Claudia Mairal Buera

El problema del máximo conjunto independiente en un grafo

Universidad de Zaragoza

Director del trabajo: Alfredo Martín García Olaverri

14 de junio de 2023

Summary

In graph theory, an independent set in an undirected graph $G = (V, E)$ is a subset of vertex S of V such that for every two vertices in S, there is no edge connecting the two. In this dissertation, we will solve the problem of finding the maximum independent set for a given graph. The main purpose of this dissertation resides in the description of the maximum independent set problem and its computational complexity, as well as the theoretical results related to it and the known algorithms for its resolution for general graphs and some particular classes of graphs.

The dissertation consists of 4 chapters. We will briefly describe each of them.

First of all, we will bring up some basic definitions and concepts of graph theory used throughout the document such as graph, subgraph, adjacent vertex, incident vertex, complement of a graph, cycle, path, neighbourhood of a vertex, vertex cover, independent set, clique, bipartite graph, matching,... Furthermore, we will explain the maximum independent set problem and its equivalences with the minimum vertex cover and the maximum clique problem. Finally, we will mention some some real life applications and, particularly, we will give an example of an activity selection problem or activities that present conflicts due to the simultaneous use of resources. This kind of problems that are modeled by incompatibility graphs, in which two vertices are connected if they are not mutually compatible, are common in the maximum independent set problem and are used in many fields and applications.

In the next chapter, we will focus on the computational complexity theory. First, we will introduce some of the concepts and notations related to it such as the computational complexity of an algorithm, the big-O notation, decision problems or P and NP complexity classes. Next, we will prove the NP-Completeness of the maximum independent set problem. Given the NP-Completeness of Boolean satisfiability problem we will prove the NP-Completeness of Boolean 3-satisfiability problem which we will use to prove that the minimum vertex cover problem is NP-Complete too. This result will be equivalent to proving the NP-Completeness of maximum independent set. It is an important task to study the NP-Completeness of a problem, since if some NP-complete problem had an algorithm that would solve it in polynomial time, then all problems of the NP class would also have it. Unfortunately no such algorithm could be found so far for any NP-complete problem. Even worse, no one has so far been able to demonstrate that such algorithm cannot exist and and this is still one of the great unanswered questions in mathematics.

In chapter 3 we will introduce some theoretical results related to the maximum independent set problem: König's Theorem on bipartite graphs and the Erdos-Szekeres Theorem on complete subgraphs. Although the main body of this dissertation is focused on solving the maximum independent set problem for all types of graphs, the first of them provides us with a method to find the maximum independent set in a bipartite graph. König's Theorem theorem states that in any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover. Finding a maximum matching in a bipartite graph can be treated as a network flow problem. The solution of the the problem of calculating the maximum flow in a graph will give a solution for maximum matching problem for bipartite graphs and, thus, a solution for minimum vertex cover for bipartite graphs. We can say that, in bipartite graphs, the resolution of the NP-Hard problem of the maximum independent set can be reduced to solving the polynomial problem of maximum matching for bipartite graphs.

In the second part of the chapter we will focus on a theoretical result from the area of Ramsey theory related to this problem: the Erdos-Szekeres Theorem. This theorem states that for any $s, t \geq 2$ there is $R(s,t) < \infty$, called *Ramsey number*, such that any graph on $R(s,t)$ vertices contains either an independent

set of size s or a clique of size t. Besides,

$$R(s,t) \leq \binom{s+t-2}{s-1} \quad \wedge \quad R(s,t) \leq R(s-1,t)+R(s,t-1)$$

Finally, in chapter 4 we present some exact and heuristic algorithms for the maximum independent set problem, as well as a polynomial algorithm for the particular case of interval graphs.

We start by giving some insight about the exact resolution of these problem throughout history. We present the most obvious algorithm, brute force algorithm that finds a maximum independent set for a graph with n vertices in time $O^*(2^n)$. We mention some of the first algorithms that managed to break the trivial bound, such as Tarjan and Trojanowski(1976) who found an algorithm that could reduce the time to $O^*(2^{\frac{n}{3}})$ or Robson (1986) who reduced it to $O^*(2^{0,296n})$. However, in this part the algorithm that we will be interested in and that will be explained in detail will be Formin's algorithm (2006). This algorithm applies "Measure and Conquer" approach to the analysis of a very simple backtracking algorithm solving the well-studied maximum independent set problem. Fistly, we will explain some reduction rules used in the description of the algorithm such as *folding*, *mirroring* or *dominance*. After this, we can give a step-by-step explanation of the algorithm. Finally, we will proceed to carry out an exhaustive analysis of the asymptotic time complexity of the algorithm. In the worst case, the result of the analysis reaches a complexity of $O^*(2^{0,406 \cdot n})$, being able to carry out a more refined and complicated analysis reaching a complexity of $O^*(2^{0,288 \cdot n})$. This is competitive with the current best time bounds obtained with far more complicated algorithms.

In general, the maximum independent set problem cannot be approximated to a constant factor in polynomial time (unless P = NP). This is an essential question that we will deal with in the second part of the chapter along with some definitions and concepts about approximation algorithms and heuristic algorithms. Afterwards, we will explain one of the most basic heuristic algorithms for the maximum independent set problem. Finally, we give an integer linear programming formulation of the problem. Many optimization graph problems can be expressed in this way. Although at first it may seem that expressing it as an integer linear problem can lead to an easier, and even polynomial, resolution, this configuration is the same or more complex in time than the previous ones.

Throughout this chapter we have seen numerous algorithms to solve the maximum independent set problem in general graphs. Even though finding a maximum independent set is NP-Hard in general graphs, for some types of graphs such as interval graphs the maximum independent set problem can be solved in polynomial time. Therefore, in this part we give some definitions and results related to interval graphs. We present and prove an exact linear algorithm for the resolution of the maximum independent set in this particular class of graphs. To end this chapter we study a very special case of the maximum independent set problem where the instances of the problem are limited to interval graphs, the *scheduling* problems. These are very common problems in real life where we are assigned a set of resources, each interval represents a request for a resource for a specific period of time and we want to find a maximum set of requests that can be satisfied without interfering with each other. A real life example about the filming time of an actor's movies is given.

Índice general

Capítulo 1

Introducción

A lo largo de este capítulo primero presentaremos formalmente algunas definiciones y notaciones utilizadas en este trabajo. A continuación, enunciaremos el problema del máximo cojunto independiente de un grafo así como problemas de grafos equivalentes a este. Finalmente, mencionaremos algunas de sus aplicaciones.

1.1. Definiciones básicas

Un **grafo no dirigido** es un par $G = (V,E)$ donde V es un conjunto finito de vértices y E es un conjunto de ejes donde cada eje es un par no ordenado de vértices $e = (v,u)$, $v,u \in V$. Un vértice v es **incidente** al eje e si $v \in e$ y dos vértices v,u son **adyacentes** si son incidentes al mismo eje. El **orden de un grafo** es su número de vértices, es decir $n = |V|$. El **tamaño de un grafo** es su número de ejes, es decir $m = |E|$. El **grado** $d(v)$ de un vértice v es el número de ejes incidentes en él. Un **camino** es una sucesión de vértices y ejes, que empieza y termina en vértices, tal que cada vértice es incidente con el eje que le sigue y el que le precede en la secuencia, es decir, un camino de $v \in V$ a $u \in V$ es una secuencia $< v_1, v_2, ..., v_k >$ tal que $v = v_1$, $u = v_k$, y $(v_{i-1}, v_i) \in E$, para $i = 2, ..., k$. Un **ciclo** es un camino donde el vértice inicial coincide con el vértice final. Para cada $v \in V$, el conjunto de **vecinos de** v se denota $N(v) = \{u \in V |\ u$ es adyacente a $v\}$ y $N[v] = v \cup N(v)$. De igual manera, sea $S \subseteq V$ un subconjunto de vértices de G, el conjunto de **vecinos de** S en G denotado por $N_G(S)$, corresponde al conjunto de vértices adyacentes a vértices de S. El **grafo complementario** de $G = (V,E)$, es un grafo $\bar{G} = (V,\bar{E})$ donde $\bar{E} = \{(v,w) | v,w \in V,\ v \neq w,$ y $(v,w) \notin E\}$. El **subgrafo inducido** en G por $S \subset V$ subconjunto de vértices de G es el grafo $G[S]$ cuyo conjunto de vértices es S y cuyo conjunto de ejes son todas las aristas en E incidentes en ambos extremos a vértices de S. Un grafo $G = (V,E)$ es **conexo**, cuando $\forall i, j \in V$ existe un camino de i a j. Un subgrafo maximal y conexo de G se denomina **componente conexa**. Un grafo **completo**, K_n, es un grafo no dirigido con n vértices, y con todos los ejes salvo los bucles (es decir, con $\frac{n(n-1)}{2}$ ejes).

Definición. Un **recubrimiento de vértices** es un subconjunto B de vértices de $G = (V;E)$, $B \in V$, tal que todo eje de G posee al menos uno de sus extremos en B.

Definición. Un **clique** K del grafo G es un subconjunto de vértices $K \in V$ tal que el subgrafo inducido por K es completo, es decir, K es un subconjuntos de vértices, todos adyacentes entre sí.

Definición. Un **conjunto independiente** de G es un subconjunto S de V tal que no existe ningún eje que tenga ambos extremos en S, es decir, un conjunto de vértices $S \in V$ tal que ninguno de sus vértices es adyacente a otro.

Definición. Un **matching** (o emparejamiento) de un grafo G es un conjunto de ejes $K \subset E$ con extremos disjuntos. Este matching se dice que es un **máximo matching** si no existe otro matching con más ejes.

Definición. Un grafo $G = (V;E)$ es **bipartito**, si V puede particionarse en dos partes no vacías V_1 y V_2, de manera que todos los ejes tienen un extremo en V_1 y el otro en V_2. En un grafo bipartito $G = (V,E)$ con conjuntos de vértices V_1 y V_2, un matching es un conjunto de ejes $M \in E$ tal que todo eje $(i, j) \in M$ tiene un extremo $i \in V_1$ y otro $j \in V_2$ y no existe ningún otro eje en M con extremos en i ó j.

1.2. Descripción del problema

Ya hemos visto en que consiste el concepto de conjunto independiente, veamos ahora en que consiste el problema del máximo conjunto independiente que nos ocupa en este trabajo, así como otros problemas equivalentes.

Dado el grafo $G = (V,E)$ el **problema del máximo conjunto independiente** que denotaremos MIS (*Maximum Independent Set*) consiste en encontrar un conjunto independiente S del tamaño más grande posible para el grafo G, es decir, un subconjunto $S \in V$ de cardinalidad máxima de vértices no adyacentes por pares. El número de elementos de S o **número de independencia** de G se denota por $\alpha(G)$.

Para poder entender mejor el problema vamos a dibujar un grafo y su máximo conjunto independiente. Sea $G = (V,E)$ el grafo representado en la figura 1.1.

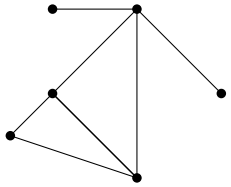

Figura 1.1: Grafo G

El máximo conjunto independiente de G es el conjunto de vértices $S \in V$ coloreado en rojo en la figura 1.2 pues claramente ningún eje de E tiene ambos extremos en S y no podemos encontrar uno que sea de cardinalidad mayor, ya que cualquier subconjunto de V de cuatro vértices tendrá al menos dos vértices adyacentes, entonces $\alpha(G) = 3$. Es importante diferenciar este problema del de encontrar un conjunto independiente maximal, esto es, un conjunto independiente tal que añadiendo cualquier otro vértice al conjunto, este deja de ser independiente. Un grafo puede tener muchos conjuntos independientes maximales de tamaños variados, pero solo uno, el más grande (o varios igual de grandes), corresponde con el conjunto independiente máximo. En el grafo G el máximo conjunto independiente $S \in V$ no es el único conjunto independiente maximal, también lo es el conjunto $M \in S$ en azul (véase 1.2).

