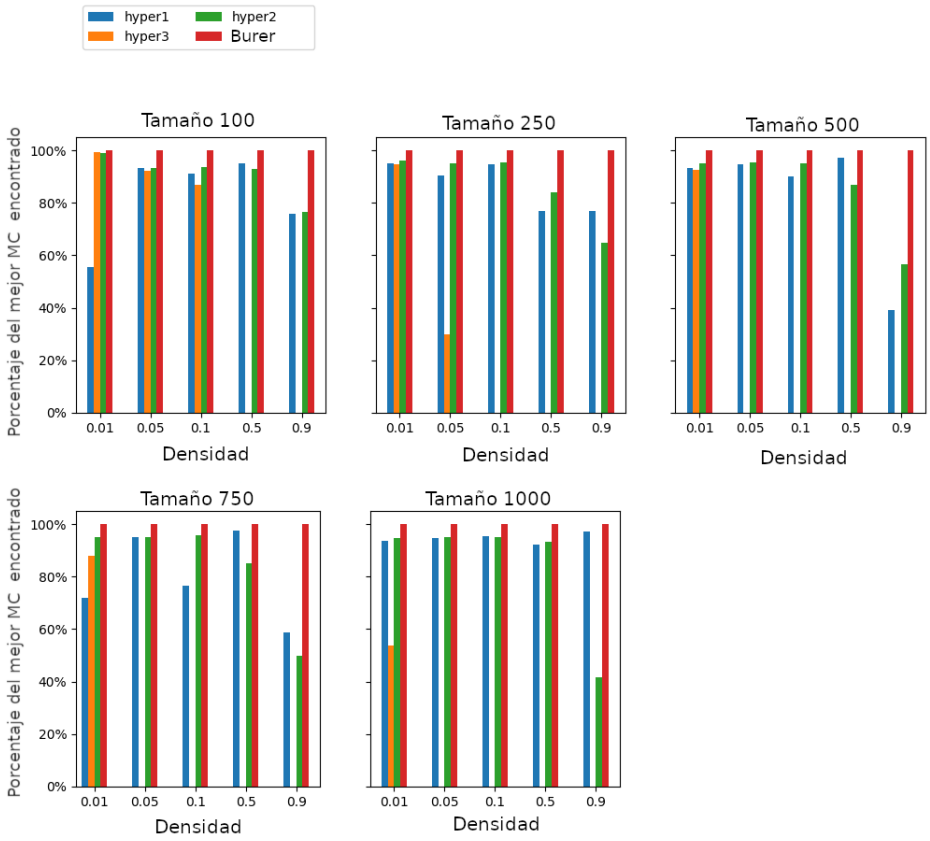

Figura 4.3: Gráficos de barra para las tres configuraciones de redes neuronales con "gb_1_100i" comparadas con la heurística de Burer et al [12] en grafos con pesos que van desde el 1 hasta el 100. Se decidió solo comparar con la mejor heurística para que los gráficos fueran más claros. El eje X es la densidad de los grafos y el eje Y es el porcentaje del máximo corte obtenido entre todos los algoritmos, agregado entre varias instancias. Hay múltiples gráficos para distintos tamaños de los grafos.

original y se le llamó hyper1, hyper2 y hyper3 respectivamente. En hyper1 solo se usó una capa de dimensión 5. En hyper2 se usaron dos capas de dimensión 1000. En hyper3 se usaron tres capas de dimensión 6000.

El artículo original [68] afirmaba que la red no podía tener pesos negativos, por lo tanto solo se pudo testear con "gb_1_10f", "gb_1_100i" y "Mannino", los tres datasets que no tenían aristas negativas. Se decidió correr todas las pruebas con el mismo límite de tiempo que los algoritmos clásicos, 100 segundos.

4.2.1. Datasets gb

Para el dataset "gb_1_100i" se pueden apreciar los resultados en la Tabla 4.2 y la Figura 4.3. Se puede apreciar que el algoritmo clásico de Burer et al [12] se desempeña mejor consistentemente frente a las heurísticas, sin embargo, la diferencia se hace más notable a medida que aumenta la densidad.