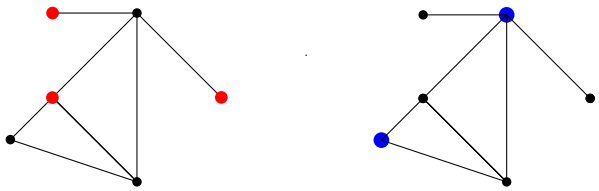

Figura 1.2: Conjunto independiente máximo $S \in V$ en el centro. Conjunto independiente maximal $M \in V$ a la derecha

Las siguientes relaciones entre conjuntos independientes, cliques y recubrimiento de vértices nos permiten relacionar este problema con otros ampliamente estudiados en teoría de grafos como el problema del máximo clique(lo denotaremos CLIQUE) o el problema de recubrimiento mínimo (lo denotaremos VERTEX COVER) Dado el grafo $G = (V,E)$ y $S \in V$ las siguientes afirmaciones son equivalentes:

a) $S \in V$ es un conjunto independiente de G

b) $V - S$ es un recubrimiento de vértices de G

c) $V-S$ es un clique en el grafo complementario $\bar{G}$ de G.

a) $\rightarrow$ b): Si $V-S$ no es un recubrimiento de vértices de G, existirá un eje (u,v) con $u,v \notin V-S$. En tal caso, $u,v \in S$ necesariamente. Esto contradice con que S sea un conjunto independiente.

a) $\leftarrow$ b): Si S no es un conjunto independiente de G, existirá un eje (u,v) con $u,v \in S$. En tal caso, $u,v \notin V-S$ necesariamente. Esto contradice con que $V-S$ sea un recubrimiento de vértices.

a) $\leftrightarrow$ c) : Por definición de grafo complementario, $(u,v) \in E \leftrightarrow (u,v) \notin \bar{E}$.

Así vemos que, estos tres problemas podrían considerarse simplemente como formas diferentes de ver el mismo problema. Además, estas relaciones permiten tranformar fácilmente cualquiera de los problemas en cualquiera de los otros.

En la figura 1.2 podemos ver la equivalencia de estos tres problemas, a la izquierda vemos el grafo complementario $\bar{G}$ al grafo G de la figura1.1, seguido del máximo clique K_3 del grafo complementario $\bar{G}$ que corresponde con el grafo inducido por el máximo conjunto independiente encontrado en 1.2. Finalmente, a la derecha el mínimo recubrimiento de vértices de G que es lo mismo que el complementario del conjunto independiente anterior $V-\bar{S}$.

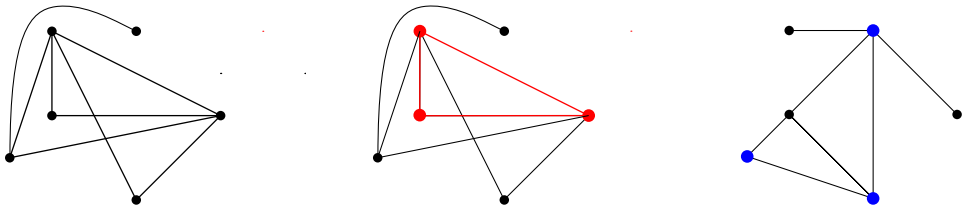

Figura 1.3: Grafo complementario $\bar{G}$ a la izquierda. Máximo clique K_3 de $\bar{G}$ en el centro. Recubrimiento mínimo $V-\bar{S}$ de G a la derecha

A pesar del hecho de que el problema MIS tiene su propia utilidad de manera idependiente, en numerosas ocasiones usaremos esta relación entre los problemas MIS, VERTEX COVER y CLIQUE para probar resultados de completitud de NP y estudiar sus propiedades o algoritmos. Esto se debe a que los resultados que se cumplen para MIS se cumplirán para el resto.

1.3. Aplicaciones

El problema del conjunto independiente máximo y sus equivalentes están involucrados en probar la complejidad computacional de muchos problemas teóricos como veremos en el siguiente capítulo. Asimismo son problemas de optimización que se encuentran presentes en la vida real en numerosos ámbitos diferentes como la genética, la biología molecular, la bioinformatica, la economía, la localización industrial, la teoría de códigos, el reconocimiento de patrones, el tratamiento de la información, las redes inalámbricas, la planificación y organización y numerosos otros ámbitos en los que aparecen problemas de optimización de este tipo. Es por ello que su modelización y resolución tiene tantas aplicaciones diversas. Por lo general, para resolverlo se trata de expresar el problema real en cuestión como un modelo máximo conjunto independiente para que puedan usar los métodos conocidos para resolverlo.

A continuación vamos a estudiar un ejemplo de esas aplicaciones para el caso de planificar y organizar los horarios de un torneo de deportes de una universidad.

Suponemos que la Universidad de Zaragoza quiere organizar una jornada de deportes universitaria y necesita programar un partido de una hora por cada modalidad deportiva. Cada equipo de estudiantes puede apuntarse a una o más modalidades, por lo que dos partidos pueden jugarse simultaneamente si ningún equipo se ha apuntado a dos de ellas. Con las intalaciones de la universidad el espacio para jugar simultáneamente no es un problema. La directiva de la universidad quiere que el viernes a las 3 p.m se

juegen el mayor número de partidos posible, pues justo a esa hora habrá una jornada de puertas abiertas para posibles patrocinadores de eventos deportivos, y a la universidad le gustaría tener sus pistas llenas.

La forma de resolver este problema es modelarlo como un problema de conjunto independiente máximo en el que cada deporte es un vértice, y dos vértices están unidos por un eje si hay al menos un equipo que está apuntado en ambas modalidades. Programar el número máximo de partidos para el viernes a las 3 p.m corresponde a encontrar el máximo conjunto independiente en este grafo.

A continuación dibujamos el grafo $G = (V,E)$ correspondiente a este problema, en él cada vértice representa un deporte marcado con su abreviación (10 modalidades distintas en total) y cada eje muestra si hay al menos un equipo apuntado a los dos deportes a los que une. El conjunto de vértices es:

$$\begin{aligned} V = \{&\text{Tenis}(T),\ \text{Fútbol sala}(FS),\ \text{Balonmano}(BM),\ \text{Baloncesto}(B),\\ &\text{Volleyball}(V),\ \text{Rugby}(R),\ \text{Fútbol 7}(F7),\ \text{Fútbol 11}(F11),\\ &\text{Beísbol}(BE),\ \text{Hockey hierba}(H),\ \text{Frisbee}(F),\ \text{Atletismo}(A)\} \end{aligned}$$

Hallamos el máximo conjunto independiente

$$S_5 = \{T,B,F,F7,A\} \in V$$

de G que se muestra en rojo en la figura 1.4.

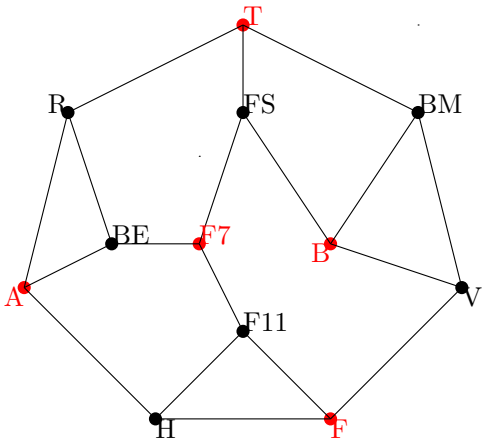

Figura 1.4: Grafo G, en rojo $S_5 \in V$ máximo conjunto independiente de G

Resolver el problema del conjunto independiente máximo este grafo, nos permite encontrar un conjunto de deportes tal que se puedan programar el máximo número de partidos a las 3 p.m. del viernes sin que ningún equipo tenga que jugar simultáneamente a varias modalidades. Esta consiste en programar los eventos de baloncesto, fútbol 7, tenis, frisbee y atletismo el viernes a las 3 p.m. para concentrar la mayor multitud de personas en las pistas a esa hora.

Este es un ejemplo de problema de programación de actividades que presentan conflictos por utilización simultánea de recursos. Este tipo de problemas que se modelan mediante grafos de incompatibilidad, en los cuales dos vértices están conectados si no son mutuamente compatibles, son comunes en el problema del máximo conjunto independiente y se utilizan en muchos ámbitos y aplicaciones.

Capítulo 2

Complejidad del problema del máximo conjunto independiente de un grafo

Cuando hablamos de algoritmos para resolver problemas de grafos debemos hablar de la teoría de complejidad computacional, especialmente cuando tratamos con grafos con numerosos vértices y ejes como en este caso. En este capítulo primero introduciremos algunas de estas ideas y luego probaremos la NP-Completitud del problema del máximo conjunto independiente.

2.1. Complejidad computacional

Definición. Un algoritmo es un proceso o conjunto ordenado y finito de instrucciones a seguir en los cálculos u otras operaciones de resolución de un problema. Los algoritmos requieren de una entrada o input, que proporciona la información sobre el estado inicial del problema, y, tras la ejecución de las intrucciones, presenta una salida o output con el resultado obtenido.

Para medir la complejidad computacional de un problema, se busca saber cuánto tiempo requiere resolverlo usando el mejor algoritmo que se conoce para ello. El tiempo de ejecución depende también del tamaño del problema, para grafos más grandes y con mayor número de ejes y vértices el problema tardará más en resolverse. Por lo tanto, para medir la complejidad en tiempo del algoritmo se calcula como crece el número de operaciones elementales, en el peor de los casos, según el tamaño de entrada del problema. En este trabajo trabajamos con problemas de grafos en los que el input será la información del grafo $G = (V,E)$, con $n = |V|$ y $m = |E|$ y para los cuales mediremos el tiempo de resolución mediante pasos, considerando cada paso una operación elemental, es decir, una suma, resta, multiplicación, división o comparación entre dos número. Finalmente, está expresión de complejidad del algoritmo se presentará mediante una función que dependerá de n y m.

Para clasificar la complejidad computacional se utilizan los límites superior e inferior del máximo tiempo requerido para resolver el problema con el algoritmo más eficiente conocido. Los límites superior e inferior generalmente se expresan de manera asintótica usando la notación **O-grande**, que oculta factores constantes y términos más pequeños:

Definición. Diremos que $f(n) = O(g(n))$ si $\exists c > 0$ y $\exists n_0$ tal que

$$\forall n \geq n_0 \quad 0 \leq f(n) \leq c \cdot g(n)$$

Por ejemplo, un problema de tamaño n de complejidad $T(n) = 7n^2 + 15n + 40$, en notación O-grande se escribiría $T(n) \in O(n^2)$.

De la misma forma que la notación O-grande ignora valores constantes la notación **O^*-grande** ignora factores polinomiales, es decir $f(n) = O^*(g(n))$ si existe algún polinomio p y $\exists n_0$ tal que

$$\forall n \geq n_0 \quad f(n) \leq p(n) \cdot g(n)$$

Esta notación se usa, sobretodo, para comparar funciones que crecen exponencialmente, en cuyo caso los términos polinomiales son menos importantes. Por ejemplo, un problema de tamaño n y de complejidad $O(2^n n^2)$ se escribiría $O^*(2^n)$ en notación O^*-grande.

Dentro de esta clasificación tenemos los problemas de complejidad polinomial **P** (*polynomial time*) y los problemas de tiempo polinomial no determinista **NP** (*non-deterministic polynomial-time*).

Definición. Diremos que un algorimo es **polinomial** si, dado un problema de entrada de tamaño n, existe $k > 0$ tal que el número de operaciones elementales o pasos que realiza para resolverlo está acotado por un polinomio que depende del tamaño de entrada, es decir, si resuelve el problema en tiempo $O(n^k)$.