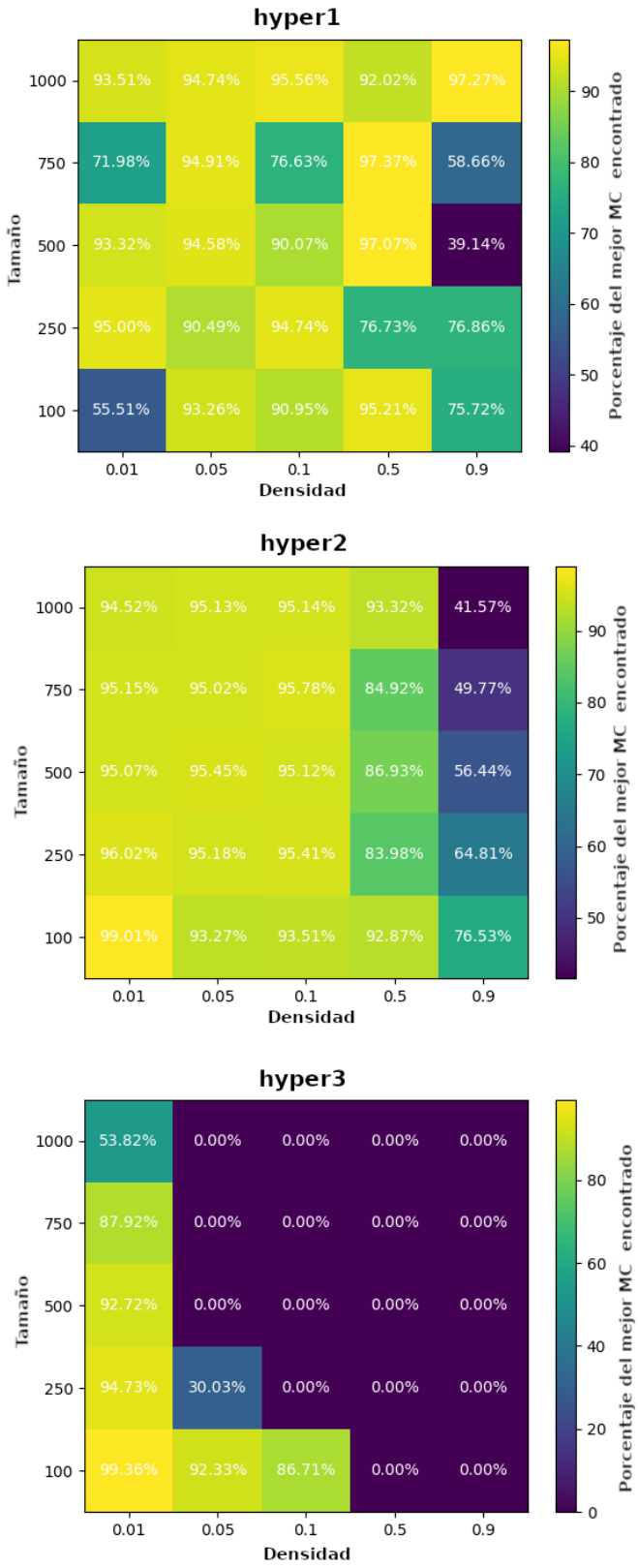

Figura 4.4: Mapas de calor para las configuraciones de hyper1, hyper 2 y hyper3 en "gb_1_100i". El eje X es la densidad de los grafos y el eje Y es el tamaño. El color representa el porcentaje que obtuvo del corte más grande entre todos los algoritmos.

Tabla 4.2: Resumen de los resultados en el dataset "gb_1_100i". Se calcula el promedio, la mediana y la desviación estándar del porcentaje de la mejor respuesta encontrada para todas las instancias en cada algoritmo. El dataset incluía grafos que variaban en densidad y tamaño, teniendo aristas de pesos enteros entre 1 y 100.

Algoritmo	**Promedio**	**Mediana**	**Desviación Estándar**
hyper1	82.25 %	93.26 %	14.99 %
hyper2	86.40 %	94.52 %	15.63 %
hyper3	25.51 %	0 %	39.37 %
Burer	99.99 %	100 %	0.01 %
Sousa	90.63 %	93.20 %	6.90 %
Duarte	99.89 %	100 %	0.25 %

Tabla 4.3: Resumen de los resultados en el dataset "gb_1_10f". Se calcula el promedio, la mediana y la desviación estándar del porcentaje de la mejor respuesta encontrada para todas las instancias en cada algoritmo. El dataset incluía grafos que variaban en densidad y tamaño, teniendo aristas de pesos entre 1 y 10, incluyendo decimales.