Definición. Diremos que un problema es polinomial, es decir, pertenece a la clase **P** (*polynomial time*) si existe un algoritmo polinomial que lo resuelve.

Definición. Dados A y B dos problemas, decimos que A es **reducible** a B en tiempo polinomial, si existe un algoritmo de tiempo polinomial para transformar las entradas α del problema A en entradas β del problema B , de modo que el resultado de B con la entrada β nos proporcione un resultado para A con la entrada α.

Dentro de la teoría de complejidad, existen problemas con carácterísticas similares que podemos agrupar.

Definición. Los **problemas de optimización** son aquellos que buscan minimizar o maximizar (es decir, optimizar) el valor de una solución en un grupo de soluciones generadas para una entrada específica. El problema del máximo conjunto independiente de un grafo es un problema de optimización discreta en el cuál se busca encontrar dicho conjunto máximo entre un conjunto finito de conjuntos independientes posibles.

Definición. Los **problemas de decisión** son aquellos donde dada una entrada específica del problema se busca una salida de "Sí" ó "No".

Cualquier problema de optimización puede ser manejado como un problema de decisión incluyendo un valor objetivo K para una entrada n del problema y preguntando si existe o no existe una solución factible en el conjunto de soluciones $F(n)$, con el valor de la solución $c(s)$ optimizado sobre K, para el caso de maximización sería $c(s) \geq K$. El problema del máximo conjunto independiente puede tratarse entonces conmo un problema de decisión si dado un grafo $G = (V,E)$ y un entero $K \in \mathbb{N}$ nos hacemos la pregunta ¿existe un conjunto de vértices no adyacentes por pares $I \subseteq V$ de tamaño $|I| \geq K$?

Dentro de la teoría de complejidad será más conveniente tratar con este tipo de problemas de decisión que con problemas de optimización. De esta manera, podemos definir la clase de problemas **NP**:

Definición. La clase de problemas **NP** para problemas de decisión incluye los problemas de decisión con verificación polinomial de la respuestas "Sí", es decir, problemas cuya salida con respuesta "Sí" se puede comprobar su corrección en tiempo polinomial. Por ejemplo, dado un grafo G y un entero K, ¿contiene G un conjunto independiente con al menos K vértices?

El resultado más destacado dentro de este área demostrado por Cook en 1971 [3] es el siguiente:

Teorema 2.1 (Teorema de Cook)**.** *Cualquier problema de la clase **NP** es polinomialmente reducible al problema de la satisfacibilidad de expresiones booleanas.*

Definición. Un problema de decisión C es **NP-Completo** si $C \in$ **NP** y cualquier otro problema $C' \in$ **NP** es reducible a C en tiempo polinomial.

Definición. La clase de problemas **NP-Dura** (**NP-Hard**) contiene los problemas de optimización tales que su versión de decisión es NP-Completa.

Vamos a demostrar que MIS se trata, en concreto, de un problema de complejidad NP-Dura. Esto es debido a que su versión de decisión pertenece a la clase NP-Completa. Por las equivalencias mostradas en el primer capítulo, VERTEX COVER y CLIQUE también serán NP-Duros.

2.2. El problema del máximo conjunto independiente es NP-Duro

La NP-Completitud de MIS fue demostrada por el informático Richard Karp en 1972 [2], a partir de los resultados de Cook (1971) [3] sobre la NP-Completitud del problema de de la satisfacibilidad de expresiones booleanas.

Definición. Sea $U = \{u_1, u_2, ..., u_n\}$ un conjunto de *variables booleanas*. Una *expresión booleana* se compone de variables booleanas combinadas mediante los operadores AND ($\wedge$), OR ($\vee$) y NOT (para negar la variable booleana u escribimos $\bar{u}$). Una *asignación de valores* para U es una función $t : U \to \{T, F\}$. Si $t(u) = T$ decimos que u es "verdadero" bajo t y si $t(u) = F$ decimos que u es "falso" bajo t. Si u es una variable de U, entonces u y su negación $\bar{u}$ son *literales* de U. Una *cláusula* sobre U es una disyunción de literales de U que se satisface bajo una asignación de valores si y solo si al menos uno de sus miembros es verdadero bajo esa asignación. Por ejemplo la expresión booleana $(u_1 \vee \bar{u_3} \vee u_8)$ es una cláusula de U que se satisface bajo t si al menos una asignación es verdadera, es decir, $t(u_1) = V$, $t(u_3) = F$ o $t(u_8) = V$ se cumplen. Una colección C de cláusulas sobre U se satisface si y solo si existe alguna asignación de valores de U que satisface todas las cláusulas de C.

El problema de decisión de la **satisfacibilidad de expresiones booleanas** (lo denotaremos SAT) consiste en, dado un conjunto U de variables booleanas y una colección de cláusulas C de variables de U, ver si existe una asignación posible de valores para sus variables que hace que la conjunción de todas las claúsulas sea verdadera.

Los problemas de 3-satisfacibilidad (3SAT) son una versión restringida de los de satisfacibilidad de expresiones booleanas en la que todas las entradas tienen exactamente tres literales por cláusula. Por su estructura más sencilla trabajaremos con este problema para probar la NP-Completitud del problema del máximo conjunto independiente.

Dado un problema A $\in$ NP, todo lo que necesitamos hacer para ver que es NP-Completo es mostrar que algún problema B $\in$ NP-Completo ya conocido puede transformarse en A en tiempo polinomial. Por ello a partir de la NP-Completitud de SAT (véase el teorema 2.1) probaremos la NP-Completitud de 3SAT y a partir de esta la de VERTEX COVER. Este resultado será equivalente a probar la NP-Completitud de MIS.

Teorema 2.2. *El problema de la 3SAT es NP-Completo.*

Demostración. El problema de decisión 3SAT $\in$ NP puesto que conocida un asignación de valores podemos facilmente verificar en tiempo polinomial si esa asignación satisface todas las cláusulas de tres literales dadas.

Para ver que 3SAT $\in$ NP-Completo basta ver que el problema SAT $\in$ NP-Completo puede reducirse polinomialmente a 3SAT. La reducción toma una entrada arbitraria ϕ de SAT y la transforma en una entrada ϕ' de 3SAT, tal que la satisfacibilidad se conserva, es decir, ϕ' se satisface si y solo si ϕ lo hace. Recordamos que una entrada de SAT es un AND de algunas cláusulas, y cada cláusula es un OR de algunos literales. Una entrada de 3SAT es una tipo especial de entrada de SAT en la que cada cláusula tiene exactamente 3 literales.

Sea $U = \{u_1, u_2, ..., u_n\}$ un conjunto de variables booleanas y $C = \{C_1, C_2, ..., C_m\}$ un conjunto de cláusulas sobre U que forman una entrada de SAT. Queremos ver que cualquier cláusula de C se puede reemplazar por una conjunción de cláusulas C' sobre un conjunto de variables U' tal que: todas las cláusulas de C' contienen exactamente 3 literales y C se satisface si y solo si C' se satisface.

- <u>caso 1</u> *Cláusulas con un literal* Sea C_j formado por un único literal z (u_i ó $\bar{u}_i$ para algún i). Sean u'_1 y u'_2 dos nuevas variables. Reemplazamos C_j por la conjunción de las siguientes cuatro cláusulas de tres literales:

  $$C'_j = (z \vee u'_1 \vee u'_2), (z \vee \bar{u_1}' \vee u'_2), (z \vee u'_1 \vee \bar{u_2}'), (z \vee \bar{u_1}' \vee \bar{u_2}')$$

  Claramente el AND lógico de las cuatro cláusulas anteriores es igual a z. Por lo tanto, la nueva fórmula obtenida al reemplazar C_j por C'_j estas cuatro cláusulas calcula la misma función booleana que la fórmula original. Por lo tanto, C_j se satisface si y solo si C'_j se satisface.

- caso 2 *Con dos literales* Sea $C_j = (z_1 \vee z_2)$ formada por dos literales z_1 y z_2 donde cada uno es una variable u_i o una variable negada $\bar{u}_i$. Sea u'_1 una nueva variable. Reemplazamos C_j por la conjunción de las siguientes dos cláusulas:

$$C'_j = (z_1 \vee z_2 \vee u'_1), (z_1 \vee z_2 \vee \bar{u_1}')$$

Se verifica que el AND lógico de las dos cláusulas anteriores es igual a $C_j = z_1 \vee z_2$ Así, la nueva fórmula obtenida al reemplazar C_j por C'_j calcula la misma función booleana que la fórmula original. Por lo tanto, C_j se satisface si y solo si C'_j se satisface.

- caso 3 *Con tres literales* En este caso ya tenemos una cláusula C_j de tres literales que se satisface, luego no hace falta reemplazarla.

- caso 4 *Con $k > 3$ literales* Sea $C_j = (z_1 \vee z_2 \vee ... \vee z_k)$ con $k > 3$ donde cada z_l es una variable u_i o una variable negada $\bar{u}_i$. Sean $u'_1, u'_2, ..., u'_{k-3}$, $k-3$ nuevas variables. Reemplazamos C_j por la conjunción de las siguientes $k-3$ cláusulas:

$$C'_j = (z_1 \vee z_2 \vee u'_1), (z_3 \vee \bar{u_1}' \vee u'_2), (z_4 \vee \bar{u_2}' \vee u'_3), ..., (z_{k-2} \vee \bar{u_{k-4}}' \vee u'_{k-3}), (z_{k-1} \vee z_k \vee \bar{u_{k-3}}')$$

A diferencia de los casos anteriores, el AND lógico de las $k-2$ cláusulas anteriores no es lo mismo que C_j. Sin embargo, para probar que C_j se satisface si y solo si C'_j se satisface se pueden probar las siguientes dos afirmaciones:

  - Cuando C_j se satisface, entonces C'_j se satisface.
  - Cuando C_j no se satisface, entonces C'_j no se satisface.

$\Rightarrow$) C_j se satisface, luego al menos un literal $z_1, ..., z_k$ es verdadero:

  - Si z_1 o z_2 son verdaderos, tomamos el resto de variables adicionales $u'_1, u'_2, ..., u'_{k-3}$ falsas. Entonces todas las cáusulas de C'_j tendrán algún término verdadero y, por tanto, C'_j se satisface.
  - Si z_k o z_{k-1} son verdaderos, tomamos el resto de variables adicionales $u'_1, u'_2, ..., u'_{k-3}$ verdaderas. Entonces todas las cáusulas de C'_j tendrán algún término verdadero y, por tanto, C'_j se satisface.
  - Si z_l con $l \notin \{1, 2, k-1, k\}$ es verdadero, tomamos las primeras $u'_1, u'_2, ..., u'_{l-3}$ verdaderas y las siguientes $u'_{l-1}, u'_2, ..., u'_{k-3}$ falsas. Entonces todas las cáusulas de C'_j tendrán algún término verdadero y, por tanto, C'_j se satisface.

Luego, cuando C_j se satisface, entonces C'_j se satisface.

$\Leftarrow$) Ahora C_j no se satisface, luego ningún literal $z_1, ..., z_k$ es verdadero. Para que C'_j se satisfaga, todas las claúsulas deben ser verdaderas. Para que la primera claúsula de la expresión C'_j sea verdadera, u'_1 debe ser verdadero, ahora, para que lo sea la segunda claúsula u'_2 debe ser verdadero, así sucesivamente:

$$u'_2 = T \rightarrow u'_3 = T ... \rightarrow u'_{k-4} = T \rightarrow u'_{k-3} = T$$

De esta forma la última claúsula de la expresión C'_j será falsa. Por lo tanto C'_j no se satisface. Llegamos a que si C_j no se satisface, entonces C'_j no se satisface.

Por lo tanto hemos probado la reducción de cláusulas de más de tres literales.