Algoritmo	**Promedio**	**Mediana**	**Desviación Estándar**
hyper1	87.57 %	92.83 %	11.70 %
hyper2	83.24 %	94.57 %	21.93 %
hyper3	24.60 %	0 %	39.58 %
Burer	99.99 %	100 %	0.01 %
Sousa	90.62 %	93.72 %	6.91 %
Duarte	99.88 %	100 %	0.25 %

Analizando el mapa de calor de hyper1 en la Figura 4.4 se puede ver que sus resultados son muy inconsistentes, tiene un buen desempeño con frecuencia pero es propensa a fallar. El desempeño de esta red no parece seguir ningún patrón con respecto al tamaño y la densidad, su desempeño no parece depender de ninguno de los dos parámetros.

Analizando el mapa de calor de hyper2 en la Figura 4.4, se puede ver que su desempeño es mucho más estable, obteniendo algo cercano a 95 % constantemente en densidades bajas, pero se puede notar claramente que con densidades altas las respuestas bajan considerablemente su calidad.

Analizando el mapa de calor de hyper3 en la Figura 4.4, se puede ver que no responde en gran parte de las instancias. Esto se debe a que el tiempo dado no fue suficiente para entrenar a la red de forma correcta, una red tan grande no se puede entrenar en 100 segundos. Por otro lado, analizando los resultados a los que sí se pudo llegar, se puede apreciar que esta red no es significativamente mejor que hyper2, por lo tanto no es una opción que valga tanto la pena seguir revisando.

Analizando los resultados del dataset "gb_1_10f" en la Tabla 4.3 y la Figura 4.5 se puede apreciar que son muy similares a los del dataset anterior. Con esto se puede concluir que no

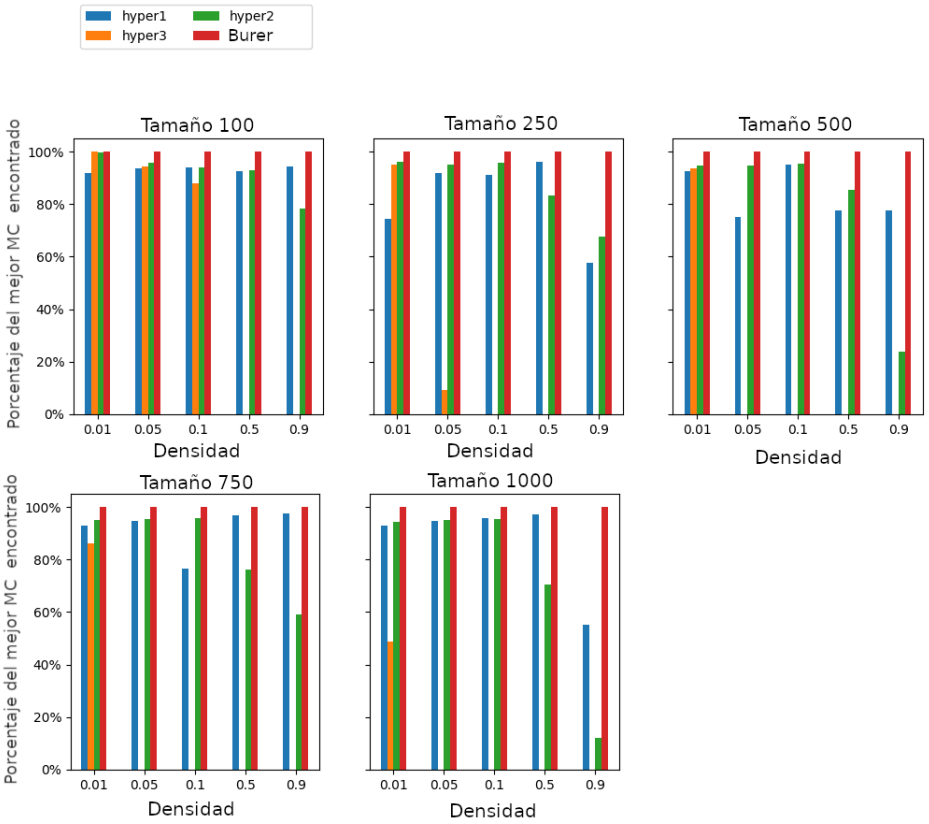

Figura 4.5: Gráficos de barra para las tres configuraciones de redes neuronales con "gb_1_10f" comparadas con la heurística de Burer et al [12] en grafos con pesos que van desde el 1 hasta el 100. Se decidió solo comparar con la mejor heurística para que los gráficos fueran más claros. El eje X es la densidad de los grafos y el eje Y es el porcentaje del máximo corte obtenido entre todos los algoritmos, agregado entre varias instancias. Hay múltiples gráficos para distintos tamaños de los grafos. Se puede apreciar su similitud con los resultados del dataset que solo tenía números enteros como pesos, la presencia de números flotantes parece no afectar la calidad de los algoritmos.