Luego cualquier claúsula en una expresión de entrada de SAT se puede remplazar en tiempo polinomial por una conjunción de claúsulas de tres literales, es decir, en una entrada de 3SAT. Así pues, SAT $\in$ NP-Completo puede reducirse polinomialmente a 3SAT $\in$ NP, por consiguiente 3SAT $\in$ NP-Completo. □

Teorema 2.3. *El problema de VERTEX COVER es NP-Completo.*

Demostración. Sabemos que VERTEX COVER está en NP ya que podemos verificar cualquier solución en tiempo polinomial con un simple recorrido de todos los ejes para examinar su incidencia en el recubrimiento de vértices dado. Para ver que VERTEX COVER$\in$ NP-Completo basta ver que 3SAT $\in$ NP-Completo puede reducirse polinomialmente a VERTEX COVER.

Sea $U = \{u_1, u_2, ..., u_n\}$ un conjunto de variables booleanas y $C = \{C_1, C_2, ..., C_m\}$ un conjunto de cláusulas sobre U que forman una entrada de 3SAT. Para la reducción, debemos construir un grafo $G = (V, E)$ y encontrar un entero positivo $k \leq |V|$ tal que G tiene un recubrimiento de vértices de tamaño menor o igual a k si y solo si C tiene una asignación de valores para sus variables que hace verdaderas todas sus cláusulas.

Lo primero que haremos es para cada $u_i \in U$ crear una componente $T_i = (V_i, E_i)$ con $V_i = \{u_i, \bar{u}_i\}$ y $E_i = \{(u_i, \bar{u}_i)\}$, es decir, un par de vértices para cada literal y su negación unidos por un eje. Entonces, un $u_i \in U$ producirá dos vértices en G:

Figura 2.1: Componente T_i

Notar que cualquier recubrimiento de vértices contendrá entonces al menos uno o u_i o $\bar{u}_i$ para cubrir el eje E_i.

A continuación, creamos una componente para representar la cláusulas. Lo que haremos es para cada $C_j \in C$ crearemos un triángulo $S_j = (V'_j, E'_j)$, que consta de tres vértices donde cada vértice representa un literal de la cláusula C_j y está conectado a los otros dos mediante ejes. Sea $C_j = (z_{j,1} \vee z_{j,2} \vee z_{j,3})$ formada por tres literales $z_{j,1}$,$z_{j,2}$ y $z_{j,3}$ donde cada uno es una variable u_i o una variable negada $\bar{u}_i$.

$$V'_j = \{z_{j,1}, z_{j,2}, z_{j,3}\}$$

$$E'_j = \{(z_{j,1}, z_{j,2}), (z_{j,2}, z_{j,3}), (z_{j,3}, z_{j,1})\}$$

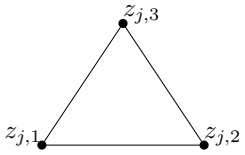

Figura 2.2: Componente S_j

Notar que cualquier recubrimiento de vértices contendrá entonces al menos dos vétices de V'_j para cubrir los ejes de E'_j.

Finalmente conectamos cada vértice de cláusula de S_j con cada vértice de variable de T_i que tenga escrito el mismo literal. Para cada cláusula $C_j \in C$, denotemos los tres literales en C_j por $z_{j,1}$,$z_{j,2}$ y $z_{j,3}$ y a_1,a_2 y a_3 los respectivos literales correspondientes de los T_i. Entonces los ejes de comunicación que salen de cada S_j para conectarlo con los T_i están dados por:

$$E''_j = \{(z_{j,1}, a_1), (z_{j,2}, a_2), (z_{j,3}, a_3)\}$$

Eso es todo lo que hay que hacer para construir G. La construcción de nuestra entrada de VERTEX COVER se completa estableciendo $k = n + 2m$ donde n es el número de literales y m el número de cláusulas y $G = (V, E)$, donde

$$V = \left(\bigcup_{i=1}^{n} V_i\right) \cup \left(\bigcup_{j=1}^{m} V'_j\right)$$

y

$$E = \left(\bigcup_{i=1}^{n} E_i\right) \cup \left(\bigcup_{j=1}^{m} E'_j\right) \cup \left(\bigcup_{j=1}^{m} E''_j\right)$$

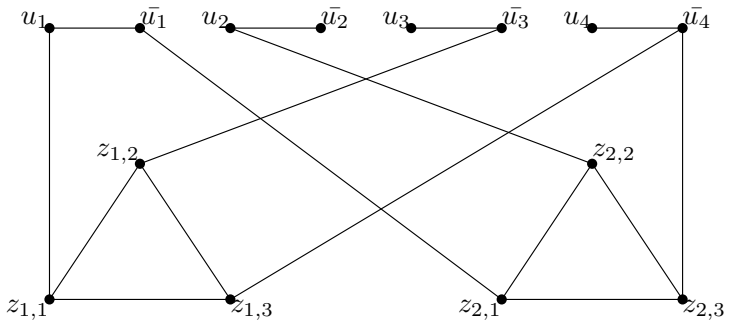

Figura 2.3: Ejemplo de entrada de VERTEX COVER generada a partir de la entrada de 3SAT: $U = \{u_1, u_2, u_3, u_4\}$, $C = \{\{u_1, \bar{u_3}, \bar{u_4}\}, \{\bar{u_1}, u_2, \bar{u_4}\}\}$. Donde $k = n + 2m = 8$

Esta transformación describe completamente la construcción de una entrada de VERTEX COVER a partir de una de 3SAT en tiempo polinomial. Pero, queda por demostrar es que C tiene asignación de valores para sus variables que hace verdaderas todas sus cláusulas si y solo si G tiene un recubrimiento de vértices de tamaño menor o igual a k.

$\Leftarrow$) Primero supongamos que $V' \subseteq V$ es un recubrimiento de vértices de G con $|V'| \leq k$. Por lo visto anteriormente, V debe contener al menos un vértice de cada T_i y al menos dos vértices de cada S_j. Esto nos da un total de al menos $n+2m = k$ vértices, luego V debe contener, de hecho, exactamente un vértice de cada T_i y dos vértices de cada S_j. Por lo tanto, podemos usar la forma en que V' intersecta cada T_i para obtener una asignación de valores $A : U \to \{T, F\}$ para C. Simplemente elegimos una asignación A tal que $A(u_i) = T$ si $u_i \in V'$ y $A(u_i) = F$ si $u_i \notin V'$. Para ver que esta asignación de valores satisface cada una de las cláusulas $C_j \in C$, considero los tres ejes de E''_j. Solo dos de esos ejes pueden estar cubiertos por vértices de $V'_j \cap V'$, por lo que uno de ellos debe estar cubierto por un vértice de algún V_i que pertenezca a V'. Esto implica que el literal correspondiente, ya sea u_i ó $\bar{u_i}$, de la cláusula C_j es verdadero bajo la asignación A, y por lo tanto la cláusula C_j se satisface bajo A. Debido a que esto es válido para todo $C_j \in C$, se deduce que C se satisface bajo A.

$\Rightarrow$) Para la otra implicación, suponemos que existe que existe $A : U \to \{T, F\}$ una asignación de valores que satisface C. El recubrimiento de vértices correspondiente V' incluye un vértice de cada T_i y dos vértices de cada S_j. El vértice de T_i en V' es u_i, si $A(u_i) = T$ y $\bar{u_i}$ si $A(u_i) = F$. Esto asegura que al menos uno de los tres ejes de cada conjunto E''_j esté cubierto, porque A satisface cada cláusula C_j. Por lo tanto, solo necesitamos añadir a V' los vértices de S_j que corresponden a los extremos de los otros dos ejes de E''_j (que pueden o no estar también cubiertos por vértices de algún T_i). Esto nos da el recubrimiento de vértices de tamaño menor o igual a k que buscabamos. □

Capítulo 3

Resultados teóricos

En el presente apartado vamos a presentar algunos resultados teóricos relacionados con el problema del conjunto independiente máximo: el Teorema de König sobre grafos bipartitos y el Teorema de Erdos-Szekeres sobre subgrafos completos.

3.1. Teorema de König

El **Teorema de König-Egerváry**, a veces simplemente llamado **teorema de König**, demostrado por los mateméticos Dénes Kőnig y Jenő Egerváry(1931), establece la relación entre el problema del recubrimiento de vértices mínimo y el problema de máximo matching para grafos bipartitos. Este teorema da un método para hallar un matching máximo en grafos bipartitos, o lo que es equivalente, para encontrar el conjunto independiente máximo de un grafo bipartito.
Antes de presentar dicho teorema enunciamos algunos resultados previos necesarios para demostrarlo, para ello vamos a tener en cuenta algunos de los conceptos que ya indroducimos en el primer capítulo como máximo matching o grafo bipartito.

Definición. Una **red de flujo** Es un grafo dirigido $G = (V,E)$ simple, conexo, con dos vértices distinguidos s (el origen) y t (el destino), donde cada eje $(i,j) \in E$ tiene asociado un número no negativo $c(i,j) \geq 0$ llamado capacidad de ese eje.

Definición. Un **flujo** sobre la red de flujo G es una aplicación f de $V \times V$ en $\mathbb{R}$ cumpliendo:

$$0 \leq f(i,j) \leq c(i,j) \quad \forall (i,j) \in V \times V$$

$$\sum_{j \in V} f(i,j) = \sum_{j \in V} f(j,i) \quad \forall i \in V - \{s,t\}$$

El **valor de un flujo** f se define como

$$\gamma(f) = \sum_{j \in V} f(s,j) - \sum_{j \in V} f(j,s)$$

es decir, se trata del flujo neto que sale de s

Definición. Dado un conjunto de vértices $X \in V$ y $\bar{X} = V - X \in V$ su conjunto complementario en G, denotamos por $(X,\bar{X})$ al conjunto de ejes que tienen el extremo inicial en algún $i \in X$ y el final en algún $j \in \bar{X}$. Dada una red de flujo G, un **corte** separando s de t, es un conjunto de ejes $(X,\bar{X})$ donde $s \in X$ y $t \in \bar{X}$. A la cantidad $c(X,\bar{X})$ le llamamos capacidad del corte $(X,\bar{X})$ y es la suma de los valores de c en esos ejes.

Definición. El problema de máximo flujo consiste en dada una red de flujo calcular un flujo f con valor $\gamma(f)$ máximo.

Definición. El problema de mínimo corte consiste en dada una red de flujo encontrar un corte con capacidad $c(X,\bar{X})$ mínima.

Teorema 3.1 (Teorema de máximo flujo-mínimo corte)**.** *Sea G un grafo de flujo. El valor máximo de $\gamma(f)$ tomando entre todos los posibles flujos f es igual al mínimo de $c(X,\bar{X})$ tomado entre todos los posibles cortes $(X,\bar{X})$.*

El teorema de máximo flujo-mínimo corte permite dar demostraciones sencillas de varios resultados clásicos en grafos. Entre ellos, el Teorema de König que probaremos a continuación.