Tabla 4.4: Resultados en el dataset Mannino. Se calcula el porcentaje de la mejor respuesta dada entre todos los algoritmos.

file	hyper1	hyper2	hyper3	Burer	Duarte	Sousa
mannino_k48.mc	77.52 %	68.90 %	70.14 %	100 %	100 %	100 %
mannino_k487a.mc	69.95 %	79.74 %	8.56 %	100 %	99.86 %	94.35 %
mannino_k487b.mc	70.28 %	45.35 %	0.76 %	100 %	99.89 %	93.77 %
mannino_k487c.mc	77.53 %	75.53 %	16.31 %	100 %	99.94 %	92.36 %

hay gran diferencia entre enteros y números flotantes para esta red neuronal y no es necesario indagar más en las configuraciones para este caso.

En general, se puede ver un patrón con respecto a la calidad de los resultados en la mayoría de las configuraciones. Se puede notar que la densidad es un factor importante con respeto al desempeño de la red. El tamaño de igual forma pero no en tanta medida como la densidad. Es importante notar que puede que hayan más factores involucrados, el tamaño y la densidad son solo dos de los factores más fáciles de percibir ya que su cambio afecta directamente al tamaño del problema. En cuanto más grande es el problema, más complejo va a ser obtener una solución.

4.2.2. Dataset Mannino

Los resultados del test en que a todos los algoritmos se les dio 100 segundos para resolver los problemas se pueden ver en la Tabla 4.4, estos son los resultados sobre grafos que representan distintas mediciones de interferencias de frecuencias de radio. Se puede apreciar que el algoritmo de Burer et al [12] siempre obtuvo la mejor respuesta, las otras heurísticas estuvieron cerca pero las redes neuronales tuvieron un desempeño notablemente peor. Los problemas de desempeño en grafos de mayor densidad se hacen notorios en estas instancias. Las diferencias entre los algoritmos clásicos son mínimas en comparación con las diferencias con las redes neuronales.

4.3. Análisis

La gran diferencia que existe entre la misma red neuronal para distintos problemas es que para el problema del Máximo Corte con solo pesos positivos la solución en general solo aumenta cuando se agregan nodos, se tiene que buscar la configuración óptima para que estas ganancias sean positivas pero nunca es un óptimo no tomar nada porque siempre es positivo agregar las aristas, en cambio, para el problema del Máximo Conjunto Independiente al agregar un nodo la solución ya empieza a tomar en cuenta el posible penalty que recibiría de agregar nodos y prefiere no tomar nada. Se sospecha que si la red neuronal para el problema del Máximo Corte funcionara con aristas negativas, también tendería a no tomar ningún nodo.

Se puede apreciar lo fácil que es sacar conclusiones rápidas con estas visualizaciones de

los datos gracias al framework. Aunque estas no sean instancias difíciles del problema, son sumamente útiles para encaminar a la red neuronal en la dirección correcta y juzgar que tan bien van los resultados en sus primeros pasos. Como se pudo ver en la GNN para el Máximo Conjunto independiente, cambiar parámetros y la estructura de la red no era un factor determinante para la calidad de las soluciones, esto significaría que es necesario replantear la forma en que se aborda el problema para definitivamente conseguir mejores resultados, este enfoque tiende a quedarse pegado en óptimos locales. Sin embargo, para el problema del Máximo corte las soluciones, a pesar de su diferencia en calidad con los algoritmos clásicos, sí daban respuestas válidas y se espera que después de un ajuste en las configuraciones de la red, cambios de estructura o de parámetros, se pueda avanzar a una solución que usando la misma base obtenga mejores resultados.

Capítulo 5

Conclusión

5.1. Resultados

En este trabajo se recopiló información de dos problemas, el Máximo Set Independiente y el Corte Máximo, y con la información obtenida se construyeron dos instancias de un framework para poder testear nuevas GNNs y compararlas con algoritmos clásicos ya existentes. Se demostró la utilidad de esta herramienta al probarla con una GNN de prueba y ver que era posible guiar la investigación analizando los datos entregados por la herramienta. Los resultados del testing de esta GNN demostraron que no estaba a la altura de los algoritmos clásicos y aún queda bastante trabajo para que las GNN se usen de forma práctica para resolver estos problemas.

La recopilación de información fue exitosa, pero la implementación del framework dejó cosas que desear, como serían la calidad de los algoritmos clásicos usados como estándar para comparar los resultados, que obtuvieran los mismos resultados que aparecían en los artículos originales, y la falta de una interfaz gráfica que facilite el uso de la herramienta para el usuario. Sin embargo, la herramienta se hizo lo suficientemente extensible para permitir la fácil incorporación de nuevos algoritmos e instancias de prueba.

Con respecto a los objetivos de la memoria:

1. Se recopiló información de los algoritmos clásicos para ambos problemas de forma exitosa, se obtuvo una lista de algoritmos que representaban el estado del arte y se hizo un estudio a profundidad de su funcionamiento.
2. Se obtuvieron datasets de grafos para ambos problemas de forma exitosa, estos tomaban en cuenta múltiples casos sintéticos y aplicaciones reales de los problemas estudiados.
3. Se implementaron los algoritmos para el problema del Máximo Conjunto Independiente, pero no se logró conseguir que estos obtuvieran los mismos resultados que se obtenían en los artículos originales, aún así se pudieron evaluar y usar como punto de comparación. Para el problema del Máximo Corte se recopilaron los algoritmos que ya estaban implementados.

4. Se logró evaluar el funcionamiento de una GNN en comparación con los algoritmos que fueron implementados y se logró hacer un análisis de los resultados de forma exitosa.
5. Se logró organizar las partes para generar una herramienta que logra comparar un nuevo algoritmo con los algoritmos recopilados en esta memoria. Sin embargo, no se desarrolló una interfaz gráfica para la memoria, es solamente una herramienta de terminal.

5.2. Análisis

Gran parte del trabajo fue reimplementar algoritmos que ya se habían implementado pero que no estaban disponibles para el público, esto tomó mucho más trabajo del que se esperaba, incluso considerando que no se llegaron a los resultados óptimos en cuanto a las respuestas que se mencionaban en los trabajos originales. Si se hubiera distribuido mejor el tiempo, es probable que los algoritmos hubieran alcanzado los mismos resultados no perfectos, pero el resto de la herramienta hubiera estado en mejor estado. Escribir los algoritmos tal que se llegue a los mismos resultados original es una labor que requiere de mucho más trabajo y análisis en los detalles finales, una vez implementada la herramienta es necesario reajustar parámetros y optimizar detalles que solo se pueden arreglar con un análisis profundo del algoritmo. Fue bastante ambicioso querer implementar varios algoritmos y se sufrieron las consecuencias de ese error.

El trabajo realizado podría ser útil para los primeros pasos de la construcción de una GNN, tecnología que está muy de moda en el ámbito científico, con sus más básicos análisis es posible encontrar rápidamente problemas claros que podría tener el trabajo. Sin embargo, la herramienta no ofrece mucha ayuda en el análisis más avanzado, solo entrega datasets útiles para su evaluación, pero no da mucha accesibilidad a ellos. Sin embargo, esta ayuda inicial podría ser beneficiosa para muchas personas al poder testear rápido si una idea de GNN es buena o no, y para dónde guiar la investigación. Sin embargo, el framework no es exclusivo para GNNs, no se aprovecha de ninguna de sus características, solamente el hecho de que usa grafos, pero es posible medir cualquier algoritmo con el framework a pesar de que está pensado para ser usado con redes neuronales. No se tiene una idea clara de la forma en que se podría haber utilizado este factor más que para guiar el desarrollo de las utilidades que se quería proveer.

5.3. Trabajo futuro

Un problema que se tuvo con mucha frecuencia en el planteamiento de este trabajo, fue que no se tenía claro como se podría utilizar un gran rango de grafos para analizar el desempeño de la GNN. Con una gran cantidad de características es muy difícil analizar como se relacionan los resultados con cada una. Sin embargo, uno de los trabajos mencionados tiene una solución muy interesante: Dunning et al [20] para hacer machine learning sobre los grafos al querer elegir la mejor heurística codifica los grafos en una lista de decenas de características, en base a esas características construye un árbol de decisión para encontrar la mejor heurística. Sería posible que en base a los resultados del testing de una GNN, se

construya algo similar a ese árbol de decisión para evaluar bajo que características la GNN es más débil.