Teorema 3.2 (Teorema de König-Egerváry)**.** *Sea $G=(V,E)$ un grafo bipartito no dirigido. Es decir, $V=S\cup T$, $S\cap T=\emptyset$, $S,T\neq\emptyset$, $E\subseteq S\times T$. Entonces, el máximo número de ejes de E que podemos hallar con extremos disjuntos (número de ejes en un matching máximo) es igual al número mínimo de vértices que hay que eliminar del grafo para hacer desaparecer todos los ejes (número de vértices en un recubrimiento de vértices mínimo).*

Demostración. Sean $s_1,...,s_n$ los nodos de S y $t_1,...,t_m$ los nodos de T. Formamos el grafo $G'=(V',E')$ con $V'=V\cup\{s,t\}$, y conjunto de ejes E' formado por los ejes de E, dirigidos desde s_i a t_j , junto con los n ejes (s,s_i) y los m ejes (t_j,t) con $i=1,...,n$ y $j=1,...,m$. En G', hallaremos el máximo flujo de s a t poniendo capacidad 1 en los $n+m$ ejes nuevos (s,s_i) y (t_j,t), y capacidad ∞ en los ejes (s_i,t_j). Sea k el valor de ese flujo y G_1 el subgrafo de G' formado por los ejes en los que circula flujo. Podemos ver lo siguiente:

1. En cada eje de G_1 el flujo que circula tiene que ser 1.
2. De s salen (en G_1) k ejes, a t llegan k ejes, y si a un vértice intermedio s_i o t_j, llega un eje con flujo, sale otro eje con flujo.
3. Como consecuencia los ejes de G_1 conectando vértices de S y vértices de T son disjuntos en sus extremos, y hay exactamente k de ellos, forman entonces un matching de tamaño k.
4. No puede haber en G un matching $\{(s_l,t_l)|l=1,...,k+1\}$ de tamaño $k+1$, pues enviando una unidad de flujo por cada camino $s\to s_{il}\to t_{jl}\to t$ obtendríamos un flujo de valor $k+1$ mayor que el flujo máximo k, llegando así a una contradicción.

Encontramos así un matching máximo para G, es decir, el número máximo de ejes de E disjuntos en sus extremos.

Además sea $(X,\bar{X})$ un corte mínimo, por el teorema de máximo flujo-mínimo corte 3.1, como el valor del máximo flujo es k entonces la capacidad de $(X,\bar{X})$ será también k. No puede haber conexiones de algún $s_i\in X$ con algún $t_j\in\bar{X}$ pues esos ejes son de capacidad ∞. Por tanto como la capacidad del corte es k, $(X,\bar{X})$ debe contener exactamente k conexiones de capacidad 1, que solo pueden estra formadas por ejes de s a nodos de $\bar{X}\cap S$ y por ejes de nodos de $X\cap T$ a t. Por consiguiente la capacidad del corte:

$$|\bar{X}\cap S|+|X\cap T|=k$$

. Si eliminamos esos k nodos de $\bar{X}\cap S$ y $X\cap T$, desaparecen todos los ejes. Con menos de k vértices no pueden desconectarse S y T pues al menos hay que borrar un extremo de cada eje del matching. Por lo tanto tengo que para nuestro grafo bipartito, el máximo número de ejes de E que podemos hallar con extremos disjuntos, es igual al número mínimo de vértices que hay que eliminar del grafo para hacer desaparecer todos los ejes. Quedando así demostrado el teorema. □

Este teorema proporciona una equivalencia entre el problema de máximo matching y el de hallar un recubrimiento de vértices mínimo en grafos bipartitos. Vemos un ejemplo de su aplicación. Partiremos del siguiente grafo bipartito $G=(V,E)$ mostrado en la parte central de la figura 3.1.. A continuación, como en la demostración del teorema, formamos el grafo $G'=(V',E')$ con $V'=V\cup\{s,t\}$, y conjunto de ejes E' formado por los ejes de E, junto con los 8 nuevos ejes de s a S y los 9 ejes de t a T (véase 3.1). En

G', hallaremos un máximo flujo máximo de s a t poniendo capacidad 1 en los ejes nuevos, y capacidad ∞ en los ejes de E. El máximo flujo $\gamma(f) = 7$ encontrado nos proporciona el máximo matching para nuestro grafo bipartito G (en rojo en la figura 3.1) Por otra parte, el corte mínimo $(X, \bar{X}) = 7$ de G' (en azul en la figura 3.1) nos proporciona un conjunto independiente de G que son precisamente los vértices de $X \cap S$ y $\overline{X} \cap T$ (en naranja en la figura 3.1).

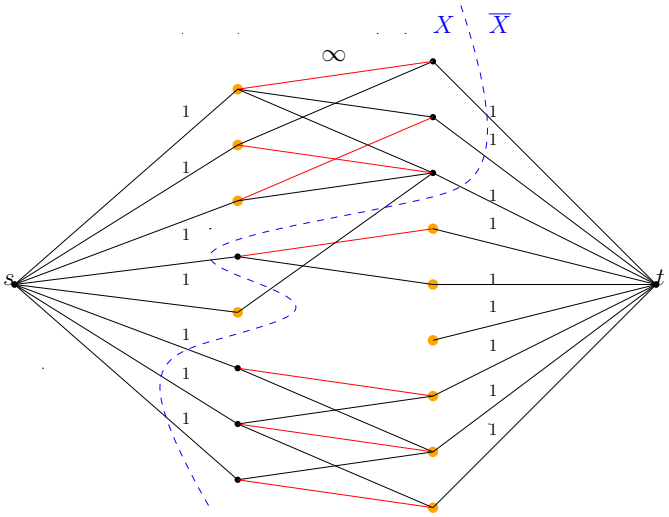

Figura 3.1: Grafo G', máximo matching de G en rojo y conjunto independiente de G en naranja

Podemos decir que, en grafos bipartitos, la resolución de problema del máximo conjunto independiente se reduce a resolver el problema polinomial del máximo matching para grafos bipartitos.

3.2. Teorema de Erdos-Szekeres

Otro resultado teórico de gran importancia y carácter elemental relacionado con este problema es el Teorema de Erdos-Szekeres.

Como ya se ha mencionado, el problema del conjunto independiente máximo es equivalente al problema de máximo clique pues un conjunto es independiente si y solo si es un clique en el grafo complementario. De hecho, si un grafo G no tiene grandes cliques, entonces uno podría esperar que G tenga un conjunto independiente grande. Esta situación se estudia dentro de la teoría de Ramsey, que dice para cualquier par de enteros s y t, definimos un entero $R(s,t)$ (*número de Ramsey*) como el menor valor tal que cualquier grafo completo de $R(s,t)$ nodos contiene o un clique de s nodos o un conjunto independiente de t nodos.

Para trabajar con este teorema vamos a considerar particiones de ejes de un grafo completo K_n, por conveniencia una partición la vamos a llamar un coloreado (rojo o azul), y un subgrafo completo de K_n será rojo o azul si todos sus ejes son, respectivamente, rojos o azules. Para ilustrar esto vamos a ver en la figura 3.2 el grafo completo K_5 con dos coloreados de 5 ejes, uno rojo y otro azul.

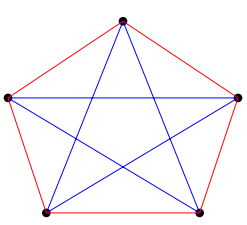

Figura 3.2: Grafo K_5 con dos coloreados

El teorema entonces establece que para cualquier coloreado aplicado a K_n suficientemente grande, se hallan subgrafos completos monocromáticos. Para dos colores: rojo y azul, dado un número natural s, existe un entero $R(s)$ tal que si $n \geq R(s)$ entonces cualquier coloreado aplicado a K_n contiene o un subgrafo rojo K_s o uno azul K_t. Para mostrar la existencia de $R(s)$ en general, para cualquier s y t, definimos $R(s,t)$ (*número de Ramsey*) como el menor valor de n para el cual cada coloreado rojo-azul de K_n genera o un subgrafo rojo K_s o uno azul K_t. Para ilustrar esto vamos a ver en la figura 3.3 los siguientes grafos K_5 y K_6 con coloreados rojo y azul. Como podemos ver en el primero de ellos, de 5 vértices, cualquier subgrafo K_3 es no monocromático, mientras que en el de 6 vértices, siempre encontraremos un subgrafo K_3 monocolor, es decir, cualquier coloreado de K_6 contiene un K_3 monocromático

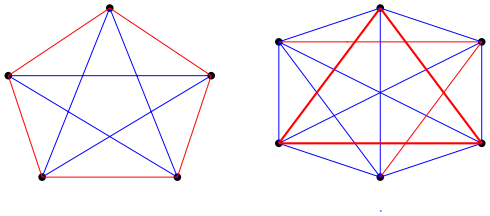

Figura 3.3: Coloreado de K_5 sin K_3 monocromático. Grafo K_6 con K_3 monocromático rojo o azul. Cualquier coloreado de K_6 contiene un K_3 monocromático

Antes de continuar con el teorema, vamos a probar algunos casos triviales que utilizaremos para demostrar el teorema.

Teorema 3.3. $\forall s,t \geq 2$

$$R(s,t) = R(t,s)$$

Demostración. Obvio por simetría. □

Teorema 3.4. $\forall s,t \geq 2$

$$R(2,t) = R(t,2) = t$$

.

Demostración. Este resultado equivale a decir que cualquier coloreado rojo-azul del grafo K_t contiene o un eje azul K_2 o un coloreado rojo del grafo completo K_t. Si tomo un grafo más pequeño K_{t-1} rojo este no contiene ni el grafo K_t pues no tiene suficientes vértices, ni un eje azul K_2, por lo tanto, $R(2,t) > t-1$. Por otra parte cualquier grafo de t vértices contiene o un eje azul K_2 o, en caso contrario, todos los ejes son rojos, es decir, contiene un K_t rojo, luego $R(2,t) <= t-1$. Combinando ambos resultados y añadiendo que $\forall s,t \geq 2$

$$R(s,t) = R(t,s)$$

llegamos a que, efectivamente,

$$R(2,t) = R(t,2) = t$$

□

El teorema de Erdos-Szekeres para subgrafos completos, extensión del teorema original de Ramsey, establece que $R(s,t)$ es finito para todo s y t y, al mismo tiempo, da una cota para $R(s,t)$. El teorema dice lo siguiente:

Teorema 3.5 (Teorema de Erdos-Szekeres)**.** *La función $R(s,t)$ es finita para todo $s,t \geq 2$ tal que cualquier grafo de $R(s,t)$ vértices contiene o un conjunto independiente de tamaño s o un clique de tamaño t. Si $s > 2$ y $t > 2$ entonces*

$$R(s,t) \leq R(s-1,t) + R(s,t-1) \tag{3.1}$$

y

$$R(s,t) \leq \binom{s+t-2}{s-1} \tag{3.2}$$

Demostración. Vemos 3.1 y 3.2

1. Ya que $R(s-1,t)$ y $R(s,t-1)$ son finitos.

   Sea $n = R(s-1,t)+R(s,t-1)$ y consideramos un coloreado de ejes de K_n con rojo y azul. Probaremos que en este coloreado hay o un coloreado rojo K_s o uno azul K_t.

   Para ello, sea x un vértice de K_n. Como $d(x) = n-1 = R(s-1,t)+R(s,t-1)-1$ entonces o tenemos $n_1 = R(s-1,t)$ ejes rojos incidentes a x ó $n_2 = R(s,t-1)$ ejes azules incidentes a x. Por simetría podemos suponer cierto el primer caso. Consideramos ahora el subgrafo K_{n_1} de K_n inducido por los n_1 vértices que están unidos a x por ejes rojos. Si K_{n_1} contiene un K_t azul, hemos terminado.

   En caso contrario, como $R(s-1,t)$ es finito, por definición de número de Ramsey, $n_1 = R(s-1,t)$ es el menor valor de para el cual cada coloreado rojo-azul de K_{n_1} genera o un subgrafo rojo K_{s-1} o uno azul K_t. Entonces el grafo K_{n_1} contiene un K_{s-1} rojo que junto con el vértice x forma un K_s rojo.

   $R(s,t)$ es finito se sigue por inducción.

2. Es claro que la desigualdad 3.2 se cumple si $t = 2$ o $s = 2$. De hecho, se trata de una igualdad, pues por los resultados triviales que demostramos previamente $\forall s,t \geq 2$, $R(2,t) = R(t,2) = t$ y, por las propiedades de los números combinatorios:

   $$\binom{2+t-2}{2-1} = \binom{t}{1} = t$$

   Por lo que, para $s = 2$

   $$R(s,t) = \binom{s+t-2}{s-1} = t$$

   Y lo mismo sucede para $t = 2$.