Bibliografía

[1] Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5(1):29–31, December 2022.

[2] Francisco Barahona. Network design using cut inequalities. *Siam Journal on Optimization - SIAM J OPTIMIZATION*, 6, 08 1996.

[3] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 05 1988.

[4] Trinayan Baruah, Kaustubh Shivdikar, Shi Dong, Yifan Sun, Saiful A Mojumder, Kihoon Jung, José L. Abellán, Yash Ukidave, Ajay Joshi, John Kim, and David Kaeli. Gnnmark: A benchmark suite to characterize graph neural network training on gpus. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 13–23, 2021.

[5] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon, 2020.

[6] P. Berman and A. Pelc. Distributed probabilistic fault diagnosis for multiprocessor systems. In *[1990] Digest of Papers. Fault-Tolerant Computing: 20th International Symposium*, pages 340–346, 1990.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, NY, 2016.

[8] G.A. Bodino. *Economic Applications of the Theory of Graphs*. Tracts on mathematics and its applications. Gordon and Breach, Science Publishers, 1962.

[9] Thorsten Bonato, Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. Lifting and separation procedures for the cut polytope. *Mathematical Programming*, 146(1):351–378, Aug 2014.

[10] Mark G. Brockington and Joseph C. Culberson. Camouflaging independent sets in quasi-random graphs. In *Cliques, Coloring, and Satisfiability*, 1993.

[11] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[12] Samuel Burer, Renato Monteiro, and Yin Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12, 07 2001.

[13] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9):1672–1696, 2011.

[14] Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses, 2013.

[15] K. Corrádi and Sándor Szabó. A combinatorial approach for keller's conjecture. *Periodica Mathematica Hungarica*, 21:95–100, 1990.

[16] Gabor Pataki David Johnson. Seventh dimacs implementation challenge, 2000. `https://www.dimacs.rutgers.edu/archive/Challenges/Seventh/`.

[17] Samuel de Sousa, Yll Haxhimusa, and Walter G. Kropatsch. Estimation of distribution algorithm for the max-cut problem. In Walter G. Kropatsch, Nicole M. Artner, Yll Haxhimusa, and Xiaoyi Jiang, editors, *Graph-Based Representations in Pattern Recognition*, pages 244–253, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Dover Books on Mathematics. Dover Publications, 2016.

[19] Abraham Duarte, Angel Sanchez, Felipe Fernández, and Raúl Cabido. A low-level hybridization between memetic algorithm and vns for the max-cut problem. pages 999–1006, 06 2005.

[20] Iain Dunning, Swati Gupta, and John Silberholz. What works best when? a systematic evaluation of heuristics for max-cut and qubo. *INFORMS Journal on Computing*, 30:608–624, 08 2018.

[21] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints, 2015.

[22] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2022.

[23] M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*, volume 148 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[24] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation, 2019.

[25] Thomas Feo and Mauricio Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 03 1995.

[26] Paola Festa, Panos Pardalos, and Mauricio Resende. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 17, 07 2002.

[27] Victor Fung, Jiaxin Zhang, Eric Juarez, and Bobby G. Sumpter. Benchmarking graph neural networks for materials chemistry. *npj Computational Materials*, 7:1–8, 2021.

[28] M. R. Garey and D. S. Johnson. " strong " np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, jul 1978.

[29] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[30] Fred Glover. *Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges*, pages 1–75. Springer US, Boston, MA, 1997.

[31] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42:1115–1145, 1995.

[32] Pierre Hansen and Nenad Mladenović. *Developments of Variable Neighborhood Search*, pages 415–439. Springer US, Boston, MA, 2002.

[33] Eric Harley, Anthony Bonner, and Nathan Goodman. Uniform integration of genome mapping data using intersection graphs. *Bioinformatics (Oxford, England)*, 17:487–94, 07 2001.

[34] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[35] Xin Huang, Laks V.S. Lakshmanan, and Jianliang Xu. *Community Search over Big Graphs*. Morgan & Claypool Publishers, 2019.

[36] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2019.

[37] Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, 37:20–33, 2015.