   Para $s > 2$, $t > 2$ suponemos que la desigualdad 3.2 se cumple para todo par (s',t') con $2 \leq s'+t' < s+t$, entonces, como por 3.1 tenemos que $R(s,t) \leq R(s-1,t)+R(s,t-1)$, la desigualdad 3.2 será cierta para $(s-1,t)$ y $(s,t-1)$ por lo que

   $$R(s-1,t) \leq \binom{s+t-3}{s-2} \qquad R(s,t-1) \leq \binom{s+t-3}{s-1}$$

   Finalmente, por las propiedades de los números combinatorios sabemos que

   $$\binom{n}{m} + \binom{n}{m+1} = \binom{n+1}{m+1}$$

   entonces tenemos que

   $$\binom{s+t-3}{s-2} + \binom{s+t-3}{s-1} = \binom{s+t-2}{s-1}$$

   Combinando todos los resultados anteriores llegamos precisamente al resultado que buscabamos:

   $$R(s,t) \leq R(s-1,t)+R(s,t-1) \leq \binom{s+t-3}{s-2} + \binom{s+t-3}{s-1} = \binom{s+t-2}{s-1}$$

□

Capítulo 4

Algoritmos exactos y heurísticos para el problema del máximo conjunto independiente

4.1. Algoritmos exactos

En este apartado buscamos algoritmos exactos que nos permitan resolver el problema del máximo conjunto independiente de un grafo. No sé conoce ningún algoritmo eficiente de tiempo polinomial para resolver el problema del máximo conjunto independiente de un grafo. De hecho, como ya hemos visto, es un problema NP-Duro. Desde la década de los 70, matemáticos e informáticos han trabajado para acortar el tiempo que se tarda en encontrar un máximo conjunto independiente para un grafo dado.

El algoritmo obvio es el algorimo de fuerza bruta, que describimos a continuación:

- Para cualquier grafo $G = (V, E)$ de $|V| = n$ existen 2^n subconjuntos de vértices de V.
- Para cada subconjunto el algoritmo comprueba como máximo $\binom{n}{2}$ ejes (el número máximo de ejes posibles en un conjunto de n vértices) para verificar si es independiente. Si lo es, entonces almacena el subconjunto y su número de independencia $\alpha(G)$.
- El algoritmo de fuerza bruta continua este proceso hasta examinar los 2^n subconjuntos de V.
- Una vez completado, toma el subconjunto más grande.

Por lo tanto, como para $c > 0 \quad \binom{n}{2} = \frac{n\cdot(n-1)}{2} \leq c \cdot n^2$, el algoritmo de fuerza bruta tardaría $O(n^2 2^n)$ en encontrar una solución óptima, podemos expresar este tiempo con la notación $O^*(2^n)$. Basta ver la función dentro de los paréntesis de la notación O-grande para ver que es exponencial y no polinomial.

Los primeros en resolver el problema en el caso general de forma exacta de una manera más eficiente que el algoritmo de fuerza bruta, fueron Tarjan y Trojanowski [10] en 1976 quienes encontraron un algoritmo que rompía con la cota trivial de $O^*(2^n)$ y podía encontrar un máximo conjunto independiente para un grafo con n vértices en tiempo $O^*(2^{\frac{n}{3}})$. Numerosos otros algoritmos se desarrollaron a partir de ese, en 1986 Robson [14] redujo el tiempo de cálculo a $O^*(2^{0{,}296n})$.

Todos estos algoritmos están basados en árboles de búsqueda con los que se representan las posibles soluciones, estos árboles se recorren utilizando reglas de ramificación y acotación, esta larga lista de reglas puede dar lugar a una tediosa y complicada tarea de análisis de cada caso. Al contrario, los algoritmos "Divide y Vencerás", resuelven un problema de forma recursiva, a partir de una colección de subproblemas del mismo de menor tamaño. A su vez, estos subproblemas se resuelven de la misma forma y así sucesivamente hasta llegar a subproblemas lo suficientemente pequeños y simples como para resolverlos directamente. En esta sección vamos a tratar con un algortimo para el problema del máximo conjunto independiente de un grafo que sigue el esquema "Divide y Vencerás". Se trata del algoritmo desarrollado por **Formin** en 2006 [11] y, al contrario que los anteriores, este algoritmo es mucho más

simple y se puede describir en unas pocas líneas. Además, reduce el tiempo de ejecución a $O^*(2^{0{,}288n})$. A fecha de hoy no hay mejor algoritmo publicado.

Antes de enunciar formalmente el algoritmo introduciremos los conceptos de *dominancia*, *plegamiento* o *folding* y *reflejo* o *mirroring* que utilizaremos más delante y que permiten reducir el problema y su descripción.

Definición. Un vértice v se dice *plegable* si su conjunto de vecinos $N(v) = \{u_1, ..., u_{d(v)}\}$ no contiene triangulos en el grafo complementario. El proceso de *plegamiento* o *folding* de un vértice v de G es el proceso de transformar G en $\bar{G} = \bar{G}(v)$ mediante:

- Añadir un nuevo vértice u_{ij} para cada par de vértices no adyacentes $u_i, u_j \in N(v)$
- Añadir ejes entre cada u_{ij} y los vértices de $N(u_i) \cup N(u_j)$
- Añadir un eje entre cada par de nuevos vértices u_{ij}
- Eliminar $N[v]$

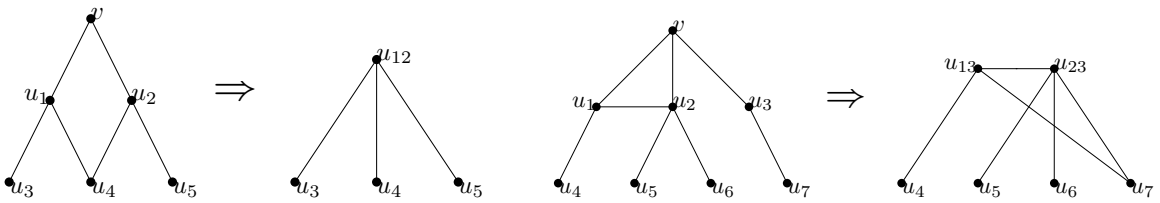

Figura 4.1: *Plegamiento* de un vértice v plegable de grado $d(v) = 2$ a la izquierda y de grado $d(v) = 3$ a la derecha.

Si *plegamos* un vértice v de grado cero o uno, simplemente eliminamos $N(v)$ del grafo. Esta estrategia se utilizará en el algoritmo solamente cuando nos permita reducir el tamaño del problema, es decir, cuando tengamos un vértice *v plegable* de grado $d(v) \leq 3$ o un vértice *v plegable* de grado $d(v) = 4$ que tenga por lo menos tres vértices no adyacentes en $N(v)$.

Lema 4.1. *Para cualquier grafo G, sea $\alpha(G)$ el tamaño del máximo conjunto independiente:*

- ***Componente conexa*** *Si G contiene una componente conexa C entonces*

$$\alpha(G) = \alpha(C) + \alpha(G - C)$$

- ***Dominancia*** *Si existen dos vértices v y w tal que $N[w] \subseteq N[v]$ decimos que w domina a v y entonces*

$$\alpha(G) = \alpha(G - \{v\})$$

- ***Plegamiento*** *Si v es un vértice plegable de G entonces*

$$\alpha(G) = 1 + \alpha(\bar{G}(v))$$

Demostración. Las dos primeras propiedades son obvias. Veamos para la tercera. Sea S el máximo conjunto independiente de G.

- Si $v \in S$, $S - v$ es un conjunto independiente de $\bar{G} = \bar{G}(v)$.
- Si $v \notin S$, entonces S, al ser de cardinalidad máxima, debe contener al menos un vértices de $N(v)$ y $S \cap N(v) \neq \emptyset$:
  - Si $S \cap N(v) = \{u\}$, $S - u$ es un conjunto independiente de $\bar{G}$.

- Si $S \cap N(v) = \{u_i, u_j\}$, para $\{u_i, u_j\}$ dos vértices no adyacentes, entonces $S \cup \{u_{ij}\} - \{u_i, u_j\}$ es un conjunto independiente de $\bar{G}$.

Se sigue entonces que $\alpha(G) \leq 1 + \alpha(\bar{G}(v))$ e igualmente se demuestra que $\alpha(G) \geq 1 + \alpha(\bar{G}(v))$

□

Definición. Sea un vértice v, un *reflejo* o *mirror* de v es un vértice $u \in N^2(v)$ ($N^2(v)$ el conjunto de vecinos de $N(v)$ denotado por $N^2(v) = \{u \in V |\ u$ es adyacente a $N(v)\}$) tal que $N(v) - N(u)$ es un clique o un conjunto vacío. Denotamos por $M(v)$ al conjunto de reflejos de v.

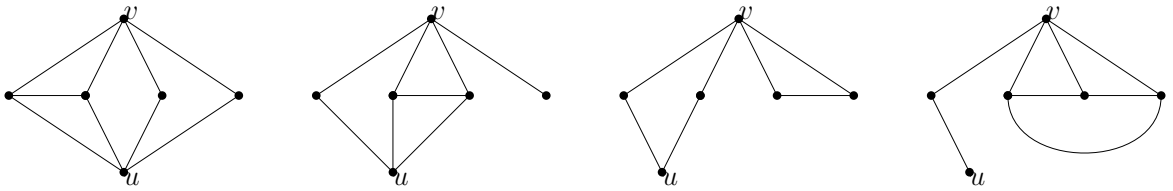

Figura 4.2: Ejemplos de *reflejos*: u es un reflejo de v.

Esta estrategia de reflejo se utiliza en el algoritmo para reducir el tamaño del problema ya que cuando descartamos un vértice v, podemos descartar sus vértices reflejos sin alterar el tamaño del máximo conjunto independiente:

Lema 4.2 (Mirroring)**.** *Para cualquier grafo G y cualquier vértice v de G,*

$$\alpha(G) = \text{máx}\, \alpha(G - \{v\} - M(v)), 1 + \alpha(G - N[v])$$

Demostración. En primer lugar , a partir de la pertenencia o no de v al máximo conjunto independientede de G tengo que: Si v pertenece al máximo conjunto independiente de G entonces $\alpha(G) = 1 + \alpha(G - N[v])$ Si v no pertenece al máximo conjunto independiente de G entonces $\alpha(G) = \alpha(G - \{v\})$ De aquí deduzco trivialmente que,

$$\alpha(G) = \text{máx}\, \alpha(G - \{v\}), 1 + \alpha(G - N[v]) \tag{4.1}$$

A partir de esta ecuación, bastaría con probar que, en el caso en el que v no está contenido en ningún máximo conjunto independiente, lo mismo ocurre con sus vértices reflejos $M(v)$. Siguiendo la demostración del lema 4.1, si ningún máximo conjunto independiente contiene a v, todo máximo conjunto independiente contendrá al menos dos vértices de $N(v)$. Considerando el reflejo $u \in M(v)$ de v, todo máximo conjunto independiente contiene al menos un nodo de $N(v) - N(u)$ (que es un clique por suposición), por lo tanto todo máximo conjunto independiente contendrá al menos un nodo en $N(v) \cap N(u) \subseteq N(u)$. De aquí se sigue entonces que $u \in M(v)$ reflejo de v no está contenido en ningún máximo conjunto independiente, como queríamos probar. Luego podemos reescribir la ecuación 4.1 como,

$$\alpha(G) = \text{máx}\, \alpha(G - \{v\} - M(v)), 1 + \alpha(G - N[v])$$

□

4.1.1. Algoritmo de Formin

Una vez explicadas las reglas de reducción de *dominancia*, *plegamiento* o *folding* y *reflejo* o *mirroring* pasamos a abordar el algoritmo de Formin en detalle, cuyo objetivo es buscar el máximo conjunto independiente de un grafo dado.

Partimos de un grafo $G = (V, E)$ de n vértices. Como se trata de un algoritmo recursivo, vamos a definir cada llamada a nuestro algoritmo como *MIS* (*Maximum Independent Set*, $MIS(G)$ devolverá el máximo conjunto independiente del grafo G.

(0) En primer lugar, si G se trata de un grafo vacío,

$$MIS(G) = \emptyset$$

En caso contrario, el algoritmo trata de reducir el problema mediante alguna de las tres estrategias del lema 4.1:

(1) Si se trata de un grafo no conexo y contiene una **componente conexa** C el resultado será la suma de resolver los subproblemas inducidos por C y $G-C$ por separado, es decir,

$$MIS(G) = MIS(C) \cup MIS(G-C)$$

(2) Si existen dos vértices v y w tal que $N[w] \subseteq N[v]$, w domina a v y entonces el agoritmo descarta v:

$$MIS(G) = MIS(G - \{v\})$$

(3) Si no se cumple ninguno de los casos anteriores y existe un vértice plegable de grado $d(v) \leq 3$ o un vértice v plegable de grado $d(v) = 4$ que tenga como mucho tres vértices no adyacentes en $N(v)$ (o lo que es lo mismo, como mucho tres ejes en el complementario de $N(v)$), el algoritmo actúa seleccionando un vértice v de grado mínimo que cumpla estas restricciones y aplicándole la estrategia de plegamiento transformando G en $\bar{G} = \bar{G}(v)$, entonces,

$$MIS(G) = \{v\} \cup MIS(\bar{G}(v))$$

Como ya mencionamos, esta estrategia solo tiene sentido utilizarla cuando nos permita reducir el tamaño del problema, de ahí las restricciones aplicadas a los vértices que plegamos. Profundizaremos más en esto durante el análisis posterior del algoritmo.

(4) Finalmente, como última opción, el algoritmo selecciona un vértice v de grado máximo y se ramifica aplicando el lema 4.2:

$$MIS(G) = \text{máx}\{MIS(G - \{v\} - M(v)), \{v\} \cup MIS(G - N[v])\}$$

4.1.2. Análisis del algoritmo de Formin

En este apartado analizaremos la complejidad en sentido asintótico del algoritmo de Formin para el problema del máximo conjunto independiente de un grafo $G = (V, E)$ de n vértices de entrada. Para ello utilizaremos el número de nodos como una medida del tamaño del problema, es decir, $k = k(G) = n$. Sea $P[k]$ el número total de hojas en el árbol de búsqueda generado para resolver un problema de tamaño k.

- Para el caso (0) del algoritmo podemos suponer $P[0] = 1$. De hecho, cuando $k = 0$, el algoritmo resuelve el problema sin bifurcarse (en tiempo polinomial).

  Si se cumple una de las condiciones de los pasos (1), (2) y (3), $P[k] \leq P[k-1]$.

- Para el caso (1) suponiendo que existe componente conexa C de tamaño k_1, es claro que el tamaño de $G-C$ será $k_2 = k - k_1$ y entonces $P[k] \leq P[k_1] + P[k_2]$

- Para el caso (2) en el que aplicamos dominancia, generamos un único subproblema, el tamaño de este subproblema decrecerá en una unidad pues estamos descartando un vértice v y por tanto $P[k] \leq P[k-1]$.

- Para el caso (3) en el existe un vértice plegable de grado $d(v) \leq 3$ o un vértice v plegable de grado $d(v) = 4$ que tenga como mucho tres vértices no adyacentes en $N(v)$, vemos que también $P[k] \leq P[k-1]$. Sea $N(v) = \{u_1, u_2, ..., u_{d(v)}\}$. Es claro que cuando $d(v) = 0$ o $d(v) = 1$ $P[k] \leq P[k-1]$ es trivialmente cierto ya que simplemente eliminamos $N[v]$ del grafo. Cuando $d(v) = 2$,

u_1, u_2 deben ser no adyacentes, y plegando v eliminamos $N[v]$ e introducimos un único vértice u_{12} reduciendo así el número de vértices. Cuando $d(v) = 3$, analizando los casos posibles, vemos que $N(v)$ debe contener exactamente un eje. Pongamos que este es (u_1, u_2), y el resto de vértices de $N(v)$ son no adyacentes. Si plegamos v, eliminamos $N[v]$ e introducimos dos nuevos vértices u_{13}, u_{23} reduciendo así el número de vértices, y por tanto $P[k] \leq P[k-1]$ Cuando $d(v) = 4$ y existen como mucho tres vértices no adyacentes en $N(v)$ (como mucho tres ejes en el complementario de $N(v)$). Asumiendo que $N(v)$ contiene dos ejes no incidentes en $N(v)$, pongamos que son (u_1, u_2) y (u_3, u_4), vemos que, como no puede haber cuatro ejes en el complementario de $N(v)$, debe haber un tercer eje incidente a (u_1, u_2) y (u_3, u_4), pongamos (u_2, u_3). En el peor de los casos (no existen más ejes), cuando plegamos v añadimos tres nodos u_{13}, u_{24} y u_{14} reduciendo así el número de vértices y por tanto $P[k] \leq P[k-1]$.

Este no es siempre el caso cuando $d(v) > 4$ o cuando $d(v) = 4$ y existen más de tres vértices no adyacentes en $N(v)$ de hecho, en estos casos, añadimos más vértices nuevos que los que eliminamos, aumentando así el tamaño del problema, por lo que no tiene sentido aplicar este tipo de reducción en estos casos.

- Para el resto de casos posibles, es decir el caso (4), considero el vértice v de grado máximo en el que vamos a ramificar el problema. Necesariamente $d(v) \geq 3$, ya que todos los casos con $d(v) \leq 2$ quedan cubiertos en los pasos anteriores.

  Sea $d(v) = 3$, si descartamos v, descartamos un reflejo de v o plegamos un vecino w de v en el siguiente paso (ya que $d(w) = 2$ después de plegar y eliminar v). En ambos casos, disminuimos el número de vértices en al menos dos. Si no descartamos v y lo incluimos, eliminamos $N[v]$, donde $|N[v]| = 4$. Esto conduce a $P[k] \leq P[k-2] + P[k-4]$.

  Suponemos ahora que $d(v) \geq 4$. En el peor de los casos, v no tiene reflejos ($M(v) = \emptyset$). Cuando descartamos o tomamos v, eliminamos al menos uno o cinco nodos, respectivamente. Por tanto, $P[k] \leq P[k-1] + P[k-5]$.

  En los peores casos posibles, tenemos entonces dos recurrencias homogéneas para $k \in \mathbb{N}$:

$$P[k] \leq P[k-1] + P[k-5]$$

$$P[k] \leq P[k-2] + P[k-4]$$

  Podemos resolverlas y hallar una fórmula explícita para conocer el término n-ésimo. Sus polinomios carácterísticos respectivos son:

$$x^5 - x^4 - 1$$

$$x^4 - x^2 - 1$$

  Resolvemos ambos polinomios con *SageMath* hallando las siguientes 5 raíces para el polinomio $x^5 - x^4 - 1$:

$$x = -\frac{1}{2}\left(\frac{1}{18}\sqrt{23}\sqrt{3} + \frac{1}{2}\right)^{\frac{1}{3}}\left(i\sqrt{3}+1\right) - \frac{-i\sqrt{3}+1}{6\left(\frac{1}{18}\sqrt{23}\sqrt{3}+\frac{1}{2}\right)^{\frac{1}{3}}},$$

$$x = -\frac{1}{2}\left(\frac{1}{18}\sqrt{23}\sqrt{3} + \frac{1}{2}\right)^{\frac{1}{3}}\left(-i\sqrt{3}+1\right) - \frac{i\sqrt{3}+1}{6\left(\frac{1}{18}\sqrt{23}\sqrt{3}+\frac{1}{2}\right)^{\frac{1}{3}}},$$

$$x = \left(\frac{1}{18}\sqrt{23}\sqrt{3} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{1}{3\left(\frac{1}{18}\sqrt{23}\sqrt{3}+\frac{1}{2}\right)^{\frac{1}{3}}},$$

$$x = -\frac{1}{2}i\sqrt{3} + \frac{1}{2}, \qquad x = \frac{1}{2}i\sqrt{3} + \frac{1}{2}$$

También podemos hallar aproximaciones numéricas de las soluciones:

$$-0{,}66235898-0{,}56227951i, \quad -0{,}66235898+0{,}56227951i,$$
$$1{,}3247180, \quad 0{,}50000000-0{,}86602540i, \quad 0{,}50000000+0{,}86602540i$$

De la misma forma hallamos las 4 raíces del polinomio x^4-x^2-1 con *SageMath*:

$$x=-\sqrt{\frac{1}{2}\sqrt{5}+\frac{1}{2}}, \qquad x=-\sqrt{-\frac{1}{2}\sqrt{5}+\frac{1}{2}},$$
$$x=\sqrt{\frac{1}{2}\sqrt{5}+\frac{1}{2}}, \qquad x=\sqrt{-\frac{1}{2}\sqrt{5}+\frac{1}{2}}$$

Y sus aproximaciones numéricas:

$$-1{,}2720196, \quad 6{,}8438531\times10^{-10}-0{,}78615138i,$$
$$1{,}2720196, \quad -6{,}8438531\times10^{-10}+0{,}78615138i$$

Concluimos entonces que $P[k]\leq\alpha^k$, donde α será la raíz más grande de los polinomios (x^5-x^4-1) y (x^4-x^2-1). Vemos entonces que

$$\alpha=\left(\frac{1}{18}\sqrt{23}\sqrt{3}+\frac{1}{2}\right)^{\frac{1}{3}}+\frac{1}{3\left(\frac{1}{18}\sqrt{23}\sqrt{3}+\frac{1}{2}\right)^{\frac{1}{3}}}=1{,}3247180...<2^{0{,}406}$$

Dado que en cada paso el tamaño de los grafos generados disminuye en al menos uno, la profundidad del árbol de búsqueda es como máximo n. Además, la resolución de cada subproblema, sin considerar las posibles llamadas recursivas, requiere tiempo polinomial. Por tanto, la complejidad en tiempo del algoritmo es $O^*(P[n])=O^*(2^{0{,}406\cdot n})$.

Cuando medimos el tamaño de entrada de un problema de máximo conjunto independiente con el número de vértices, no tenemos en cuenta cuenta el hecho de que disminuir el grado de un vértice v tiene un impacto positivo en el progreso del algoritmo. Teniendo en cuenta esta consideración, se puede realizar un análisis más refinado y complicado llegando a obtener una complejidad de $O^*(2^{0{,}288\cdot n})$ para este algortimo, no obstante este análisis largo y complicado se escapa de los objetivos de este TFG [11].

4.2. Algoritmos heurísticos

Al tratarse de un problema NP-Duro, en muchos casos es mejor utilizar algoritmos que debido a la complejidad del problema descartan encontrar soluciones óptimas y se centran en buscar buenas soluciones lo más cercanas posibles a la óptima. En esta sección introduciremos los algoritmos heúristicos y de aproximación y veremos que este problema se trata, de hecho, de un problema que no se puede aproximar a un factor constante en tiempo polinomial. Antes de ello, enunciaremos algunas definiciones:

Definición. Los **algoritmos heurísticos** son algoritmos que producen soluciones válidas, no muy alejadas de la óptima pero sin ninguna garantía de optimalidad y, a su vez, por lo general tienen un tiempo de ejecución mucho menor.

Definición. Los **algoritmos de aproximación** son algoritmos que producen soluciones aproximadas, que si bien no son la óptima, se puede "medir" cuán cerca están de la óptima, es decir, soluciones cuyo costo está dentro de un factor de α veces la solución óptima, donde $\alpha>1$ se denomina factor de aproximación del algoritmo.