[38] David Johnson, Cecilia Aragon, Lyle McGeoch, and Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39:378–406, 06 1991.

[39] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[40] D.J. Johnson, D.S. Johnson, M.A. Trick, and DIMACS (Group). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*. Center for Discrete Mathematics and Theoretical Computer Science New Brunswick, NJ: DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Society, 1996.

[41] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning tsp requires rethinking generalization. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[42] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[43] Manuel Laguna, Abraham Duarte, and Rafael Marti. Hybridizing the cross-entropy method: An application to the max-cut problem. *Computers Operations Research*, 36:487–498, 02 2009.

[44] Frauke Liers, Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. *Computing Exact Ground States of Hard Ising Spin Glass Problems by Branch-and-Cut*, chapter 4, pages 47–69. John Wiley Sons, Ltd, 2004.

[45] F. Jessie MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. 1977.

[46] Seiji Maekawa, Koki Noda, Yuya Sasaki, and Makoto Onizuka. Beyond real-world benchmark datasets: An empirical study of node classification with gnns. *ArXiv*, abs/2206.09144, 2022.

[47] Sven Mallach. Maxcut and bqp instance library. `http://bqp.cs.uni-bonn.de/library/html/index.html`, 2021 (accessed January 20, 2024).

[48] Rafael Marti, Abraham Duarte, and Manuel Laguna. Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, 21:26–38, 02 2009.

[49] Franco Mascia. Bhoslib benchmark set. `https://iridia.ulb.ac.be/~fmascia/maximum_clique/BHOSLIB-benchmark`, (accessed January 20, 2024).

[50] Franco Mascia. Dimacs benchmark set. `https://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`, (accessed January 20, 2024).

[51] Rajesh Matai, Surya Singh, and M.L. Mittal. *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*. 2010.

[52] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey, 2020.

[53] Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[54] Sadman Sadeed Omee, Steph-Yves M. Louis, Nihang Fu, Lai Wei, Sourin Dey, Rongzhi Dong, Qinyang Li, and Jianjun Hu. Scalable deeper graph neural networks for high-performance materials property prediction. *Patterns*, 3, 2021.

[55] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. Graphworld: Fake graphs bring real insights for gnns. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22. ACM, August 2022.

[56] Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. 01 1982.

[57] Yun Peng, Byron Choi, and Jianliang Xu. Graph embedding for combinatorial optimization: A survey. *CoRR*, abs/2008.12646, 2020.

[58] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[59] W. Pullan and H. H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, February 2006.

[60] Wayne Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12, 01 2006.

[61] Wayne Pullan, Franco Mascia, and Mauro Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 17:181–199, 04 2010.

[62] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. *Math. Programming*, 121(2):307, 2010.

[63] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI 2007: Advances in Artificial Intelligence*, pages 412–426, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[64] Giovanni Rinaldi. Rudy, 1998. `http://www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz`.

[65] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[66] Laura A. Sanchis. Test case construction for the vertex cover problem. In *Computational Support for Discrete Mathematics*, 1992.

[67] Stefan H. Schmieta. The dimacs library of mixed semidefinite-quadratic-linear programs. `http://archive.dimacs.rutgers.edu/Challenges/Seventh/Instances/`, (accessed January 20, 2024).

[68] Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, April 2022.

[69] Neil Sloane. Challenge problems: Independent sets in graphs. `https://oeis.org/A265032/a265032.html`, 2000 (accessed January 20, 2024).

[70] Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip Kim. Fast and flexible protein design using deep graph neural networks. *Cell Systems*, 11, 09 2020.

[71] R.J. Trudeau. *Introduction to Graph Theory*. Dover Books on Mathematics. Dover Publications, 2013.

[72] Ragav Venkatesan and Baoxin Li. *Convolutional neural networks in visual computing*. Data-Enabled Engineering. Routledge, London, England, October 2017.

[73] Natalia Vesselinova, Rebecca Steinert, Daniel F. Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8:120388–120416, 2020.

[74] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[75] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[76] Zhongnan Zhang Weili Wu. *Combinatorial Optimization and Applications*. 2020.

[77] R.J. Wilson. *Introduction to Graph Theory*. Longman, 2010.

[78] Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 26, 07 2013.

[79] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[80] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021.

[81] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8):514–534, 2007.

[82] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18. ACM, July 2018.

[83] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, 10 2019.

[84] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 353–362, New York, NY, USA, 2016. Association for Computing Machinery.