Para el caso de MIS, al ser un problema de maximización, esto significa que las soluciones deben ser al menos $\frac{1}{\alpha}$ veces el valor de la solución óptima. Eso es, sea G un grafo de n nodos, $\exists 1 \leq k \leq n$ tal que $OPT(G) = k$, es el tamañano del máximo conjunto independiente, entonces las soluciones dadas por un algoritmo de aproximación de razón α serían $A(G) = \frac{k}{\alpha}$, es decir, un conjunto independiente no "mucho más pequeño"que el máximo.

Aunque muchos problemas NP-duros pueden ser eficientemente aproximados a un factor constante en tiempo polinomial, ciertos NP-Duros como el problema del máximo conjunto independiente no pueden a menos que P = NP, esta igualdad es la principal pregunta aún sin respuesta en la teoría de la computación.

Pese a que no existen algoritmos de aproximación de factor α constante de tiempo polinomial para este problema, si existen algoritmos heúristicos rápidos pero que brindan soluciones que no necesariamente son la óptima, ni podemos medir cuán lejos de la óptima están. En esta parte, veremos una heurística para abordar el problema del máximo conjunto independiente:

Asumimos que los vértices aislados han sido eliminados pues son parte trivial de cualquier conjunto independiente máximo. Sea $G = (V,E)$ un grafo dado de n vértices el algoritmo construye un conjunto independiente I repitiendo la siguiente estrategia comenzando con $G' = G$:

- Seleccionar el vértice v con menor grado en G'
- Añadir v a I
- Eliminar v y $N(v)$ de G'

Este algoritmo se ejecuta en tiempo polinomial y da como resultado un conjunto independiente maximal I, si bien este no tiene porque ser el máximo conjunto independiente.

4.3. Formulación como un problema PLE

Para finalizar, vemos que MIS se puede formular también como un problema de programación lineal entera con variables 0-1. Sea un grafo $G = (V,E)$ de n vértices, cada variable $x_i \in \{0,1\}$ con $i = 1,...,n$ representa un vértice $v_i \in V$ y $x_i = 1$ indica si v_i se elige en el conjunto independiente o no. Tenemos restricciones que establecen que para cada eje (v_i, v_j) solo uno de los vértices se puede elegir, v_i ó v_j. Por consiguiente, el problema del máximo conjunto independiente es equivalente al problema PLE:

$$\begin{cases} \text{máx} & \sum_{i=0}^{n} x_i \\ \text{s.t.} & x_i + x_j \leq 1 \quad \forall (v_i, v_j) \in E \\ & x_i \in \{0,1\} \end{cases} \tag{4.2}$$

Sin embargo, como antes, este problema tampoco se puede resolver en tiempo polinomial y el programa lineal relajado, en general, no da una buena aproximación a la solución entera óptima.

4.4. El problema del máximo conjunto independiente en grafos de intervalos

Pese a que encontrar un conjunto independiente máximo es NP-Duro en grafos generales, para algunos tipos de grafos como los grafos de intervalos el problema de máximo conjunto independiente se puede resolver en tiempo polinomial. Esta situación aparece en uno de los casos más comunes en la vida real que son los problemas de *planificación* o *scheduling*.

Definición. Un grafo no dirigido $G = (V,E)$ es un **grafo de intervalo** si está formado a partir de una familia de intervalos en la recta real $I_i \quad i = 0,1,...$ tomando un vértice v_i de cada intervalo I_i y conectando cada par de vértices v_i y v_j si y solo si los intervalos I_i e I_j interseccionan, es decir,

$$E = \{(v_i, v_j) | I_i \cap I_j \neq \emptyset\}$$

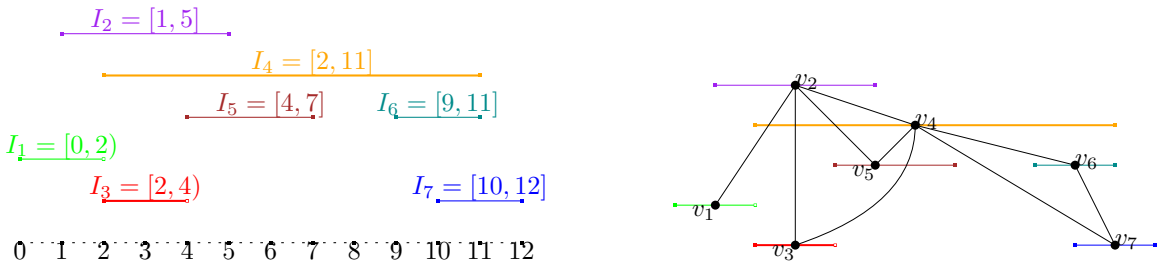

Figura 4.3: *Un conjunto de intervalos de la recta real y el grafo de intervalo definido por él*

Sea $G = (V,E)$ un grafo de intervalo de n vértices formado a partir del conjunto de intevalos I_i $i = 1,...,n$ el problema del máximo conjunto independiente consiste en hallar el subconjunto de tamaño máximo de intervalos que no se superponen en I_i.

Existe un algoritmo exacto y sencillo para este problema:

Sea I el conjunto de n intevalos.

- Ordenar los intervalos numerándolos en función de su extremo inicial $I = \{I_1,...,I_n\}$. Tomar como máximo conjunto independiente el conjunto vacío $S = \emptyset$.
- Examinar cada intervalo de I en orden inverso $I_n, I_{n-1}, ..., I_1$.
- Para cada I_i añadir I_i al máximo conjunto independiente S si ninguno de los intervalos en S interseccIona con I_i (basta comprobar que I_i no interseccIona con el último intervalo añadido a S).

Demostración. Queremos probar que existe un máximo conjunto independiente de intervalos que incluye al intervalo que comienza más tarde, I_n . Para ello, vamos a suponer el caso contrario y llegamos a una contradicción.

- Sea I_n el intervalo que empieza más tarde de I.
- Sea S un máximo conjunto independiente proporcionado por el algoritmo con $I_n \notin S$. Sea $I_k \in S$ el intervalo del conjunto independiente que empieza más tarde de entre todos los de S, como los intervalos están numerados por su extremo izquierdo es claro que $k \leq n$.
- Cualquier intervalo en S finaliza antes que el extremo izquierdo de I_k, pues S es conjunto independiente, por tanto también finalizan antes que el extremo izquierdo de I_n.
- Si $I_k \cap I_n = \emptyset$, $S' = S \cup I_n$ nos da un conjunto independiente mayor que S, contradicción.
- En caso contrario, si $I_k \cap I_n \neq \emptyset$, $S' = (S - \{I_k\}) \cup I_n$ nos da un conjunto independiente de igual tamaño.
- Por lo tanto, podemos suponer que $I_n \in S$.
- Por consiguiente, podemos descartar los intervalos $I_{n-1}, I_{n-2}...$ que interseccionan con I_n y añadir a S el primer I_i que no interseccIona con I_n. Repetimos este mismo proceso.

□

Es fácil ver que el algoritmo proporciona una solución exacta en tiempo lineal (una vez ordenados los intervalos) para el máximo conjunto independiente en grafos de intervalo.

4.4.1. El problema de scheduling

Los problemas de *planificación* o *scheduling* representan un caso muy especial del problema del máximo conjunto independiente de un grafo donde las entradas del problema se limitan a grafos de intervalo y puede ser aplicado en numerosos ámbitos. Son problemas en los que se nos asigna un conjunto de recursos (como una unidad de procesamiento de un sistema informático, el rodaje de una película, un aula para una clase) durante un período de tiempo específico. Cada intervalo representa una solicitud de un recurso por un período específico de tiempo. Buscamos encontrar un conjunto máximo de solicitudes que se puedan realizar sin interferir unas con otras.

Pongamos por ejemplo que a un actor se le ofrece un conjunto de películas con un determinado tiempo de rodaje (mes de inicio y duración). Se deben seleccionar las películas que el actor debe hacer para que el actor pueda hacer la mayor cantidad de películas en el año, sin superposiciones. Es decir, tenemos un conjunto I de n intervalos en la recta real, en el que cada intervalo representa el rodaje de una película, y queremos saber cuál es el subconjunto más grande de intervalos que no se superponen entre sí que se puede seleccionar de I, es decir, aquel con la mayor cantidad de películas. Utilizamos el grafo de la figura 4.3. Sea $G = (V, E)$ un grafo de intervalo de 7 vértices, aplicamos el algoritmo anterior, encontrando el máximo conjunto independiente S mostrado en rojo en la figura 4.4.

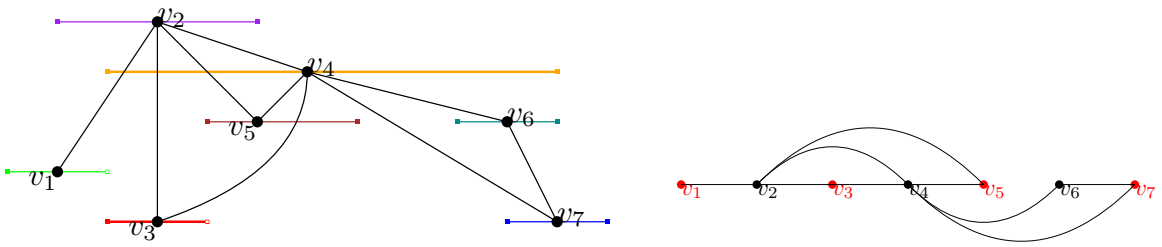

Figura 4.4: *grafo de intervalo G a la izquierda, a la derecha quedan ordenados los vértices en función de su extremo izquierdo del intervalo y muestro el máximo conjunto independiente S de G*

Encontramos entonces en tiempo polinomial el conjunto más grande de películas que el actor puede hacer en un año $S = \{v_7, v_5, v_3, v_1\}$. De esta forma tan sencilla conseguimos que un problema NP-Duro para grafos generales pueda resolverse en tiempo polinomial para esta clase de grafos.

Bibliografía

[1] J. A. BONDY Y U. S. R. MURTY, *Graph Theory with Applications*, Elsevier, New York, 1976.

[2] RICHARD M. KARP, *Reducibility among combinatorial problems*, University of California at Berkeley, 1972.

[3] S. COOK, *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium, page 151–158. New York, ACM, 1971.

[4] M.R. GAREY Y D.S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness.*, Freeman, 1979.

[5] A. GARCÍA, *Teoría de grafos*, Universidad de Zaragoza.

[6] T. BIEDL, *Graph-theoretic algorithms. Lecture notes of a graduate course*, University of Waterloo, 2005.

[7] T. H. CORMEN, C. E. LEISERSON, RONALD L. RIVEST, Y C. STEIN., *Introduction to Algorithms, 2nd edition*, MIT Press, McGraw-Hill Book Company, 2000.

[8] B. BOLLOBÁS, *Modern Graph Theory*, Springer Science & Business Media, 1998.

[9] UNIVERSITY OF OTTAWA, *The independent-set problem and an application*, disponible en `https://www.site.uottawa.ca/~lucia/courses/4105-02/a2.pdf`.

[10] R. E. TARJAN Y A. E. TROJANOWSKI, *Finding a maximum independent set*, Stanford University, June 1976.

[11] F. FOMIN, F. GRANDONI Y D. KRATSCH, *Measure and Conquer: A Simple $O(2^{0,288n})$ Independent Set Algorithm*, Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, 2006.

[12] BRENDA S. BAKER, *Approximation algorithms for NP-complete problems on planar graphs*, Journal of the ACM 41, 1, page 153–180, 1994.

[13] S. BUTENKO, *Maximum independent set and related problems, with applications*, Ph.D. Dissertation. University of Florida, USA., 2003.

[14] J.M ROBSON, *Algorithms for maximum independent sets*, Journal of Algorithms, Volume 7, Issue 3, page 425-440, 1986.